System Verification and Validation Plan for MES-ERP

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

April 4, 2025

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Nov 4, 2024	1.1	Housam: Add parts to section 3 (plan)

Contents

List of Tables

List of Figures

N/A

1 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
RBAC	Role-Based Access Control
OCR	Optical Character Recognition
UTC	Coordinated Universal Time
MFA	Multi-Factor Authentication

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

This document outlines the Verification and Validation (V&V) Plan for the McMaster Engineering Society Custom Financial Expense Reporting Platform. The V&V plan will ensure that the software performs as expected, meets user requirements, is secure, and reliable. The primary goals of this document are to define the objectives, methods, and scope of the V&V process, covering aspects such as requirements validation, design verification, and system testing.

2 General Information

2.1 Summary

The MES Custom Financial Expense Reporting Platform is a web-based application designed to streamline financial operations for McMaster Engineering Society and its associated student groups. It enables users to submit, review, and track reimbursement requests, create budgets, and manage expenses in a transparent, auditable, and efficient manner. This V&V Plan outlines the verification and validation processes for ensuring the accuracy, functionality, and security of the software.

2.2 Objectives

The primary objective of this V&V Plan is to confirm that the MES Custom Financial Expense Reporting Platform meets its requirements and operates as intended. The plan aims to:

- Build confidence in the software's functional correctness and ensure it meets all outlined requirements.
- Validate usability by demonstrating a clear, intuitive, and user-friendly interface for all key user roles.
- Ensure system security and data integrity, especially regarding sensitive financial data and authorization levels.

The plan does not prioritize nonfunctional aspects such as aesthetic design consistency or performance optimizations beyond standard system requirements, as these are not critical to initial functionality and budget constraints.

2.3 Challenge Level and Extras

1. **Usability Testing**
   Conduct formal usability testing with a variety of stakeholders (students, financial staff) to refine the interface and ensure a smooth user experience.

2. **User Documentation**
   Create comprehensive user documentation in the form of written documentation and video tutorials that guides end-users through every step of submitting expenses, reviewing budgets, and navigating the platform.

The challenge level for this project is classified as **General**, aligning with the expectations outlined in the problem statement. The team has chosen to address the critical needs of verification of functionality, security, and usability. The V&V Plan includes specific testing for the Usability extra, as detailed in Section 4.2 and Appendix A. ~~No additional extras beyond the required project scope are included in this V&V Plan.~~

2.4 Relevant Documentation

The following documents are relevant to this V&V plan and provide a basis for verification activities:

- **Software Requirements Specification (SRS)** (**?**): Defines the functional and nonfunctional requirements of the system, serving as a primary source for validation and test case development.
- **Hazard Analysis**(**?**): Outlines potential system risks and serves as a guide for security testing and fault tolerance validation.
- **Problem Statement**(**?**): Provides an overview of the project's purpose, goals, and core functionality requirements, detailing the needs of the McMaster Engineering Society for a streamlined financial reporting platform. It identifies key problems in the current system, justifying the need for an automated, digital solution to enhance efficiency and reliability in financial processes.

- **Development Plan**(**?**): Outlines the project's development lifecycle, including phases for requirements gathering, design, implementation, and testing. It establishes milestones, assigns roles, and defines the timeline, while ensuring that the team follows best practices in Agile development, with iterative check-ins for verification and validation of functional requirements.

- **Module Interface Specification (MIS)** (**?**): Provides the detailed design of each software module, including exported functions and data types, necessary for unit testing.

These documents establish the foundation for testing activities, allowing each verification and validation activity to target specific functional areas, requirements, or potential risk zones.

3 Plan

This section outlines the Verification and Validation (V&V) process for the MES Custom Financial Expense Reporting Platform, which aims to provide reliable financial management for MES and its associated student groups. The plan is structured to cover team roles, verification of the Software Requirements Specification (SRS), design verification, implementation verification, and verification of the V&V plan itself. Each subsection provides an approach tailored to the critical functional and non-functional requirements outlined in the SRS.

3.1 Verification and Validation Team

The V&V team is composed of designated members, each assigned specific responsibilities to ensure thorough verification of functionality, usability, security, and data integrity. The table below describes each team member's role and responsibilities:

Name	Role	Responsibilities
Housam	V&V Lead	Oversees all V&V activities, ensures that testing is aligned with MES requirements, and coordinates communication between team members and stakeholders. Reviews testing documentation for completeness and accuracy. Focuses on overall V&V Plan verification.
Rachid	Functional Tester	Manages the testing of all functional requirements, such as reimbursement submission, payment tracking, and budget management. Ensures that each feature meets specified criteria and conducts integration tests for module interactions. Focuses on System Tests (Section 4) and SRS Verification.
Taaha	Security Tester	Responsible for verifying data security and access controls. Conducts tests to validate role-based access, secure data handling, and compliance with MES privacy policies. Implements static and dynamic security tests. Focuses on Security Tests (Section 4.2) and Implementation Verification (Static Analysis, Code Reviews).
Sufyan	Usability Tester	Focuses on the user ex-

Each team member will participate in weekly meetings to discuss V&V progress, address issues, and adapt the V&V approach as needed. The V&V Lead will compile findings from each member and provide consolidated feedback to the project supervisor and stakeholders for continuous improvement.

