System Verification and Validation Plan for Software Engineering

Team #12, Streamliners
Mahad Ahmed
Abyan Jaigirdar
Prerna Prabhu
Farhan Rahman
Ali Zia

October 28, 2025

Revision History

Date	Version	Notes
2025-10-27	1.0	Intial draft of VnV Plan

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	**description**
T	Test

2 General Information

2.1 Summary

The software being tested is the **Large Event Management Platform**, a comprehensive web and mobile-based application developed for the **McMaster Engineering Society (MES)** to centralize and automate large-scale event management processes. The system is designed to replace fragmented tools such as Google Forms and spreadsheets with a unified, secure, and scalable solution that simplifies event organization for both administrators and attendees.

The platform enables event organizers to create and manage events, process ticket payments, collect waivers, manage RSVP and table sign-ups, and monitor live event analytics through an administrative dashboard. Attendees can register, make payments, sign digital waivers, and check in at events using QR codes. The system enforces role-based and feature-based access control to ensure that different user types (e.g., admins, volunteers, attendees) have appropriate permissions.

The platform is being developed using a **Vite + TanStack Router + TanStack Create** frontend stack paired with a **NestJS** backend and a **PostgreSQL** database. Verification and validation will ensure that all components function correctly, maintain data integrity, and deliver a consistent, user-friendly experience for MES event operations.

2.2 Objectives

This Verification and validation plan intends to accomplish the objective of ensuring the Large Event Management Platform for the McMaster Engineering Society meets its mission critical qualities that are most important as defined by the Software Requirements Specification (SRS) and Hazard Analysis (HA).

The mission critical qualities intended to be accomplished as part of the objective are as follows:

- **Functional Correctness:** The core system components and functionalities outlined in the SRS such as the payment system, role-based and feature-based access control and RSVP/Bus/Table signups should operate correctly and reliably to support the event operation of the platform for both admins and users.

- **System Reliability:** The platform remains stable and responsive under expected and unexpected load conditions and supports users and admins to accomplish their tasks without hindrance.
- **Safety and Security:** The platform must comply with security requirements documented in the HA relating to protection of payment information and personal user data as well as proper enforcement of role and feature based access controls.

Out of scope objectives due to project scope and resource limitations include:

- **Third party service verification:** External services provided by payment providers will not be verified. This objective is left out as these services are assumed to be reliable and safe for integration within the platform due to their existing brand recognition and usage in industry making this exclusion justified.
- **Accessibility features:** Accessibility features are beyond current needs and will not be verified for quality of usage. This objective is left out because a general user-friendly design is assumed to encompass most quality needs. This exclusion is justified as the required resources needed are disproportionate to its impact.

2.3 Challenge Level and Extras

This project includes several approved extras designed to enhance the quality, usability, and long-term sustainability of the system beyond its core functional requirements. These activities aim to ensure that the final product is user-friendly, maintainable, and well-documented for future MES teams.

- **Usability Testing:** The team will conduct structured usability sessions with MES organizers and student participants to evaluate workflow clarity and overall user experience. Feedback from these sessions will guide interface refinements and ensure the system remains intuitive for both attendees and administrators.
- **User Documentation:** Comprehensive user and administrator documentation will be prepared to support onboarding, maintenance, and

future development. This documentation will include setup instructions, feature overviews, and troubleshooting guidelines to help future MES teams effectively operate and extend the platform.

By including these extras, the project emphasizes usability, maintainability, and user-centered design. The planned activities will ensure that the system not only meets its core functional goals but also provides a reliable, accessible, and sustainable solution for event management within the MES environment.

2.4 Relevant Documentation

- **Hazard Analysis (HA)** Streamliners (2025b):
  The Hazard Analysis document identifies potential hazards and system failures. The VnV includes test cases to test for these possible hazards and informs the VnV testing plan of what a failed test case would look like.

- **Software Requirements Specification (SRS)** Streamliners (2025d):
  The SRS document describes in detail the functional and non-functional requirements of the system. The VnV plan reference this document as a baseline to ensure all requirements are being met.

- **Development Plan** Streamliners (2025a):
  The development plan includes the process of developing the MacSync platform. It supports the VnV plan by including how automated test cases and CI/CD pipelines will be used for verification and validation.

- **Module Interface Specification (MIS)** Streamliners (2025c):
  The MIS provides details of the expected inputs and outputs of each software module for the MacSync application. The MIS supports the VnV by providing concrete tests at the unit level for inputs and outputs of each isolated module.

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

The Verification and Validation (V&V) process for the **Large Event Management Platform** will be jointly executed by all members of the **Large Event 3 (Streamliners)** development team. Team members are paired across key verification areas to ensure overlap between functional, integration, and validation responsibilities. This approach promotes redundancy in testing coverage, continuous feedback across modules, and consistent quality assurance throughout development.

Members	V&V Focus Area	Responsibilities
Mahad Ahmed, Farhan Rahman	Backend and Integration Verification	Collaboratively design and maintain automated unit and integration tests using Jest and Supertest. Verify database operations and API endpoints for correctness, reliability, and adherence to data constraints. Perform end-to-end verification to confirm consistent data flow between backend, database, and frontend layers, ensuring transactional integrity and performance under load.
Abyan Jaigirdar, Ali Zia	Frontend and Usability Verification	Conduct verification of user interfaces and workflows using Vitest and React Testing Library. Evaluate user experience, input validation, and navigation logic for consistency with SRS-defined behaviours. Lead task-based usability walkthroughs with MES stakeholders to validate functional clarity and intuitive interaction across web and mobile views.
Prerna Prabhu	System Validation and Documentation Verification	Manage structured reviews of the SRS, V&V Plan, and associated documentation through checklist-based inspections. Maintain the requirements traceability matrix and coordinate stakeholder validation meetings. Consolidate feedback from both functional and usability testing to ensure all requirements are met and fully validated against MES operational objectives.
Luke	External Supervisor / Reviewer	Provides independent oversight of verification completeness and validates that all implemented features align with MES event-management goals. Participates in post-demo validation reviews and confirms system readiness for production testing.

All verification and validation activities, test results, and stakeholder feedback will be tracked through GitHub Issues and linked to corresponding commits to ensure full traceability between requirements, verification outcomes, and corrective actions.

3.2 SRS Verification

The SRS verification will be done through a multi-stage review process involving peer reviews, team reviews, supervisor feedback and checklist based inspection to confirm alignment with project objectives.

The verification approach will involve the following steps:

- **Peer Review (Primary reviewers):** Members of the primary reviewing team will take part in a structured review of the SRS where issues they find will be logged as GitHub Issues and resolved through tracked commits. The deliverable would be a consolidated review log documenting each issue identified and the resolution implemented by the member whose part is associated with the issue.

