System Verification and Validation Plan for Software Engineering

Team #18, Gouda Engineers
Aidan Goodyer
Jeremy Orr
Leo Vugert
Nathan Perry
Tim Pokanai

October 27, 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

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

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

The following section will outline a verification and validation plan to follow throughout the project timeline. Since our development of the project is executed in multiple sequential steps, this section will outline the verification and validation plans, as well as the tools and methods used for each milestone component leading up to our final software solution. This section will also list the entities that will participate in the verification and validation steps. First we will outline who will be involved in the verification and validation team. Then we will discuss the verification plans of the SRS, design, VnV Plan, implementation, and the automatic tools we use for testing and verification. Lastly, we will discuss how we will perform the validation of our software.

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

3.1 Verification and Validation Team

- **Aidan Goodyer**,**Jeremy Orr**,**Nathan Perry**,**Timothy Pokanai**,**Leo Vugert**:

  The five members of the capstone team will split the testing evenly among themselves. All members will be equally expected to contribute to the synthesis of both unit and system tests as well as setting up automated tests down the line. Since the author of the System Testing plan was **Jeremy Orr** and the author of the Unit Testing plan will be **Nathan Perry**, they will be considered the head of system testing and unit testing respectively. Although this does not require them to necessarily produce more tests, they will be expected to be prepared to provide guidance to other group members if they need it.

- **Dr. Henry Szechtman**, **Dr. Anna Dvorkin-Gheva**:

  The two supervisors of this project will be involved in the testing process as if they were early access users. Given that they represent a demographic very similar to what our actual users will look like (Albeit with a much better understanding of the data), their feedback will be extremely useful in understanding how well our system will be recieved by users. They will perform both User Acceptance Testing and and Usability testing on our system; trying out the various functionalities and commenting on if they think it is satisfactory and understandable for the general user base.

3.2 SRS Verification

will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

3.2.1 SRS Verification Approaches

- **Peer Reviewers:** Our first approach for SRS verification is reviewing the peer review issues created by our classmates. This is not the most rigorous or systematic but allows us to evaluate feedback on our SRS and based on these comments, determine if the SRS is consistent, correct, feasible and complete or if changes are needed.

- **Unstructured Supervisor SRS Review:** Our second approach for SRS verification is to provide our SRS document to our supervisor, Dr. Henry Szechtman and allow him to make comments on it. This approach is also quite unstructured but is still useful in our verification process. By giving our supervisor the time to look over the SRS, it will elicit any large gaps between our understanding of his goals for the system and our interpretation of these goals during our meetings to elicit said requirements. In other words, this will show us any glaring issues in our requirements from the perspective of our supervisor.

- **Structured Supervisor SRS Review:** Our third approach for SRS verification also involves our supervisor but in a much more structured manner. Our approach to this review would be a meeting in which we present to him the key aspects of the SRS and ask him if he sees any issues with these key sections. The main sections (note these do not necessarily refer to specific sections in the SRS template itself) to be covered, in descending order of priority are: Functional Requirements and Use Cases, Project Constraints and Scope, Non-Functional Requirements, Project Drivers and Project Issues. During this review we would ask him things like: Is this section complete? Is there anything missing? Does the way we laid out these requirements seem correct to you? Do these requirements seem feasible to you? Is this in accordance with your conception of what the system will look like?

- **Structured Team SRS Review:** Our final approach for SRS verification will be a review in a much similar format to the structured supervisor review but only with the five members of our team. We

would go over the same key aspects of the SRS and ask mostly the same questions regarding these sections as we would in our supervisor review. Doing an internal review like this will be helpful as although the SRS was generated collectively, the actual writing of the requirements was done in a very individual and compartmentalized way and this gives each member an opportunity to closely examine the key parts of our requirements and provide and criticisms, issues or additions that they feel are necessary to each section.

3.2.2 SRS Verification Checklist

Below is a general checklist of questions that our SRS should satisfy (i.e. the answer should be yes):

- Are all of the functional requirements necessary, feasible, complete and provide unique value to the system?

- Do the functional requirements cover every expected function of the system?

- Are all of the use cases necessary, feasible and complete?

- Do the existing use cases cover every expected function of the system?

- Does the SRS cover every relevant project constraint such that there is no unaccounted for aspect of the environment that may prevent us from implementing some of the system's functionality?

- Are all project constraints accurately described such that it is clear what exactly is constrained and it is understandable how this may affect the system?