3.2 SRS Verification Plan

The SRS document is critical for establishing a clear and agreed-upon set of requirements for the MES Custom Financial Expense Reporting Platform. Our SRS verification plan will ensure the SRS's accuracy, completeness, and alignment with MES's objectives. The following approaches will be used for verifying the SRS:

- **Peer Review**: Team members and classmates will conduct an initial peer review of the SRS document. Reviewers will assess the clarity and completeness of requirements, noting ambiguities or missing details. Feedback will be documented, and revisions will be made to address identified issues.
- **Stakeholder Review**: The SRS will be presented to MES stakeholders (VP Finance, select club leaders) in a structured review meeting. During this meeting, we will walk stakeholders through each requirement, prioritizing critical functionality such as reimbursement processing, budget management, and notification systems. Specific feedback will be sought on:
  - Completeness of functional requirements, such as the need for real-time updates on reimbursement statuses.
  - User accessibility and usability, focusing on ease of submission for student leaders.
  - Compliance with MES financial policies, including secure audit trails and access controls.
- **Supervisor Feedback**: The project supervisor will review the SRS as part of a task-based inspection. The supervisor will be provided with a checklist of essential aspects to verify, including:
  - Consistency in terminology and definitions.

  - Logical flow between sections, ensuring that each requirement aligns with MES's needs.
  - Identification of non-functional requirements related to performance, scalability, and security.

  Feedback from the supervisor will be reviewed, and adjustments will be made to enhance clarity and accuracy.

- **SRS Checklist**: A checklist based on the template in `docs/Checklists/SRS-Checklist.tex` will be used to ensure all requirements are accounted for. The checklist will include criteria such as:

  - Traceability: Each requirement must map back to a specific use case or MES need.
  - Testability: Each requirement must be written in a way that makes it possible to validate through specific tests.
  - Clarity and conciseness: Requirements should be free from ambiguous language and be concise enough for straightforward implementation.

3.3 Design Verification Plan

To confirm that the design (documented in the MG and MIS) aligns with the SRS and fulfills MES's functional and non-functional requirements, a structured design verification process will be conducted. The design verification plan includes the following elements:

- **Design Walkthroughs**: The team will conduct regular design walkthroughs, reviewing key components and workflows (e.g., module interactions for request approval, data flow for analytics). During these sessions, each module (e.g., reimbursement submission, audit logging, budget tracking) will be analyzed against the SRS to ensure alignment. Feedback from walkthroughs will be documented, and modifications will be made to address any identified issues.

- **Peer Reviews**: Team members will conduct detailed reviews of each other's design documentation (MG and MIS), checking for adherence to design principles (e.g., information hiding), completeness, and alignment with MES's goals. This review will include:

  - Checking that all use cases are addressed in the design.
  - Verifying that interactions between modules support seamless user experience and efficient functionality.
  - Ensuring that security measures (e.g., permission checks) are considered in the design for handling sensitive financial data.

- **Checklists for Design Verification**: Each design element (module in MG/MIS) will be assessed using a checklist (based on `docs/Checklists/MG-Checklist.tex` and `docs/Checklists/MIS-Checklist.tex`), covering:

  - Consistency with SRS requirements.
  - Scalability and extensibility of the design.
  - Feasibility of implementation within the project timeline and resources.
  - Compliance with MES standards, such as maintaining an intuitive user interface.

  Checklists will help maintain thoroughness and consistency across the team's design review efforts.

- **Supervisor Feedback**: After each design phase, the design documentation will be reviewed by the project supervisor. The supervisor's review will focus on verifying that the design meets project expectations for functionality, scalability, and security. A checklist will be provided to the supervisor to guide the review, focusing on aspects such as:

  - Alignment of the design with SRS requirements.
  - Feasibility of the design for expected usage levels by MES stakeholders.
  - Identification of any areas where design adjustments are needed to meet MES standards.

  Feedback from the supervisor will be integrated into the design documentation.

3.4 Verification and Validation Plan Verification Plan

The V&V plan itself requires verification to ensure that it is both feasible and comprehensive. This verification process will involve the following steps:

- **Peer Review**: Team members will review the V&V plan for logical flow, completeness, and feasibility. The reviewers will check that each component of the V&V plan aligns with the requirements in the SRS and that planned activities are feasible within the project timeline. Specific checks will ensure test cases have sufficient detail (inputs/outputs) and cover both functional and non-functional aspects, including extras like usability.
- **Mutation Testing**: ~~Selected sections of the V&V plan will undergo mutation testing to identify weaknesses or gaps in test case descriptions. This will help us refine test cases to ensure they cover a wide range of possible inputs and edge cases, making the V&V plan more robust.~~ Mutation testing will not be applied to the plan itself, but this concept will inform test case design, aiming to create tests that would catch common faults (mutants) if introduced into the code.
- **Checklist for V&V Plan Review**: A checklist (based on `docs/Checklists/VnV-Checklis` will guide the review of the V&V plan, covering:
  - Presence of all required V&V components, such as unit, integration, and system tests addressing both functional and non-functional requirements.
  - Feasibility of planned activities, given project resources and constraints.
  - Traceability between test cases and SRS requirements, ensuring comprehensive coverage.
  - Alignment of verification techniques with the specific needs of the MES platform.
  - Specificity of test inputs and expected outputs.
  - Inclusion of tests for extras (Usability) and associated appendices (e.g., survey).