- **Team Review:** Based on the review provided by the primary reviewers and the resolution implemented by the team member who was responsible for the associated issue, a different team member will work on identifying if the resolution sufficiently addresses the feedback and implements a corrective solution. The solutions that address the feedback will then be pushed to create the finalized document.

- **Supervisor Review:** In this step, a meeting will be scheduled with the project supervisor in which a structured review walkthrough will take place to validate the SRS. The structured walkthrough will be a meeting in which the SRS will be presented. The agenda will cover the following: High level overviews of each aspect of the SRS to validate main functionalities, system processes and scope boundaries, critical requirements derived from problem statements and discussing the validity of assumptions made. The validation questions that will be asked will relate to understanding if all functional requirements are measurable and verifiable and if there are any gaps or ambiguities in assumptions or scope definitions. The supervisor feedback will be documented, addressed and resolved with traceability to the specific SRS sections.

- **SRS Checklist:** In this final step the team will gather and use the below verification checklist to ensure comprehensive evaluation of the SRS using a Pass/Needs Revision criterion

Category	**Verification Criteria**
Accuracy/Completeness	All requirements are included and clearly defined
Consistency	No conflicting/contradicting/overlapping requirements exist across sections
Clarity	Requirements are clear and leave no room for ambiguity
Feasibility	Requirements are achievable and realistically implementable

3.3 Design Verification

The design verification process will confirm that the system architecture, data flow, and component design correctly satisfy all functional and non-functional requirements defined in the SRS. Verification will occur before implementation begins to ensure that the design is both complete and technically feasible.

Verification Approach: The following activities will be used to verify the design:

- **Peer Review:** Each design document will be reviewed by an assigned peer team within the course. Their feedback will focus on correctness, consistency, and alignment with the SRS. All comments will be logged and tracked through GitHub Issues until resolved.
- **Team Review:** Internal walkthroughs will validate subsystem responsibilities, data flows, and interface consistency across the mobile app, admin dashboard, and backend service.
- **Supervisor Review:** The supervisor will review key design decisions and architectural diagrams to ensure the proposed solution is feasible and satisfies project constraints.

- **Traceability Check:** A mapping between SRS functional and non-functional requirements and corresponding design components will be created to confirm full coverage.

Design Verification Checklist

Category	**Verification Criteria**
Completeness	All SRS requirements are represented in the design model.
Consistency	No contradictory or missing data/control flows between modules.
Feasibility	Each design decision is realistic given project constraints.
Modularity	System is divided into clear, maintainable, and loosely coupled components.
Traceability	Every requirement in the SRS can be traced to one or more design elements.

Summary: Design verification will rely on peer reviews by classmates, team walkthroughs, and supervisor feedback to confirm that the proposed architecture is correct, consistent, and feasible before implementation begins. Review outcomes will be documented and used to refine the design prior to implementation.

3.4 Verification and Validation Plan Verification

The Verification and Validation Plan for MacSync will be carried out by multiple techniques such as peer reviews, evaluation checklists, and mutation testing. The goal of this verification is to ensure that the VnV Plan is thorough, complete, and consistent. Successful completion of this verification will allow the team to verify that all objectives and requirements are align with the scope of the project.

Peer reviews will be conducted by classmates from other capstone groups, and potentially teaching assistants and capstone supervisors. Using a formal checklist, this will ensure the VnV plan traces back to the SRS requirements and project objectives. All feedback through peer reviews will be recorded on github as issues so the team can then conduct meetings to assess feedback and implement them if deemed appropriate.

Mutation testing will also be done to quantify the coverage of the test plan. By introducing faults into the system we can ensure that our tests catch these faults and report them in the test report.

Table 1: VnV Plan Verification Checklist

Criteria	**Verification Tasks**
Coverage	☐ Traceability to all functional and non-functional requirements ☐ Each requirement mapped to one or more test cases
Methodology	☐ Verification techniques justified and feasible (e.g., peer review, mutation testing) ☐ Validation strategies defined for stakeholder review sessions
Process	☐ Review feedback mechanism established via GitHub Issues ☐ Scheduled plan refinement meetings

3.5 Implementation Verification

The implementation verification phase ensures that the developed system correctly satisfies both its functional and non-functional requirements as specified in the SRS document. Verification activities during implementation will be conducted through a combination of automated testing, static analysis, code reviews, and post-deployment validation.

This section provides an overview of how implementation verification will be achieved, how it connects to the defined system-level tests (Section 4.1 and 4.2), and how the forthcoming unit testing phase will integrate once the detailed design (MIS) document is finalized.

Scope and Approach
Implementation verification will primarily focus on confirming that the system behavior matches the requirements defined in the SRS, with emphasis on correctness, data integrity, and security. The verification plan follows a layered testing strategy:

- **Functional Verification:** Each feature will be tested using the system tests detailed in Section 4.1 (Tests for Functional Requirements), including ticket purchasing, bus/table registration, waiver submission, and event creation/management.
- **Non-Functional Verification:** Performance, usability, security, data integrity, and reliability will be validated through the tests outlined in Section 4.2 (Tests for Nonfunctional Requirements).
- **Static Verification:** Code walkthroughs and inspections will be conducted before integration to ensure compliance with internal coding standards, proper documentation, and modular design practices.

All implementation verification tests will be executed in the staging environment before release to production. Each test result will be logged and compared against the acceptance criteria specified in Section S.6 of the SRS.

Testing Artifacts and Traceability
Verification of the implementation will rely on the following artifacts:

- **Automated Unit and Integration Tests:** To verify module-level correctness and interface behavior (to be completed post-MIS design stage).
- **System Test Suite:** The detailed test cases documented in Sections 4.1 and 4.2 for both functional and non-functional requirements.
- **Static Code Reviews:** Manual and tool-assisted code reviews to identify defects early in the development process.
- **Traceability Matrix:** A mapping of all SRS requirements to corresponding test cases will ensure that all implementation aspects have been verified.

Automated Testing and Toolchain
Implementation verification will leverage automated testing frameworks to ensure repeatability and scalability:

- **Frontend/UI:** Cypress for end-to-end testing of user workflows (e.g., registration, login, payment).
- **Backend/API:** Jest and Postman for integration, concurrency, and REST endpoint validation.
- **Static Analysis and Style Checks:** ESLint and Prettier to enforce code quality and consistent formatting.
- **Continuous Integration:** GitHub Actions for automated builds, test execution, and code coverage reporting.

Unit Testing Placeholder and Future Work
Unit testing for individual modules (e.g., authentication, database operations, and payment processing) will be developed following the completion of the MIS (Module Interface Specification) document. At that stage, each module will have:

- Defined input/output specifications.
- A corresponding suite of automated unit tests.
- Documentation describing test coverage and expected behavior.