- Is the scope properly defined such that all project components have been accounted for in the scope?

- Have all relevant out of scope components been listed and rationale provided for why they are out of scope?

- Are all non-functional requirements necessary, feasible, complete and provide unique value to the system?

- For non-functional requirements that are not completely necessary, is their value large enough to justify their existence?

- Have all obligatory non-functional requirements been accounted for, specifically those relating to legality, security or compliance that are imposed by entities outside of the system?

3.3 Design Verification

For the software design process, the goal of verification of this stage is to ensure our design correctly and completely satisfies the SRS before we start coding. The verification will ensure that during the design process we were not focused on "building the right thing", but "building the thing right". The following subsections will describe the verification techniques and checklist to be used during design verification.

3.3.1 Design Verification Techniques

- **Structured Supervisor Design Walkthrough:** The first design verification technique will be our team leading a structured walkthrough of our design documents to our supervisors, Dr. Henry Szechtman and Dr. Anna Dvorkin-Gheva. This will allow our team to present our relevant technical design documents in a clear and logically structured manner to provide transparency and room for suggestions on our design decisions, as well as communicate technical aspects as clearly as possible to reviewers with less technical knowledge.

- **Peer Review from Classmates:** The second design verification technique will be the peer reviews we receive from our classmates. Again, this technique is not expected to be the most rigorous, but another team will review our design process and contribute feedback which provides us an opportunity to verify our design process to make sure it is consistent with our SRS.

- **Semi-Structured Team Design Inspection:** Our third technique we will use for verifying our design process will be our team doing a semi-structured inspection of our design documents. Since our team is doing all the design and development as the only software engineers involved in this project, we only need to keep this inspection among us

as the development team. Doing a semi-structured inspection together would entail going through our relevant technical design documents in a logically structured manner which would ensure our design process aligns with the SRS and there are no ambiguities in the process. Additionally, the semi-structured format would allow room for open-ended discussion on design decisions that could be added, modified, or improved.

- **Low Fidelity UI Prototype Verification:** Finally our last design verification technique will be creating a low fidelity prototype specifically for the UI component using a tool like Figma or actually coding an interface, allowing our team and our supervisors to test the prototype. A big part of our design process is designing a more user-friendly and accessible UI than the existing one FRDR offers. Conducting a prototype test with our entire verification and validation team is critical, considering we would have the input and suggestions of our two supervisors who interface with FRDR on a regular basis.

3.3.2 Design Verification Checklist

Below is a general checklist of questions that our Design Process should satisfy (i.e. the answer should be yes):

- Does the design fulfill all functional requirements stated in the SRS?

- Does the design adhere to all non-functional requirements devised in the SRS?

- Is the design feasible to implement considering the given timeline, resources, and constraints of the project?

- Are all subsystems, modules, and components well-defined and logically decomposed within our overall system?

- Are the APIs and interfaces between our components clearly specified with their inputs, outputs, and data types?

- Is there traceability from our system components and architecture all the way back to our requirements?

- Do all diagrams reflect the architecture of our design consistently and accurately?

- Do the UIs follow usability and accessibility guidelines?

- Has the verification and validation team approved our design process?

[Plans for design verification —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

3.4   Verification and Validation Plan Verification

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

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 walkthroughs, code inspection, static analyzers, etc. —SS]
[The final class presentation in CAS 741 could be used as a code walkthrough. 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

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

3.7 Software Validation

4 System Tests

This section defines the tests used to verify the system's compliance with the Functional Requirements specified in Section 9.1 of the SRS. Each test ensures that the implemented features meet the expected behavior and fit criteria described in the Software Requirements Specification.

The subsections are organized according to the main areas of functionality of the platform:

- Data Filtering and Browsing (FUNC.R.1–R.3)

- Behavioral Analysis and Visualization (FUNC.R.4–R.5)

- Data Downloading and Accessibility (FUNC.R.6–R.7)

4.1 Tests for Functional Requirements

This section provides detailed tests for all functional requirements. Each test includes a test ID, control type, initial conditions, inputs, expected outputs, test case derivation, and execution description. References to the SRS are included for traceability.

4.1.1 Data Filtering and Browsing

These tests verify that users can filter, browse, and query data based on predefined labels, natural language queries, and preset datasets. They validate the backend (FastAPI + PostgreSQL) integration with the frontend filtering and NLP modules.