- **Feedback from Supervisor and Stakeholders**: A meeting will be held with the project supervisor and MES stakeholders to review the V&V plan. During this session, we will gather feedback on:
  - Adequacy of planned testing for mission-critical functions (e.g., reimbursement submission, audit logging).
  - Planned usability testing to ensure user satisfaction among student groups and administrators.
  - Risk mitigation strategies for potential project challenges, such as data security concerns.
  - Feasibility and scope of the planned testing effort.

  The feedback gathered will be used to refine the V&V plan further, ensuring it meets the expectations of all stakeholders.

By implementing the verified V&V plan, we aim to build confidence in the MES Custom Financial Expense Reporting Platform's functionality, usability, and security, ensuring it meets the expectations of the MES and its users.

3.5 Implementation Verification Plan

The implementation will be verified through a multifaceted approach:

Code Reviews

- All code changes will undergo peer review through GitHub Pull Requests before being merged into the main branch.
- At least one team member has to approve the pull request before it can be merged.
- Code reviews will use a standardized checklist (based on `docs/Checklists/Code-Checklist.`
  focusing on:
  - Implementation correctness against requirements and MIS.
  - Security considerations for financial data handling (e.g., input sanitization, permission checks).

  - Test coverage verification (ensuring new code has corresponding unit tests).
  - Error handling completeness.
  - Adherence to coding standards (Section 3.6).

Test Execution

- System tests as detailed in Section 4, with particular focus on:
  - Functional tests for reimbursement submission (Section 4.1).
  - Security and data integrity tests (Section 4.2).
  - Performance and scalability tests (Section 4.2).
  - Usability test (Section 4.2).
- Unit tests as outlined in Section 6, covering:
  - Individual module testing based on MIS specifications.
  - Integration points between components.
  - Edge cases and error conditions.

Static Analysis

- ESLint will be used for static code analysis to enforce coding style and detect potential issues.
- Type checking through TypeScript's compiler will be strictly enforced.
- ~~SonarQube will be used for code quality and security analysis.~~ Automated security vulnerability scanning (e.g., `npm audit`) will be integrated into the CI pipeline.

Review Sessions

- Weekly review sessions will be held with the team to discuss progress, issues, and verification results. Monthly meetings will include the project supervisor.

- Review sessions will include a demonstration of new features and verification results.
- Key milestones (e.g., POC, Rev 0) will involve demonstrations to stakeholders (MES VP Finance, select club leaders).
- The checkpoint meetings with the TA will also serve as informal review sessions for documentation and progress.
- The final presentation will serve as a comprehensive code walkthrough and system demonstration.

3.6 Automated Testing and Verification Tools

The following tools will be utilized for automated testing and verification, aligned with our development technology stack:

Testing Frameworks

- Jest: Primary testing framework for unit and integration testing of backend logic and utility functions.
- React Testing Library: Component testing focusing on user interactions and UI behavior for frontend components.
- ~~Cypress: End-to-end testing for user flows and system integration.~~ End-to-end (E2E) testing using Cypress will be explored if time permits, focusing on critical user flows like request submission and approval.

Code Quality Tools

- ESLint: Configured with specific rules for TypeScript and Next.js development.
- Prettier: Automated code formatting on commit to maintain consistent style.
- TypeScript: Static type checking with strict mode enabled.
- Git pre-commit hooks: Will be configured to run formatting (Prettier) and linting (ESLint) checks before commits are allowed.

Continuous Integration

- GitHub Actions for:
  - Automated unit test execution on each commit and pull request.
  - Code quality checks through ESLint.
  - Type checking through TypeScript compiler.
  - Automated deployment to staging environment for pull requests (using Vercel or similar).
  - Automated PDF generation for documentation (LaTeX build).
  - Automated dependency vulnerability scanning (`npm audit`).
- Test coverage reporting through Jest with:
  - Minimum 90% coverage requirement as specified in the Development Plan.
  - Coverage reports generated for each pull request.
  - ~~Blocking of merges if coverage thresholds aren't met.~~ Pull requests will display coverage metrics; merging below the threshold will require justification.

Performance Monitoring

- Lighthouse: Will be used manually during development and testing phases for performance, accessibility, and best practices checks.
- Chrome DevTools: Used during development for profiling memory usage and rendering performance.
- Next.js Analytics: Will be enabled in production deployments (if applicable) for real-user monitoring.

Security Testing

- npm audit: Regular dependency vulnerability scanning integrated into CI.

- ~~Automated vulnerability scanning integrated into CI pipeline.~~ Manual security code reviews focusing on potential issues like injection, access control flaws, and insecure data handling.

All tools have been selected to integrate seamlessly with our Next.js and TypeScript stack while supporting our coding standards and quality requirements. The automation of these tools through our CI/CD pipeline ensures consistent verification across all stages of development.