Since the MIS defines the internal structure of each module, the unit test plan will be appended to this document once design finalization occurs. The unit testing phase will ensure that each software component behaves correctly in isolation before being integrated into the full system.

Summary
In summary, implementation verification combines automated testing, static analysis, and structured validation against the SRS. The test cases defined in Section 4.1 and Section 4.2 will serve as the primary basis for verification. Future unit testing will build upon this foundation once the design phase (MIS) is complete, ensuring comprehensive validation of both high-level and component-level system behavior.

3.6 Automated Testing and Verification Tools

Automated testing will form the foundation of the verification process for the **Large Event Management Platform**, ensuring correctness, maintainability, and reliability throughout all development stages. The testing strategy integrates both frontend and backend verification frameworks with continuous integration pipelines to provide consistent and repeatable test execution.

For the **frontend**, the team will use **Vitest**, the native testing framework for Vite-based projects, alongside the **React Testing Library** to verify component behaviour, routing, and user interactions. These tools allow developers to simulate real user actions such as form submissions, navigation, and event sign-ups to confirm compliance with the Software Requirements Specification (SRS). The frontend test suite will focus on validating component state management, data rendering, and input handling.

For the **backend**, the team will use **Jest** in combination with **Supertest** to perform unit and integration testing of the NestJS API endpoints. These tests will validate service logic, controller routes, and database queries executed through Drizzle ORM against a local PostgreSQL instance. **Test containers** will be used to simulate isolated database environments for integration tests, ensuring consistent and reproducible results.

Static verification will be maintained through the use of **ESLint** and **Prettier**, enforcing the Airbnb JavaScript style guide and consistent formatting across the codebase. Type correctness and interface verification will be handled by **TypeScript**, ensuring compatibility between frontend and backend modules. These static tools collectively reduce logical errors and ensure the system adheres to coding standards and best practices.

Continuous Integration (CI) will be managed using **GitHub Actions**. Each pull request will trigger automated workflows that include linting, type checking, and full test execution. The CI pipeline will produce code coverage reports summarizing the percentage of tested lines and functions, which will be reviewed at the end of each sprint to assess verification completeness. A goal coverage target of 75% will try to be maintained across all core modules to ensure adequate testing depth.

These tools, configurations, and automation strategies are consistent with the team's **Development Plan** and will evolve alongside implementation progress to support scalable and maintainable verification.

3.7 Software Validation

The software validation process will mainly be focussed on confirming the system is correctly achieving its objectives and addresses stakeholder needs and aligns with functional requirements identified in the SRS. For instance, verifying features such as the payment system, role and feature based access control and RSVP/Bus/Table sign-ups are implemented, and perform correctly.

Since the Large Event Management Platform is a new undertaking of event organization, there are no external data sources that can be used for validating if results align with intended real-world use cases or behaviors. Therefore the need arises to engage stakeholders such as club executives and event organizers in a structured review session to check that the SRS captures the requirements needed for their needs and goals. The approach taken for this can be task based inspection in which the stakeholders are focused on verifying compliance of the requirements document to their needs and assessing risk by focusing on already defined functions. Additionally, following the Rev 0 demo, a dedicated validation meeting will be done with the external supervisor of the team in which the team will demonstrate each core feature through task based inspection. The aim of the inspection is to allow the supervisor to observe and provide critique on critical workflows and functions as defined in the SRS such as creating events, processing payments, verifying access permissions and managing sign-ups. Validating these workflows, will aim to confirm their operational and functional expectations align with the SRS defined requirements focused on delivering functional requirements to the stakeholders. The combination of both stakeholder and technical validation through MES executives and the external supervisor will ensure that both system alignment to needs and practical utility is achieved

4 System Tests

4.1 Tests for Functional Requirements

This section defines and organizes the functional system tests that ensure the Large Event Management System (LEMS) meets the behavioral requirements described in the SRS document. The test cases directly verify the main functional components identified in SRS Section S.4 and the high-level usage scenarios in Section G.5. Each test is uniquely identified, includes expected inputs and outputs, and is traceable to the corresponding SRS functional requirement and hazard mitigation goal.

The functional tests are divided into the following main areas:

- Ticket Purchase and Payment System
- Bus and Table Registration
- Waivers and Preference Submission
- Role-Based Access Control (RBAC)
- Event Creation and Management

4.1.1 Ticket Purchase and Payment System (FR1)

This area ensures users can securely purchase event tickets and that payments are processed correctly. It addresses requirements related to FR1 in the SRS and hazard H.2.1 (payment failure and financial data breach).

Test Case FR1-T01: Successful Ticket Purchase

1. **Test ID:** FR1-T01-1

   Control: Automatic

   Initial State: User logged in; event active with available tickets.

   Input: User selects an event, chooses one ticket, and enters valid payment details (Stripe sandbox credentials).

   Expected Output: Payment processed successfully; confirmation email and QR ticket generated and stored in the user profile.

   Test Case Derivation: Validates end-to-end workflow from SRS FR1.1 and FR1.2. Ensures secure API integration and correct ticket issuance.

   How Test Will Be Performed: Automated end-to-end test using Jest and Cypress with mock data.

2. **Test ID:** FR1-T01-2

   Control: Automatic

   Initial State: User logged in; same event selected.

Input: User enters invalid credit card number.

Expected Output: Error message displayed ("Payment could not be processed"); transaction aborted; no ticket issued.

Test Case Derivation: Confirms handling of invalid input data and payment validation logic (SRS FR1.3).

How Test Will Be Performed: Mock API test with invalid credentials in Stripe sandbox environment.

3. **Test ID:** FR1-T01-3

   Control: Automatic

   Initial State: Event active with limited ticket quantity.

   Input: Two users attempt to buy the last ticket concurrently.

   Expected Output: One transaction succeeds; second receives "Event Full" message; database capacity correctly enforced.

   Test Case Derivation: Ensures concurrency handling and race condition protection in ticket allocation.

   How Test Will Be Performed: Automated concurrency test simulating two parallel transactions.

4. **Test ID:** FR1-T01-4

   Control: Automatic

   Initial State: User purchased ticket; event data stored.

   Input: User refreshes page or revisits ticket page.

   Expected Output: Ticket information displayed correctly; no duplicate ticket generated.

   Test Case Derivation: Confirms persistence and data retrieval logic for purchased tickets (FR1.4).

   How Test Will Be Performed: Automated front-end validation using Cypress component tests.

Test Case FR1-T02: Ticket Unavailability and Refund Scenarios

1. **Test ID:** FR1-T02-1

Control: Automatic

Initial State: Event has reached full capacity.

Input: User attempts to purchase ticket.

Expected Output: "Event Full" message displayed; transaction blocked; no payment processed.

Test Case Derivation: Validates enforcement of event capacity limits and user feedback (SRS FR1.3).