Test 1: Verify Filtering by Predefined Labels

- **Related Requirement:** FUNC.R.1

- **Test ID:** FR1-Filter-Labels

- **Control:** Automatic (API + UI test)

- **Initial State:** The database contains labeled behavioral trials with fields such as `behavior_label`, `trial_id`, and `session_id`.

- **Input:** User selects a filter, e.g., `behavior_label = 'checking'`.

- **Output:** Returned dataset contains only trials where `behavior_label == 'checking'`; 100% match accuracy.

- **Test Case Derivation:** Since the fit criterion requires perfect accuracy between filter condition and output, the expected output is all rows meeting the filter condition.

- **How test will be performed:** Use an automated test script (e.g., Postman or PyTest) to send GET requests to `/api/filter?behavior_label=checking`. Compare returned data with a query directly run on the database to verify no mismatched rows.

Test 2: Verify NLP-Based Query Returns Correct Dataset

- **Related Requirement:** FUNC.R.2

- **Test ID:** FR2-NLP-Query

- **Control:** Manual + Automatic hybrid (human semantic check)

- **Initial State:** NLP model and API are running. Database populated with varied behavioral patterns.

- **Input:** User enters: "Show me trials with strong checking behavior after 5 injections."

- **Output:** One dataset is returned that matches this semantic filter.

- **Test Case Derivation:** Based on the SRS fit criterion, one dataset should always be returned for a valid query, containing the subset matching the interpreted condition.

- **How test will be performed:** Send query via UI or API. Validate that the system outputs one dataset (non-empty). A human tester verifies semantic correctness (dataset contextually fits query meaning).

Test 3: Verify Predefined Data Categories Available for Browsing

- **Related Requirement:** FUNC.R.3

- **Test ID:** FR3-Predefined-Sets

- **Control:** Manual (UI check)

- **Initial State:** System deployed with at least 3 predefined sets (e.g., "Exploration Trials", "Compulsive Behavior", "Control Trials").

- **Input:** User opens the "Browse" interface.

- **Output:** At least 3 predefined datasets appear for selection.

- **Test Case Derivation:** Requirement specifies a minimum of 3 sets available for browsing.

- **How test will be performed:** Manually navigate to the Browse page and confirm at least 3 datasets are displayed and accessible.

4.1.2 Behavioral Analysis and Visualization

These tests ensure that behavioral categorizations and visual representations are correctly generated for user-selected data subsets.

Test 4: Verify Each Trial Has Behavioral Categorization

- **Related Requirement:** FUNC.R.4

- **Test ID:** FR4-Behavior-Metrics

- **Control:** Automatic (backend validation)

- **Initial State:** Database populated with trial records.

- **Input:** Query any trial dataset.

- **Output:** Each trial entry includes at least one behavioral category field (e.g., "checking", "homebase", etc.).

- **Test Case Derivation:** Requirement demands every trial have a behavior category, so absence of any `NULL` category field indicates success.

- **How test will be performed:** Run automated query validation across dataset ensuring no missing categorization values.

Test 5: Verify Visuals Are Generated for Each Result Set

- **Related Requirement:** FUNC.R.5

- **Test ID:** FR5-Visualization

- **Control:** Automatic (UI rendering check)

- **Initial State:** User has filtered dataset (non-empty result).

- **Input:** User clicks "Generate Visualization".

- **Output:** Visualization component renders trajectory or metric chart corresponding to the dataset.

- **Test Case Derivation:** The fit criterion specifies at least one visual per result set; therefore, successful rendering of a chart or trajectory fulfills this requirement.

- **How test will be performed:** Use UI test automation (e.g., Selenium) to trigger visualization and check that the chart container loads successfully (non-empty DOM element).

4.1.3 Data Downloading and Accessibility

These tests confirm that datasets and visuals can be downloaded correctly and that the platform maintains global accessibility.

Test 6: Verify Data Download and Integrity

- **Related Requirement:** FUNC.R.6

- **Test ID:** FR6-Download

- **Control:** Automatic

- **Initial State:** Dataset generated and displayed.

- **Input:** User clicks "Download CSV" or "Download Visualization".

- **Output:** File downloaded successfully; first 100 rows match database query results.

- **Test Case Derivation:** Fit criterion requires verification against the first 100 rows; thus, equality of the first 100 entries validates correctness.

- **How test will be performed:** Automated script downloads dataset and compares first 100 entries with direct database query output.