3.7 Software Validation Plan

The validation plan ensures the software meets the needs of the McMaster Engineering Society and its student groups through a comprehensive approach:

External Data Validation

- Representative sample data (anonymized if necessary) from MES's previous Excel-based system will be used to validate:
  - Accuracy of financial calculations (e.g., budget totals, reimbursement amounts).
  - Correct budget categorization logic.
  - ~~Processing time improvements.~~ Consistency with historical reporting formats where applicable.
- Sample receipts (various formats, qualities, provided by MES and team) will be used to validate the receipt processing system (including OCR accuracy and manual override).

Stakeholder Requirements Review

- Task-based inspection sessions with MES VP Finance and selected club leaders to verify SRS completeness:
  - Walkthrough of common reimbursement scenarios (e.g., simple expense, multi-item receipt, request requiring clarification).
  - Review of financial approval workflows (including multi-level approvals if implemented).

  - Verification of audit requirements and log content.

- Requirements validation through prototype demonstrations:
  - Interactive sessions with student group leaders representing different club types (e.g., large team, small club).
  - Feedback collection on user interface design and workflow intuitiveness.
  - Verification of notification preferences and content.

Rev 0 Demo Validation

- Scheduled demonstration with external supervisor (MES VP Finance).
- Focus areas for supervisor feedback:
  - Validation of implemented requirements from the SRS (specifically those targeted for Rev 0).
  - Verification of financial workflow accuracy.
  - Assessment of security measures (role separation, access control).
  - User interface evaluation (clarity, ease of use).
- This feedback will be documented as GitHub issues and incorporated into subsequent development iterations.

User Acceptance Testing

- Beta testing program with selected student groups (minimum 3-5 diverse groups):
  - Testing with real reimbursement scenarios over a defined period (e.g., 2 weeks).
  - Validation of notification systems (email delivery, content accuracy).
  - Performance under typical usage patterns (concurrent submissions, report generation).
- Specific validation metrics:

  - Reimbursement request completion time (target: <5 minutes average).
  - Budget categorization accuracy (target: >95% accuracy for rule-based categorization, user satisfaction with suggestions if ML is used).
  - System response times under load (target: key actions <3 seconds).
  - User error rates during submission process (target: <10% requiring correction).
  - Subjective feedback via survey (Appendix A).

Requirements Validation

- Regular requirement review sessions with the team and supervisor to ensure:
  - Alignment with MES financial policies (as documented or clarified).
  - Compliance with security requirements (RBAC, data handling).
  - Meeting of performance targets.
- Validation of specific SRS requirements:
  - Custom budget creation functionality.
  - Real-time ledger tracking.
  - Multi-level approval workflows (if implemented).
  - Automated notifications.
  - Audit log completeness and accuracy.

All validation activities will be documented and tracked through GitHub issues and project boards. Feedback will be incorporated into the development cycle. There will be particular attention to the core requirements of reducing wait times and preventing loss of reimbursement requests. Any changes to requirements identified through validation will be reviewed and updated in the SRS document accordingly.

4 System Tests

4.1 Tests for Functional Requirements

Test 1: Submission Confirmation

- **Test ID**: `test-id1`
- **Control**: Manual
- **Initial State**: User (`Role: 'Club Member', UserID: 'user123'`) is logged into the system. User is on the `/forms` page. No pending requests exist for `'user123'`.
- **Input**: Completed reimbursement form with fields:
  - *Group*: 'Engineering Society'
  - *Who Are You?*: 'Club Member'
  - *Email*: `'user123@mcmaster.ca'`
  - *Full Name*: 'Test User'
  - *Phone*: '905-555-1234'
  - *Role*: 'Treasurer'
  - *Budget Line*: 'Office Supplies'
  - *Approved Individual*: 'Project Lead'
  - *Reimbursement/Payment*: 'Reimbursement'
  - *Currency*: 'CAD'
  - *Amount*: '120.50'
  - *Payment Timeframe*: '2025-02-15'
  - *Payment Method*: 'E-Transfer'
  - *Payable To*: 'Test User'
  - *Interac Email*: `'user123@mcmaster.ca'`

  Attached digital receipt:

  - *File Name*: `receipt_office_supplies.pdf`

  - *File Format*: PDF
  - *File Size*: 150 KB

- **Output**: UI displays message: "Request submitted successfully!". Database table `payment_requests` contains a new row with `request_id` (e.g., UUID), `user_id='user123'`, `group_id` matching 'Engineering Society', `status`='Pending', `amount_requested_cad`=120.50, and other matching input fields. Email notification triggered (verify separately in Test 5).

- **Test Case Derivation**: Ensures the form submission logic updates the `payment_requests` table correctly, as described in the SRS (Requirement FR1).

- **How Test Will Be Performed**: 1. Log in as `'user123'`. 2. Navigate to `/forms`. 3. Fill form fields exactly as specified in Input. 4. Upload the specified PDF file. 5. Click 'Submit'. 6. Verify UI confirmation message. 7. Query database `payment_requests` table for the new record, verifying all fields match input and defaults.

Test 2: Approval Workflow

- **Test ID**: `test-id2`

- **Control**: Manual

- **Initial State**: Admin user (`Role: 'MES Admin', UserID: 'admin01'`) is logged in. A request (`request_id: 'req123'`) exists in `payment_requests` with `status`='Pending', submitted by `'user123'`.

- **Input**: Admin user navigates to `/dashboard/requests`, finds request `'req123'`, selects 'Approved' from the status dropdown.