How Test Will Be Performed: Automated test with mock event database entries.

2. **Test ID:** FR1-T02-2

   Control: Automatic

   Initial State: User successfully purchased ticket.

   Input: Organizer cancels event.

   Expected Output: Refund transaction triggered; user receives refund notification email.

   Test Case Derivation: Confirms proper refund workflow and integration with payment API (SRS FR1.4).

   How Test Will Be Performed: Automated backend test using Stripe test API for refund simulation.

3. **Test ID:** FR1-T02-3

   Control: Automatic

   Initial State: User attempts to repurchase refunded ticket.

   Input: User retries transaction after refund.

   Expected Output: Transaction processed normally if capacity reopens.

   Test Case Derivation: Ensures reusability and recovery after refund workflow.

   How Test Will Be Performed: Backend integration test validating ticket status updates.

Test Case FR1-T03: Payment Security and Error Handling

1. **Test ID:** FR1-T03-1

Control: Automatic

Initial State: User logged in; valid event and payment options available.

Input: User attempts purchase while network connection drops mid-transaction.

Expected Output: System rolls back incomplete transaction; no double charge occurs.

Test Case Derivation: Confirms transactional atomicity (FR1.4).

How Test Will Be Performed: Network interruption simulated during payment call.

2. **Test ID:** FR1-T03-2

   Control: Automatic

   Initial State: User logged in; payment processing API temporarily offline.

   Input: User submits payment request.

   Expected Output: System returns "Payment service unavailable" error; retry option presented.

   Test Case Derivation: Validates graceful error handling and user communication (FR1.4).

   How Test Will Be Performed: Simulated API downtime using mock server responses.

3. **Test ID:** FR1-T03-3

   Control: Automatic

   Initial State: User logged in; payment API active.

   Input: User attempts to alter payment request via browser console.

   Expected Output: Request rejected; unauthorized data tampering prevented.

   Test Case Derivation: Confirms security mechanisms (input validation, token verification) from Hazard H.2.3 mitigation.

   How Test Will Be Performed: API security test through Postman and OWASP ZAP scanning.

—

4.1.2 Bus and Table Registration (FR2)

These tests validate proper seat and transportation assignment during event registration. They directly support SRS requirements FR2.1–FR2.3 and hazard mitigation for H.3.2 (overbooking errors).

Test Case FR2-T01: Successful Bus/Table Registration and Confirmation

1. **Test ID:** FR2-T01-1

   Control: Automatic

   Initial State: User logged in with valid event ticket. Bus and table options available.

   Input: User selects an available bus route and table number.

   Expected Output: System assigns user to chosen bus/table; confirmation message displayed; database updates seat count.

   Test Case Derivation: Validates standard seat assignment workflow from SRS FR2.1.

   How Test Will Be Performed: Automated UI and backend test through Cypress to confirm database entry and confirmation message.

2. **Test ID:** FR2-T01-2

   Control: Automatic

   Initial State: User logged in with multiple available buses.

   Input: User selects bus A; system concurrently processes two users choosing same seat.

   Expected Output: One transaction succeeds; second user receives "Seat Taken" error; database integrity maintained.

   Test Case Derivation: Confirms concurrency management and atomic seat updates (SRS FR2.2).

   How Test Will Be Performed: Automated concurrency test simulating two simultaneous seat reservations.

3. **Test ID:** FR2-T01-3

   Control: Automatic

Initial State: Event database populated with several buses and tables.

Input: User changes seat selection before final confirmation.

Expected Output: Original seat released; new selection confirmed; no duplicates remain.

Test Case Derivation: Ensures seat reallocation logic works as intended (SRS FR2.3).

How Test Will Be Performed: Automated API-level testing verifying previous record removal and reassignment.

Test Case FR2-T02: Overbooking and Capacity Enforcement

1. **Test ID:** FR2-T02-1

   Control: Automatic

   Initial State: Table at full capacity.

   Input: User attempts to select the same table.

   Expected Output: Error message displayed ("Table Full"); user prompted to choose another table.

   Test Case Derivation: Ensures accurate enforcement of capacity constraints (SRS FR2.2).

   How Test Will Be Performed: Automated UI and backend test simulating multiple full-table assignments.

2. **Test ID:** FR2-T02-2

   Control: Automatic

   Initial State: Table nearly full (1 seat remaining).

   Input: Two users select final seat simultaneously.

   Expected Output: First request succeeds; second receives capacity error message; database count remains consistent.

   Test Case Derivation: Verifies concurrency and data integrity during high-load operations.

   How Test Will Be Performed: Automated stress test simulating simultaneous HTTP requests through Jest mock server.

3. **Test ID:** FR2-T02-3

Control: Automatic

Initial State: Bus route capacity reached.

Input: Organizer views bus assignments in Admin Dashboard.

Expected Output: Capacity indicator highlighted in red; "Bus Full" status displayed; no further reservations allowed.

Test Case Derivation: Confirms that UI state accurately reflects database limits (FR2.3).

How Test Will Be Performed: Automated end-to-end test confirming synchronization between backend state and frontend visuals.

Test Case FR2-T03: Cancellations, Waitlists, and Updates

1. **Test ID:** FR2-T03-1

   Control: Automatic

   Initial State: User registered with confirmed table and bus.

   Input: User cancels registration.

   Expected Output: Seat released; table and bus availability incremented by one; confirmation displayed.

   Test Case Derivation: Validates cancellation and reallocation logic per SRS FR2.3.

   How Test Will Be Performed: Automated API test verifying correct record deletion and database updates.

2. **Test ID:** FR2-T03-2

   Control: Automatic

   Initial State: Table and bus fully booked; waitlist enabled.

   Input: User requests to join waitlist.

   Expected Output: User added to waitlist queue; notification displayed confirming enrollment.

   Test Case Derivation: Ensures waitlist creation and management (FR2.3).

   How Test Will Be Performed: Backend simulation using mock users to confirm proper queue ordering.

3. **Test ID:** FR2-T03-3

Control: Automatic

Initial State: Seat becomes available due to cancellation.

Input: Waitlisted user automatically promoted.

Expected Output: User receives notification and confirmation email; seat reassigned successfully.

Test Case Derivation: Confirms automated reassignment and notification mechanism (FR2.3).

How Test Will Be Performed: Automated backend event simulation and notification delivery validation.

—

4.1.3 Waiver and Preference Submission (FR3)

This testing area validates the form submission process for waivers, dietary needs, and accessibility preferences. These tests address FR3.1 and hazard H.4.2 (data loss or unsubmitted forms).

Test Case FR3-T01: Successful Waiver Submission

1. **Test ID:** FR3-T01-1

   Control: Automatic

   Initial State: User logged in with confirmed event ticket.

