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**
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	**description**
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

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

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

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

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walk-throughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walk-through. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

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

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

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

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   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:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

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 → select ticket → pay → ticket issued.

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

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 ≤ 1.5 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 ≤ 0.30 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 ≤ 0.30 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
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

Table 2 (continued)

Requirement ID	**Requirement Description (SRS Reference)**	**Associated Test Cases**
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
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

Author Author. System requirements specification. https://github.com/..., 2019.

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.