- **Output**: UI updates status for `'req123'` to 'Approved'. Database `payment_requests` table shows `status`='Approved' for `request_id='req123'`. Email notification sent to `'user123@mcmaster.ca'` (verified in Test 5).

- **Test Case Derivation**: Verifies administrators can change request status as per SRS Requirement FR2.

- **How Test Will Be Performed**: 1. Log in as `'admin01'`. 2. Navigate to `/dashboard/requests`. 3. Locate request `'req123'`. 4. Change status dropdown to 'Approved'. 5. Verify UI update. 6. Query database `payment_requests` to confirm status change for `'req123'`.

Test 3: Expense Addition

- **Test ID**: `test-id3`
- **Control**: Manual
- **Initial State**: User `'user123'` logged in. Operating budget line for 'Office Supplies' under group 'Engineering Society' exists with amount 500.00 and type 'expense'. Request `'req123'` (Test 2) has status 'Approved' with amount 120.50.
- **Input**: System automatically (or admin manually triggers process for status change to 'Reimbursed' for request `'req123'`).
- **Output**: A new line is added to `operating_budget_lines` table with `request_id='req123'`, `line_type`='expense', `amount`=120.50, linked to 'Engineering Society' group. The `total_budget` for 'Engineering Society' in the `groups` table is reduced by 120.50. UI on `/dashboard/operating_budget` reflects the new line and updated group total.
- **Test Case Derivation**: Verifies that approved and reimbursed requests correctly impact the operating budget totals (Implicit requirement related to budget tracking).
- **How Test Will Be Performed**: 1. Log in as admin. 2. Approve request `'req123'` (if not already done). 3. Change status of `'req123'` to 'Reimbursed'. 4. Query `operating_budget_lines` table for the new expense line linked to `'req123'`. 5. Query `groups` table to verify `total_budget` for 'Engineering Society' has decreased by 120.50. 6. Navigate to `/dashboard/operating_budget` and verify UI update.

Test 4: Budget Creation and Categorization

- **Test ID**: `test-id4`
- **Control**: Manual

- **Initial State**: Admin user `'admin01'` logged in. No group named 'Test Club Budget' exists.
- **Input**: Admin navigates to `/dashboard/operating_budget`. Clicks 'Add Group'. Enters 'Test Club Budget' as name. Clicks '+ Add Line'. Enters Label='Membership Dues', Amount='500', Type='income'. Clicks '+ Add Line'. Enters Label='Event Catering', Amount='200', Type='expense'. Clicks 'Save All'.
- **Output**: UI shows new group 'Test Club Budget' with two lines. Database table `groups` has new entry for 'Test Club Budget' with `total_budget`=300 (500 income - 200 expense). Database table `operating_budget_lines` has two new entries linked to the new group ID with correct labels, amounts, and types.
- **Test Case Derivation**: Verifies admins can create new groups and budget lines via the operating budget interface (Requirement FR6).
- **How Test Will Be Performed**: 1. Log in as `'admin01'`. 2. Follow steps in Input. 3. Verify UI updates. 4. Query `groups` table for 'Test Club Budget' and check `total_budget`. 5. Query `operating_budget_lines` for the two new lines, verifying details.

Test 5: Notification on Reimbursement Status Change

- **Test ID**: `test-id5`
- **Control**: Manual (trigger) / Automated (check)
- **Initial State**: Request `'req123'` exists with status 'Pending', submitted by `'user123@mcmaster.ca'`. Notification system (SendGrid) is configured.
- **Input**: Admin user `'admin01'` changes status of request `'req123'` to 'Approved'.
- **Output**: Email is sent via SendGrid to `'user123@mcmaster.ca'` with Subject containing "Status Update - Approved" and Body containing request ID `'req123'`.
- **Test Case Derivation**: Verifies email notifications are triggered on status change (Requirement FR3).

- **How Test Will Be Performed**: 1. Log in as 'admin01'. 2. Change status of 'req123' to 'Approved'. 3. Check SendGrid logs or recipient inbox ('user123@mcmaster.ca') for the notification email matching the expected content.

Test 6: Audit Trail Logging

- **Test ID**: test-id6
- **Control**: Manual (trigger) / Automated (check)
- **Initial State**: System is operational. Audit logging is enabled.
- **Input**: User 'user123' submits a new request (req456). Admin 'admin01' approves request 'req123'.
- **Output**: Database table (e.g., audit_log) contains entries for: a) 'req456' creation event, linked to user_id='user123', with correct timestamp. b) 'req123' approval event, linked to user_id='admin01', with correct timestamp and status change details.
- **Test Case Derivation**: Verifies critical actions are logged for auditing (Requirement FR5).
- **How Test Will Be Performed**: 1. Perform the two actions specified in Input. 2. Query the audit_log table for entries matching the performed actions, verifying user_id, timestamp, action type, and relevant entity ID (request_id).

4.2 Tests for Nonfunctional Requirements

Test 7: Response Time Under Load

- **Test ID**: test-id7
- **Type**: Performance, Dynamic, Automatic (Load Test Tool)
- **Initial State**: System running with baseline load (< 5 active users). Database contains approx 1000 requests.