   Input: User uploads a valid waiver file in PDF format and selects dietary and accessibility preferences.

   Expected Output: System stores waiver file and preference data in user record; confirmation message displayed; email receipt sent.

   Test Case Derivation: Confirms successful upload, form validation, and database persistence (SRS FR3.1).

   How Test Will Be Performed: Automated end-to-end test using Cypress; verification through backend data query.

2. **Test ID:** FR3-T01-2

   Control: Automatic

   Initial State: User logged in; waiver already uploaded.

Input: User attempts to upload an updated waiver document.

Expected Output: Old document replaced with new file; system logs replacement timestamp; confirmation displayed.

Test Case Derivation: Validates update and overwrite functionality (FR3.2).

How Test Will Be Performed: Automated backend test confirming version history and successful replacement.

Test Case FR3-T02: Input Validation and Error Handling

1. **Test ID:** FR3-T02-1

   Control: Automatic

   Initial State: User logged in; waiver submission page open.

   Input: User attempts to upload an invalid file format (e.g., .exe).

   Expected Output: Upload rejected; "Invalid File Type" error displayed; submission disabled until valid file provided.

   Test Case Derivation: Validates input format verification and file-type restrictions (SRS FR3.1).

   How Test Will Be Performed: Automated validation test verifying client-side and server-side restrictions.

2. **Test ID:** FR3-T02-2

   Control: Automatic

   Initial State: User logged in with waiver form active.

   Input: User leaves mandatory fields (dietary preference) blank.

   Expected Output: System displays inline validation message; prevents submission.

   Test Case Derivation: Ensures proper enforcement of required fields (SRS FR3.2).

   How Test Will Be Performed: Front-end validation test using Cypress automated form interaction.

Test Case FR3-T03: Organizer Access and Data Integrity

1. **Test ID:** FR3-T03-1

   Control: Automatic

   Initial State: Organizer logged in; event participant data loaded.

   Input: Organizer views waiver submissions.

   Expected Output: Organizer sees submission statuses ("Submitted," "Pending," "Missing"); sensitive data hidden.

   Test Case Derivation: Confirms proper permission-based visibility and anonymization (SRS FR3.2).

   How Test Will Be Performed: Backend API test verifying restricted data queries and UI display.

2. **Test ID:** FR3-T03-2

   Control: Automatic

   Initial State: Organizer dashboard active; participant waivers available.

   Input: Organizer downloads event-wide waiver report.

   Expected Output: CSV generated containing participant names, waiver status, and dietary/accessibility selections.

   Test Case Derivation: Ensures export functionality aligns with data protection and reporting (FR3.3).

   How Test Will Be Performed: Automated API test confirming file generation and proper field inclusion.

—

4.1.4 Role-Based Access Control (RBAC) (FR4)

This test area verifies that the system correctly enforces user roles and associated permissions for students, administrators, and event organizers. It ensures that access rights, authentication, and authorization are implemented consistently to maintain data security and privacy. These tests align with SRS requirements FR4.1–FR4.3 and mitigate hazards H.5.1 (unauthorized access), H.5.2 (privilege escalation), and H.5.3 (data visibility breach).

Test Case FR4-T01: User Authentication and Login Permissions

1. **Test ID:** FR4-T01-1

   Control: Automatic

   Initial State: User not logged in; login page displayed.

   Input: User enters valid credentials (email and password).

   Expected Output: System authenticates credentials, redirects user to dashboard corresponding to their assigned role.

   Test Case Derivation: Confirms login workflow and authentication success (SRS FR4.1).

   How Test Will Be Performed: Automated test using Cypress and mock authentication API responses.

2. **Test ID:** FR4-T01-2

   Control: Automatic

   Initial State: Login page active.

   Input: User enters incorrect password.

   Expected Output: Error message "Invalid Credentials" displayed; login attempt logged; user remains on login page.

   Test Case Derivation: Ensures error handling and prevention of unauthorized access (FR4.2).

   How Test Will Be Performed: Automated authentication test using invalid credential sets.

3. **Test ID:** FR4-T01-3

   Control: Automatic

   Initial State: User attempts to bypass login page via URL manipulation.

   Input: User directly accesses an admin endpoint (e.g., /admin/dashboard).

   Expected Output: Access denied; redirected to login page.

   Test Case Derivation: Confirms route protection and proper authentication middleware setup.

   How Test Will Be Performed: Security test via Postman and Cypress route validation.

Test Case FR4-T02: Role Enforcement and Privilege Management

1. **Test ID:** FR4-T02-1

   Control: Automatic

   Initial State: Student logged in with base privileges.

   Input: Student attempts to access "Manage Events" admin page.

   Expected Output: Access denied; "Insufficient Permissions" message displayed.

   Test Case Derivation: Confirms enforcement of role restrictions per SRS FR4.2.

   How Test Will Be Performed: Role-based permission validation using mock user tokens and Cypress tests.

2. **Test ID:** FR4-T02-2

   Control: Automatic

   Initial State: Event organizer logged in.

   Input: Organizer attempts to approve or deny waiver submissions.

   Expected Output: Action allowed and recorded in audit log; system confirms update.

   Test Case Derivation: Validates correct privilege execution for organizers (FR4.3).

   How Test Will Be Performed: Backend API test verifying authorization scopes.

3. **Test ID:** FR4-T02-3

   Control: Automatic

   Initial State: Admin logged in; role management page open.

   Input: Admin attempts to modify another user's role (e.g., student $\rightarrow$ organizer).

   Expected Output: Role updated successfully; audit log entry created.

   Test Case Derivation: Ensures privilege escalation occurs only for authorized admin accounts (FR4.3).

   How Test Will Be Performed: Automated backend test verifying restricted database write access.

Test Case FR4-T03: Session Security and Access Persistence

1. **Test ID:** FR4-T03-1

   Control: Automatic

   Initial State: User logged in; session token active.

   Input: User remains idle for 20 minutes.

   Expected Output: Session expires automatically; user redirected to login page.

   Test Case Derivation: Confirms timeout mechanism and token expiry (FR4.3).

   How Test Will Be Performed: Automated timeout simulation test using JWT expiration.

2. **Test ID:** FR4-T03-2

   Control: Automatic

   Initial State: User logged in across multiple devices.

   Input: Admin invalidates all sessions for that user.

   Expected Output: All active sessions terminated; user forced to re-login.

   Test Case Derivation: Validates centralized session revocation and global logout (FR4.3).

   How Test Will Be Performed: Backend simulation using token invalidation endpoint and multi-device login testing.

3. **Test ID:** FR4-T03-3

   Control: Automatic

   Initial State: User with expired token tries accessing protected route.

   Input: HTTP request sent with invalid or expired JWT token.

   Expected Output: "Unauthorized" response; no data returned.

   Test Case Derivation: Confirms backend authentication middleware correctly rejects invalid tokens.

How Test Will Be Performed: Postman API test simulating expired token authentication.

—

4.1.5 Event Creation and Management (FR5)

These tests validate administrative and organizer capabilities to create and manage event listings, per SRS FR5.1–FR5.3 and hazards H.5.1 (incorrect listings) and H.5.3 (data synchronization errors).

Test Case FR5-T01: Event Creation and Publishing

1. **Test ID:** FR5-T01-1

   Control: Automatic

   Initial State: Organizer logged in with event creation privileges.

   Input: Organizer selects "Create Event" and enters valid event data (title, date, location, ticket limit).

   Expected Output: New event record created in the database; confirmation displayed; event visible to attendees.

   Test Case Derivation: Confirms primary event creation flow (SRS FR5.1).

   How Test Will Be Performed: Automated UI and backend test verifying record creation, field population, and confirmation message.

2. **Test ID:** FR5-T01-2

   Control: Automatic

   Initial State: Organizer dashboard active.

   Input: Organizer enters incomplete form (missing required field such as date).

   Expected Output: System prevents submission; displays "Field Required" message; no event created.

   Test Case Derivation: Validates input validation and form integrity (FR5.1).

   How Test Will Be Performed: Automated front-end validation test using Cypress form interaction.

3. **Test ID:** FR5-T01-3

   Control: Automatic

   Initial State: Organizer creating multiple events.

   Input: Organizer attempts to create event with identical name and date.

   Expected Output: System displays "Duplicate Event" warning; prevents creation.

   Test Case Derivation: Ensures event uniqueness constraint enforced in database (FR5.2).

   How Test Will Be Performed: Automated backend test verifying duplicate-check middleware response.

Test Case FR5-T02: Event Editing and Updates

1. **Test ID:** FR5-T02-1

   Control: Automatic

   Initial State: Existing event in database.

   Input: Organizer modifies event date and capacity.

   Expected Output: Changes saved successfully; all linked registrations and tickets updated automatically.

   Test Case Derivation: Confirms cascading updates across dependent systems (FR5.2).

   How Test Will Be Performed: Backend integration test validating field updates and data consistency.

2. **Test ID:** FR5-T02-2

   Control: Automatic

   Initial State: Event published and visible to attendees.

   Input: Organizer updates event description or venue.

   Expected Output: System updates event info in attendee view; notification sent to registered users.

   Test Case Derivation: Verifies synchronization between admin dashboard and attendee UI (FR5.3).

How Test Will Be Performed: Automated end-to-end test confirming data propagation and push notification delivery.

3. **Test ID:** FR5-T02-3

   Control: Automatic

   Initial State: Organizer editing event under heavy traffic load.

   Input: Multiple simultaneous update requests.

   Expected Output: Only latest confirmed update persisted; older requests rejected; no data corruption.

   Test Case Derivation: Ensures concurrency handling in event modification (FR5.4).

   How Test Will Be Performed: Automated concurrency simulation test using API request flooding in Jest.

Test Case FR5-T03: Event Deletion and Archiving

1. **Test ID:** FR5-T03-1

   Control: Automatic

   Initial State: Event published with active registrations.

   Input: Organizer attempts to delete event.

   Expected Output: System prevents deletion; prompts organizer to archive instead.

   Test Case Derivation: Confirms data retention policy and prevention of accidental data loss (FR5.4).

   How Test Will Be Performed: Automated backend test verifying deletion restrictions and user prompts.

2. **Test ID:** FR5-T03-2

   Control: Automatic

   Initial State: Event archived successfully.

   Input: Organizer reopens archived event.

   Expected Output: System restores event status to active; attendee visibility re-enabled.

Test Case Derivation: Ensures proper archive and restore operations (FR5.4).

How Test Will Be Performed: Automated backend database state validation test.

3. **Test ID:** FR5-T03-3

   Control: Automatic

   Initial State: Archived event record exists.

   Input: Admin exports archived event data.

   Expected Output: CSV or PDF export generated successfully with complete event metadata.

   Test Case Derivation: Confirms data export feature functions correctly for historical events (FR5.4).

   How Test Will Be Performed: Automated backend export test verifying file creation and data completeness.

—

Summary: The above test cases comprehensively validate all primary functional requirements outlined in the SRS and address key hazards identified in the Hazard Analysis document. Each test ensures correct behavior of the system under both normal and boundary conditions, supporting verification of system correctness, reliability, and user safety.

4.2 Tests for Nonfunctional Requirements

This section will cover the system tests for Non-Functional Requirements identified in the MacSync SRS Document. The goal of these tests is to metricize the quality attributes and operational characteristics of the system, ensuring it performs its functional requirements to a high standard.

4.2.1 Performance (NFR1)

Test Case NFR1-T01: End-to-End Registration

1. NFR1-T01-1

   Type: Dynamic, Automatic, Performance

Initial State: DB with 5 events, with capacities varying from 50-500, 750 registered users.

Input/Condition: One user completes the registration workflow: browse $\rightarrow$ select ticket $\rightarrow$ pay $\rightarrow$ ticket issued.

Output/Result: End-to-end latency $\leq 2.0\,\text{s}$, average $\leq 1.5\,\text{s}$, zero timeouts.

How test will be performed: Measure the time from starting registration to receiving the ticket confirmation.

2. NFR1-T01-2

   Type: Dynamic, Automatic, Load/Stress

   Initial State: DB with 5 events, with capacities varying from 50-500, 750 registered users.

   Input/Condition: 200 users attempt to complete the full registration workflow concurrently.

   Output/Result: Average end-to-end registration time $\leq 3.0\,\text{s}$, and system successfully processes all registrations without errors or crashes.

   How test will be performed: Simulate 200 users concurrently registering for an event. Collect the total response time for each user and note any delayed transactions.

Test Case NFR1-T02: RSVP/Signups

1. NFR1-T02-1

   Type: Dynamic, Automatic, Performance

   Initial State: Event/session with capacity set (e.g., 100 seats).

   Input/Condition: One user submits an RSVP/sign-up.

   Output/Result: Operation completes in $\leq 1.0\,\text{s}$. Confirmation returned to user.

   How test will be performed: Measure API time from submit to confirmation.

2. NFR1-T02-2

   Type: Dynamic, Automatic, Load/Stress

   Initial State: Event/session with capacity set (e.g., 100 seats).

   Input/Condition: 100 users submit RSVP/sign-up concurrently.

   Output/Result: Average completion time $\leq 1.5\,\mathrm{s}$, make sure there are no duplicate allocations. The final accepted count equals capacity.

   How test will be performed: Simulate 100 concurrent submissions, record response times and verify final counts.