- **Input/Condition**: Simulate 50 concurrent users submitting the reimbursement form (`/forms`) over a 60-second period using a tool like k6 or JMeter. Each submission uses unique valid data and a small receipt file (<500KB).
- **Output/Result**: Average response time for form submission API endpoint is <2 seconds. 95th percentile response time is <3 seconds. Error rate is <1%.
- **Test Case Derivation**: Verifies system's performance under moderate load, per nonfunctional requirements for speed and latency (NFR1).
- **How Test Will Be Performed**: 1. Configure load testing script (k6/JMeter) to simulate 50 virtual users. 2. Run the script targeting the reimbursement submission API endpoint for 60 seconds. 3. Collect and analyze response time (average, p95) and error rate metrics from the tool's output.

Test 8: Security and Access Control

- **Test ID**: `test-id8`
- **Control**: Manual
- **Initial State**: User `'user123'` (Role: 'Club Member') is logged in. Admin-only page `/dashboard/users` exists.
- **Input**: User `'user123'` attempts to navigate directly to `/dashboard/users` URL.
- **Output**: User is redirected to `/dashboard/home`. No access granted to the `/dashboard/users` page content. Check server logs for potential 403 status or redirection log.
- **Test Case Derivation**: Confirms that unauthorized access is prevented as per security requirements in SRS (NFR2).
- **How Test Will Be Performed**: 1. Log in as `'user123'`. 2. Manually enter `/dashboard/users` in the browser address bar. 3. Verify redirection to `/dashboard/home`. 4. Inspect browser network tab (or server logs) to confirm no sensitive data from the target page was loaded.

Test 9: Load Handling

- **Test ID**: `test-id9`
- **Type**: Performance, Dynamic, Manual
- **Initial State**: User dashboard loaded with at least 10 active, distinct user sessions.
- **Input/Condition**: 10 team members simultaneously perform common actions within a 1-minute window: 5 submit reimbursement requests, 3 update request statuses, 2 navigate the operating budget page.
- **Output/Result**: All actions complete successfully. Subjective assessment: UI remains responsive (no noticeable lag >4 seconds). Server CPU/Memory usage stays below 80%. No crashes or data inconsistencies observed.
- **Test Case Derivation**: Verifies system's load handling capacity under realistic concurrent usage (NFR5).
- **How Test Will Be Performed**: 1. Coordinate 10 testers. 2. At a designated time, each tester performs their assigned action simultaneously. 3. Testers report subjective responsiveness. 4. Monitor server resource usage during the test window. 5. Verify database consistency for actions performed.

Test 10: Data Integrity and Security

- **Test ID**: `test-id10`
- **Control**: Manual (Security Test)
- **Initial State**: Data integrity checks enabled. A request `'req789'` exists with amount 50.00.
- **Input**: Attempt to update the amount of request `'req789'` via a direct API call or manipulated form submission (e.g., intercepting request) to an invalid value (e.g., -100.00 or `'abc'`). Attempt basic SQL injection pattern in a searchable field (e.g., `' OR '1'='1'`).

- **Output**: API request to update amount fails with a validation error (e.g., 400 Bad Request). SQL injection attempt yields no results or an error, and does not expose unintended data. Original amount (50.00) for `'req789'` remains unchanged in the database.
- **Test Case Derivation**: Ensures data integrity and protection against unauthorized modification attempts as per security and data integrity requirements (NFR3).
- **How Test Will Be Performed**: 1. Use a tool like Postman or browser developer tools to attempt the invalid API call to update the request amount. 2. Enter SQL injection pattern into a search field in the UI. 3. Verify API responses indicate failure/rejection. 4. Query the database to confirm the amount for `'req789'` is still 50.00. 5. Check application/database logs for errors related to the attempts.

Test 11: Usability Evaluation

- **Test ID**: `test-id11`
- **Control**: Manual (User Testing with Survey/Checklist)
- **Initial State**: System deployed to a test environment accessible by participants. 5-10 representative end-users (e.g., mix of club treasurers, regular members) recruited. Usability checklist prepared (see Appendix A).
- **Input**: Participants are given a script with 3 typical tasks:
  1. Log in and submit a reimbursement request for $75 for 'Team Jerseys' with a provided sample receipt image (`jersey_receipt.png`).
  2. Find the status of the request submitted in task 1 on the `/dashboard/requests` page.
  3. Navigate to their account information page (`/dashboard/accountInfo`) and attempt to change their phone number.

  Participants complete tasks and then fill out the Usability Survey Checklist (Appendix A). Observer notes task completion times and any points of confusion verbally expressed or observed (e.g., hesitation, wrong clicks).

- **Output**: Quantitative data: Average task completion times (target: Task 1 <3 mins, Task 2 <1 min, Task 3 <1 min). Average satisfaction ratings from checklist (target: >4.0/5.0 overall). Qualitative data: Observer notes on difficulties (e.g., "User struggled to find the submit button", "Confused about which group to select"), user comments from survey ("The OCR was helpful", "Took a while to find my old requests").

- **Test Case Derivation**: Addresses the Usability extra requirement (detailed in Section 2), ensuring the system meets user experience standards defined implicitly in the SRS (NFR4).