Test Case NFR1-T03: Check-In

1. NFR1-T03-1
   Type: Dynamic, Automatic, Performance

   Initial State: Event with valid tickets issued.

   Input/Condition: Single QR code scan (valid ticket).

   Output/Result: Scan-to-confirmation time $\leq 0.30\,\mathrm{s}$. The staff will receive a valid ticket confirmation

   How test will be performed: Measure time from scan request to API confirmation.

2. NFR1-T03-2
   Type: Dynamic, Automatic, Performance

   Initial State: Event with valid tickets issued.

   Input/Condition: Single QR code scan (invalid ticket).

   Output/Result: Scan-to-disconfirmation time $\leq 0.30\,\mathrm{s}$. The staff will receive a valid ticket disconfirmation

   How test will be performed: Measure time from scan request to API disconfirmation.

4.2.2 Data Integrity (NFR2)

Test Case NFR2-T01: Registration

1. NFR2-T01-1

   Type: Dynamic, Automatic, Integrity

   Initial State: Event with capacity available, With payment provider enabled.

   Input/Condition: Force a *payment failure* during registration

   Output/Result: No rows created in `tickets/registrations` and no balance changes.

   How test will be performed: Execute registration up to payment → inject failure → assert zero side-effects across related tables.

1. NFR2-T01-2

   Type: Dynamic, Automatic, Integrity

   Initial State: Event with capacity available and payment provider enabled.

   Input/Condition: Force a *payment success* during registration

   Output/Result: rows created in `tickets/registrations`, balance changes.
   How test will be performed: Execute registration up to payment → inject success → assert updated rows across related tables.

Test Case NFR2-T02: No Oversell Under Concurrency

1. NFR2-T02-1

   Type: Dynamic, Automatic, Concurrency/Integrity

   Initial State: Event capacity = 100.

   Input/Condition: 120 users attempt to purchase the last 20 tickets *simultaneously.*

   Output/Result: Final **sold = 100**; remaining attempts fail or are waitlisted. Ensure there are no duplicate ticket IDs.

   How test will be performed: Simulate concurrent purchases on a event with a capacity. Ensure, the total tickets sold equals the event capacity

Test Case NFR2-T03: Idempotent Payment Webhooks

1. NFR2-T03-1

   Type: Dynamic, Automatic, Integrity

   Initial State: Pending registration awaiting payment confirmation.

   Input/Condition: Deliver the *same* payment_succeeded webhook 3 times (duplicate delivery).

   Output/Result: Exactly one ticket issued, no double charges

   How test will be performed: Replay identical webhook payloads, assert one ticket.

Test Case NFR2-T04: Admin Edits

1. NFR2-T04-1

   Type: Dynamic, Automatic, Integrity

   Initial State: Existing attendee with linked registration, ticket, and assignments.

   Input/Condition: Admin edits attendee details (e.g., email) through the approved workflow.

   Output/Result: All references (login, receipts, notifications) reflect changes.

   How test will be performed: Perform admin edit, check references.

4.2.3 Usability (NFR3)

Test Case NFR3-T01: App Usage

1. NFR3-T01-1

   Type: Dynamic, Manual, Usability

   Initial State: 10 representative users with no prior experience using MacSync.

Input/Condition: Each user completes the full registration and ticket purchase workflow without assistance.

Output/Result: $\geq$ 99% task completion rate. Average completion time $\leq$ 3 minutes. Users report that the process to register is relatively easy on a post-test survey.

How test will be performed: Conduct a post-test survey to identify how users like the registration process. Collect register time and completion rate.

2. NFR3-T01-2

   Type: Dynamic, Manual, Usability

   Initial State: Deployed MacSync web interface available on desktop and mobile browsers.

   Input/Condition: Users locate and access three common features: Event browsing, Registration history, and View Tickets.

   Output/Result: 100% of users locate all features within 15 seconds each.

   How test will be performed: Time users as they navigate to each feature. Record completion times.

3. NFR3-T01-3

   Type: Dynamic, Manual, Usability

   Initial State: User attempts registration with input errors (e.g. missing field, invalid card, expired session).

   Input/Condition: Simulated controlled failures during form submission.

   Output/Result: Error messages clearly explain the issue and provide actionable recovery steps. Users can visually see these error messages to correct them.

   How test will be performed: Observe users encountering predefined error scenarios and verify recovery time and clarity of messages.

4.2.4 Security Compliance (NFR4)

Test Case NFR4-T01: Role-Based Access Control (RBAC/FBAC)

1. NFR4-T01-1

   Type: Dynamic, Automatic, Security

   Initial State: Users with predefined roles exist in the system.

   Input/Condition: Each role attempts to access restricted endpoints outside its permission scope.

   Output/Result: Unauthorized actions are blocked with 403 Forbidden or 401 Unauthorized responses which means no data leakage occurs.

   How test will be performed: Attempt cross-role actions and verify correct denial responses.

Test Case NFR4-T02: Authentication and Session Security

1. NFR4-T02-1

   Type: Dynamic, Automatic, Security

   Initial State: Login and session management implemented.

   Input/Condition: Attempt login with invalid/valid credentials, expired sessions,

   Output/Result: System rejects all invalid attempts, approves valid attempts. All expired sessions should be prompted to revalidate their session.

   How test will be performed: Automated tests simulate invalid, valid and expired sessions. Confirm appropriate approval and rejection.

Test Case NFR4-T03: Sensitive Data Protection

1. NFR4-T03-1

   Type: Static + Dynamic, Security

   Initial State: Database entries with user and payment information.

Input/Condition: Inspect data for exposure of personal information or financial details.

Output/Result: No sensitive data (passwords, payment tokens) stored in plaintext. all sensitive data should be encrypted.

How test will be performed: Review database schema and sample dumps for plaintext values.

4.2.5 Reliability (NFR5)

Test Case NFR5-T01: System Uptime

1. NFR5-T01-1

   Type: Dynamic, Automatic, Reliability

   Initial State: MacSync deployed on the staging environment.

   Input/Condition: Continuous uptime monitoring over a 7-day period.

   Output/Result: System uptime $\geq 99\%$ with no single outage exceeding 5 minutes.

   How test will be performed: Enable automatic monitoring every 60 seconds. Record uptime percentage and log all downtime intervals.

4.2.6 Auditability (NFR6)

Test Case NFR6-T01: Logging

1. NFR6-T01-1

   Type: Dynamic, Automatic, Audit

   Initial State: System configured with active logging for all payment and registration operations.

   Input/Condition: User completes a registration and payment transaction.

   Output/Result: Each transaction automatically generates an audit log entry containing timestamp, user ID, event ID, payment ID, and status.

How test will be performed: Perform a sample registration and verify that a corresponding audit log entry is created in the system's logging database

2. NFR5-T01-2

   Type: Dynamic, Automatic, Audit

   Initial State: Admin and Organizer accounts active in the system. and system configured with active logging

   Input/Condition: Admin performs actions such as modifying event details, refunding tickets, or changing user roles.

   Output/Result: Each action is logged with actor identity, timestamp, previous and new values, and action type.

   How test will be performed: Execute sample admin operations and confirm that log entries capture all relevant details, ensuring traceability.

Summary:
The tests above layout a detailed test plan for testing the non-functional requirements in the SRS document. Each test ensures clear input and outputs for tests to ensure separation for failed and passed test cases.

4.3 Traceability Between Test Cases and Requirements

Table 2: Traceability Between Test Cases and Requirements

Requirement ID	**Requirement Description (SRS Reference)**	**Associated Test Cases**
FR1	Ticket Purchase and Payment System. Users can purchase event tickets, process payments, handle invalid inputs, and receive confirmations.	FR1-T01-1–4, FR1-T02-1–3, FR1-T03-1–3

Table 2 (continued)

Requirement ID	**Requirement Description (SRS Reference)**	**Associated Test Cases**
FR2	Bus and Table Registration. Supports seat selection, waitlists, and capacity enforcement.	FR2-T01-1–3, FR2-T02-1–3, FR2-T03-1–3
FR3	Waivers and Preferences. Allows users to submit waivers and preferences and stores data securely for organizer access.	FR3-T01-1–2, FR3-T02-1–2, FR3-T03-1–2
FR4	Role-Based Access Control (RBAC/FBAC). Verifies authentication, authorization, and correct privilege handling.	FR4-T01-1–3, FR4-T02-1–3, FR4-T03-1–3
FR5	Event Creation and Management. Enables organizers to create, edit, archive, and manage events.	FR5-T01-1–3, FR5-T02-1–3, FR5-T03-1–3
NFR1	Performance. Measures latency and throughput during registration, RSVP, and check-in.	NFR1-T01-1–2, NFR1-T02-1–2, NFR1-T03-1–2
NFR2	Data Integrity. Ensures no inconsistent or duplicate data appears under failure or concurrency.	NFR2-T01-1–2, NFR2-T02-1, NFR2-T03-1, NFR2-T04-1
NFR3	Usability. Evaluates ease of use, completion rate, and error recovery in user testing.	NFR3-T01-1–3

Table 2 (continued)

Requirement ID	Requirement Description (SRS Reference)	Associated Test Cases
NFR4	Security Compliance. Checks RBAC enforcement, authentication, and protection of sensitive data.	NFR4-T01-1, NFR4-T02-1, NFR4-T03-1
NFR5	Reliability. Verifies uptime and system stability under continuous use.	NFR5-T01-1
NFR6	Auditability. Ensures all registration, payment, and event actions are logged for traceability.	NFR6-T01-1–2

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Streamliners. Development plan, 2025a. URL https://github.com/4G06-Streamliners/MacSync/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf.

Streamliners. Hazard analysis, 2025b. URL https://github.com/4G06-Streamliners/MacSync/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf.

Streamliners. Module interface specification, 2025c. URL https://github.com/4G06-Streamliners/MacSync/blob/main/docs/Design/SoftDetailedDes/MIS.pdf.

Streamliners. Software requirements specification, 2025d. URL https://github.com/4G06-Streamliners/MacSync-SRS/blob/main/index.pdf.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

1. **What went well while writing this deliverable?**

   Prerna: During this deliverable, what went well was that from the start we were able to divide the workload evenly by looking through the sections of the VnV Plan and observing which sections were longer and had more workload and which were smaller and lighter. This way, no one was overloaded, and dependencies between sections were identified early on, making it easier to work iteratively rather than rushing before the deadline.

   Ali: The division of work went well as we were able to organize responsibilities quickly, which set the tone for the rest of the deliverable. We also identified which sections depended on others, which helped us work efficiently and submit everything on time.

   Mahad: As a group, we were able to finish our assigned sections well in advance. Because of this, we had more time for peer review and were able to refine our first draft of the VnV deliverable to a much higher standard.

   Farhan: We organized the milestone effectively by prioritizing sections with dependencies first, which made it easier for everyone to stay on track. This structured approach helped us stay coordinated, avoid blockers, and maintain a smooth workflow throughout the deliverable.

   Abyan: What went well was the clarity in communication throughout the team. We consistently shared progress updates and provided feedback, which made it easier to stay aligned. Using GitHub for document versioning also helped us manage revisions efficiently and avoid overlapping work.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**

   Prerna: Since we experienced dependency issues in the last deliverable, we were better prepared this time and identified them early for the most part. Some sections still had unforeseen dependencies closer to the deadline, but we resolved these through team communication. While writing the functional test cases, it was initially difficult to determine how detailed they should be, but after research and discussion, focusing on major functional requirements worked best.

Ali: A challenge was the lack of clear guidelines or lecture content explaining how to fill out certain sections. I resolved this by using my creative freedom and tailoring my approach to best fit the nature of our project.

Mahad: A main pain point was designing test cases for non-functional requirements (NFRs). Since we are still in the early development phase, envisioning concrete performance or reliability tests was difficult. We resolved this by creating high-level test ideas tied to the system's expected behaviour and refining them later.

Farhan: Some section descriptions were vague, making it unclear what level of detail was expected. To resolve this, I referred to examples from past capstone teams and our instructor's templates to ensure our content matched the intended goals.

Abyan: One issue was coordinating document formatting and ensuring consistency in writing tone across sections written by different team members. I helped resolve this by proofreading and reformatting sections for consistency and standardizing the technical terminology used throughout the plan.

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

   - **Mahad and Farhan:** Need to strengthen understanding of automated backend testing using Jest and Supertest, particularly in mocking database queries and validating API endpoints for correctness and performance.
   - **Abyan and Ali:** Need to gain experience with frontend component and interaction testing using Vitest and React Testing Library, focusing on simulating user actions and verifying UI state transitions.
   - **Prerna:** Needs to enhance skills in documentation traceability, structured review facilitation, and stakeholder validation techniques to ensure all system requirements are verified and validated

effectively.

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?**

   - **Mahad and Farhan:** Plan to follow the official NestJS testing documentation and Supertest tutorials, while also experimenting directly within the backend repository to implement API tests. They chose this approach because it allows immediate hands-on application aligned with their development tasks.
   - **Abyan and Ali:** Will learn through guided examples from the Vitest and React Testing Library documentation and supplement this with online workshops or tutorials on TanStack Router integration testing. They prefer this approach for its visual and practical learning focus, helping them apply testing directly to frontend modules.
   - **Prerna:** Will study IEEE verification standards and review academic examples of validation reports, complemented by supervisor feedback on documentation clarity. This method ensures professional alignment and improves the quality of written verification deliverables.