- **How Test Will Be Performed**: 1. Conduct individual usability testing sessions (in-person or remote screen share). 2. Provide users with the task script and sample receipt file. 3. Observe task completion using think-aloud protocol, noting time and issues. 4. Administer the Usability Survey Checklist post-session. 5. Aggregate quantitative results and synthesize qualitative feedback into actionable UI improvements.

4.3 Traceability Between Test Cases and Requirements

Table 1: Traceability Between Test Cases and Requirements

Requirement (Assumed SRS Ref.)	**Test Case(s)**
FR1: Reimbursement Submission	test-id1
FR2: Approval Workflow	test-id2
FR3: Notification System	test-id5
FR5: Audit Logging	test-id6
FR6: Budget Management	test-id3, test-id4
NFR1: Performance Under Load	test-id7, test-id9
NFR2: Security and Access Control	test-id8
NFR3: Data Integrity and Security	test-id10
NFR4: Usability (Extra)	test-id11

5 Nondynamic Testing Plan

In addition to the dynamic tests described in Section 4, the following nondynamic testing techniques will be used:

- **Static Code Analysis**: We will run linters (ESLint) and static analyzers (TypeScript compiler with strict checks) to detect potential errors, stylistic issues, and security vulnerabilities before runtime. This is integrated into our CI pipeline.
- **Code Reviews and Inspections**: Each major feature or significant code change will undergo a formal peer review via GitHub Pull Requests, with at least one other team member examining the code against project coding standards (Section 3.6), the design described in the MIS, and the checklist (`docs/Checklists/Code-Checklist.tex`).
- **Document Reviews**: Key project artifacts (SRS, MG, MIS, user guide) will be reviewed for consistency. This includes verifying that naming conventions, references, and requirement statements align with each other and the code base.
- **Walkthroughs and Technical Demos**: At milestone meetings (POC, Rev 0, Final), we will conduct walkthroughs of new or complex features to validate design decisions and ensure maintainability.

These nondynamic activities supplement the dynamic tests by identifying defects early, ensuring code quality, and maintaining documentation accuracy.

6 Unit Test Description

6.1 Reference to MIS and Unit Testing Philosophy

After the MIS (detailed design document) is finalized, we will reference each software module (and its Access Programs) in the MIS. Our philosophy is to:

- Write unit tests (using Jest and React Testing Library) for each major function, utility, API endpoint handler, and critical React component,

covering both normal operation and edge cases (boundary values, invalid inputs, etc.).

- Focus on the most critical or complex modules first (e.g., financial calculations, authorization checks in hooks/middleware, database interactions).
- Maintain a `__tests__` directory alongside source files or a central `src/tests` directory, with test files named using conventions like `moduleName.test.ts` or `componentName.test.tsx`, and meaningful test method names (e.g., `it('should correctly calculate budget total')`).
- Automatically run these tests in our GitHub Actions CI pipeline on every push and pull request.

6.2 Unit Testing Scope

- **In Scope**: Core modules owned by the project team, including reimbursement processing logic, budget management calculations, user authentication helpers, notification triggers, permission hooks (`usePermissions`), API route handlers, utility functions (`chartUtils`, `utils.ts`), and key UI components (e.g., `EditableStatusRow`, form sections).
- **Out of Scope**: Direct testing of third-party libraries (Supabase client library, Recharts, SendGrid/Twilio SDKs, Tesseract.js). These will be mocked during unit tests. Integration testing (manual or potentially E2E) will verify interactions with these external services.

6.3 Tests for Functional Requirements at the Unit Level

Detailed unit test cases will be implemented in the codebase following the philosophy above. Test files will be linked to the corresponding source files. Instead of listing exhaustive inputs/outputs here, we will ensure test code is self-documenting with clear `describe` and `it` blocks. Example test structures:

6.3.1 Example: API Route Handler (`update-status`)

- **Test Suite:** `update-status.test.ts`
- **Test Case 1:** `it('should update status to Approved for valid admin request')`
  - **Setup:** Mock Supabase client, mock authenticated admin user, existing request with 'Pending' status.
  - **Input:** API request body { `requestId: 'req123', newStatus: 'Approved'` }.
  - **Assertions:** Verify Supabase `update` called with correct parameters. Verify response is success (e.g., 200 OK). Verify notification function called.
- **Test Case 2:** `it('should return 403 Forbidden if user lacks permission')`
  - **Setup:** Mock Supabase client, mock authenticated user with insufficient permissions.
  - **Input:** API request body { `requestId: 'req123', newStatus: 'Approved'` }.
  - **Assertions:** Verify response status is 403. Verify Supabase `update` was *not* called.
- **Test Case 3:** `it('should return 400 Bad Request for invalid status value')` ...etc ...

6.3.2 Example: Permission Hook (`usePermissions`)

- **Test Suite:** `usePermissions.test.ts`
- **Test Case 1:** `it('should return correct permissions for a user with multiple roles')`
  - **Setup:** Mock Supabase client `from('user_roles').select()` to return roles data including 'Admin' (global) and 'Club Treasurer' (group-specific).
  - **Action:** Render a test component using the hook.

- **Assertions:** Verify the hook returns the combined, unique set of permissions from both roles. Verify `loading` becomes false.

- **Test Case 2:** `it('should return empty permissions for unauthenticated user')` ...etc ...

6.4 Tests for Nonfunctional Requirements at the Unit Level

If a particular module has performance-critical functions (e.g., complex financial calculations), we will write micro-benchmark tests to measure execution time or memory usage. For example:

6.4.1 Example: Budget Calculation Utility

1. `test-budgetUtil-perf1`
   Type: Performance, Dynamic, Automatic
   Initial State: Budget calculation utility function imported. **Input/Condition**: Call the `calculateGroupTotal` function 10,000 times with a sample budget lines array (e.g., 50 lines). **Output/Result**: The average execution time should not exceed 1ms per call on standard CI runner hardware.
   How Test Will Be Performed:

   - Run a loop calling the function 10,000 times.
   - Log the total execution time using `performance.now()` or Jest's timer mocks.
   - Calculate the average time and assert it is under 1ms.

A Appendix A: Usability Survey Checklist (Sample)

Participant ID: ____ **Date:** _____ **Task Tested:** _____

Instructions: Please rate the following aspects on a scale of 1 (Strongly Disagree) to 5 (Strongly Agree).

1. The task was easy to complete. (1) (2) (3) (4) (5)
2. I understood what I needed to do at each step. (1) (2) (3) (4) (5)
3. The buttons and labels on the screen were clear. (1) (2) (3) (4) (5)
4. It was easy to find the information I needed. (1) (2) (3) (4) (5)
5. The system responded quickly to my actions. (1) (2) (3) (4) (5)
6. I felt confident using this part of the system. (1) (2) (3) (4) (5)
7. The error messages (if any) were helpful. (1) (2) (3) (4) (5) [N/A]
8. Overall, I am satisfied with the usability of this feature. (1) (2) (3) (4) (5)

Open Feedback:

1. What did you like most about completing this task?

   _____

2. What did you find most confusing or difficult?

   _____

3. Do you have any suggestions for improvement?

   _____

Appendix

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?
   Sufyan: What went well for me this assignment was getting github actions to build the pdf on commit. I also enjoyed learning about all the tooling we could use for the project to ensure code quality, test coverage and security.
   Taaha What went well while writing this deliverable was the amount of support I had when asking teammates questions. Was extremely helpful whenever I came across an issue I was unsure of.
   Omar: What went well was the clarity of this document, we understand our project so we were able to devise this document with little to no questions asked to the TA.
   Housam: One aspect that went well for me was refining the plan structure to align with the project's goals and requirements. Working on the document also helped solidify my understanding of the V&V process and how it integrates into the larger project goals.
   Rachid: What went well for me this deliverable was that we discussed how we were going to work on this very early and this allowed me to spend more time thinking and working on this deliverable

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Sufyan: The pain points I experienced were getting the github actions

to work properly. I had to do a lot of research and trial and error to get it to work. I also had some issues with the formatting of the document. I resolved these issues by asking for help from my team members.
Taaha The pain points I experienced were mostly related to formatting the document correctly. In order to resolve this, I got help from teammates and looked online.
Omar: I got really sick during this deliverable which slowed down my progress a lot, in addition, it was harder to manage this deliverable since I had a bunch of other deliverables due near the same time.
Housam: The main challenge I faced was balancing detail with conciseness, as I wanted to cover each point in the plan without making it overly lengthy. At times, I felt I was going into too much detail, so I reviewed my work multiple times to find a clear and concise way to present information. Discussing with teammates helped refine certain sections and focus on essential content.
Rachid: This deliverable was extremely clear and as a result, I experienced no pain points.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project?

   Key skills needed for V&V include:

   - **Dynamic Testing:** Using Jest and React Testing Library for unit and integration testing. Potentially Cypress for E2E.
   - **Static Analysis:** Using ESLint to identify code issues. Leveraging TypeScript's static typing.
   - **CI/CD Setup:** Configuring GitHub Actions to automate testing and deployment.
   - **Security Testing:** Implementing npm audit for vulnerability scanning and manual security reviews.

- **Usability Testing:** Conducting accessibility and usability tests using Lighthouse and structured user feedback sessions.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   - **Dynamic Testing:** Team tutorials on Jest/React Testing Library and applying tests in the project, led by Housam. Approach 2: Pair programming on test creation. Chosen: Team tutorials initially for broad understanding, followed by individual application.
   - **Static Analysis:** Reviewing ESLint/TypeScript documentation and integrating tools during development, led by Omar. Approach 2: Analyzing linting results from CI runs. Chosen: Integrating tools early and learning from CI feedback.
   - **CI/CD Setup:** Using GitHub documentation and incremental setup of GitHub Actions, led by Sufyan. Approach 2: Examining example workflows from other projects. Chosen: Incremental setup based on official docs for reliability.
   - **Security Testing:** Researching secure coding practices for Next.js/Supabase and using npm audit, led by Taaha. Approach 2: Performing manual code reviews focused on security aspects (e.g., permission checks). Chosen: Combine automated scanning (npm audit) with manual reviews.
   - **Usability Testing:** Exploring Lighthouse reports and gathering user feedback via planned sessions, coordinated by Rachid. Approach 2: Applying usability heuristics (e.g., Nielsen's heuristics) during design reviews. Chosen: Gathering direct user feedback for real-world validation.