Test 7: Verify Global Accessibility and Performance

- **Related Requirement:** FUNC.R.7

- **Test ID:** FR7-Accessibility

- **Control:** Automatic (load testing + global simulation)

- **Initial State:** Deployed web system with CDN enabled.

- **Input:** Users (or simulated clients) from 3+ regions (e.g., North America, Europe, Asia) access the site.

- **Output:** Average page load time < 4 seconds across all regions; no functional degradation.

- **Test Case Derivation:** Requirement defines measurable metric (4 seconds); success if timing threshold not exceeded.

- **How test will be performed:** Use tools such as Lighthouse, GTmetrix, or JMeter with geo-distributed endpoints to measure average response time and record metrics.

4.2   Tests for Nonfunctional Requirements

This section defines the test procedures for verifying the nonfunctional requirements as defined in Sections 10–17 of the SRS. The tests include evaluations of appearance, usability, performance, robustness, scalability, maintainability, and compliance.

Since nonfunctional requirements often describe qualities rather than discrete functions, many tests in this section are qualitative or metric-based evaluations. Several involve user studies, performance measurement, and static reviews instead of strict pass/fail results.

4.2.1   Look and Feel Requirements

These tests ensure that the interface design aligns with the intended audience (non-technical researchers) and provides a simple, familiar, and intuitive user experience.

Test 1: Interface Resemblance to Webstore/Boutique

- **Related Requirement:** APP.R.1

- **Test ID:** NFR1-UI-Appearance

- **Type:** Static, Manual (Usability Review)

- **Initial State:** Functional web interface deployed.

- **Input/Condition:** UI inspected by 3 non-technical evaluators.

- **Output/Result:** 80% of reviewers confirm the interface resembles a webstore-style interface with clear query "packages."

- **How test will be performed:** Conduct a design walkthrough where users compare the layout to an e-commerce platform. Collect ratings via short usability survey (Appendix A).

Test 2: Simplicity of Interface Presentation

- **Related Requirement:** APP.R.2

- **Test ID:** NFR2-UI-Simplicity

- **Type:** Static, Manual (Heuristic Evaluation)

- **Initial State:** All major features visible on homepage.

- **Input/Condition:** Evaluators perform inspection for UI density and feature visibility.

- **Output/Result:** Interface should not exceed 6 visible interactive elements on the main view; evaluators report no intimidation or confusion.

- **How test will be performed:** Use heuristic evaluation checklist for cognitive load and visual complexity; summarize reviewer feedback quantitatively.

Test 3: Familiar Filtering Presentation

- **Related Requirement:** APP.R.3

- **Test ID:** NFR3-UI-Familiarity

- **Type:** Static, Manual

- **Initial State:** Filtering and search panel implemented.

- **Input/Condition:** Users asked to find a data subset using filter controls.

- **Output/Result:** 90% of participants find the filters intuitive and similar to an online storefront.

- **How test will be performed:** Conduct usability test with 5 participants. Record task completion time and perceived ease of use (Likert scale).

4.2.2 Usability and Humanity Requirements

These tests measure user learning time, ease of use, and understandability.

Test 4: Ease of Use (Floor and Ceiling)

- **Related Requirements:** EOU.R.1, EOU.R.2

- **Test ID:** NFR4-EaseOfUse

- **Type:** Dynamic, Manual (User Study)

- **Initial State:** Functional system with packaged queries and NLP interface.

- **Input/Condition:** New users complete a task set (select prepackaged query, perform NLP search, generate visualization).

- **Output/Result:** 100% task completion; average total completion time < 5 minutes for basic operations.

- **How test will be performed:** Conduct usability test with 5 nontechnical participants. Record success rate and completion time per task.

Test 5: Learning Curve Assessment

- **Related Requirement:** LEA.R.1

- **Test ID:** NFR5-LearningCurve

- **Type:** Dynamic, Manual

- **Initial State:** Fully functional platform; user documentation accessible.

- **Input/Condition:** Participants with no prior system experience attempt to complete key workflows.

- **Output/Result:** Average learning curve < 30 minutes; all users can independently perform search and visualization tasks.

- **How test will be performed:** Measure total time for users to reach proficiency (defined as completing 3 tasks unassisted).

Test 6: Understandability and Terminology Audit
- **Related Requirements:** UAP.R.1, UAP.R.2

- **Test ID:** NFR6-Understandability

- **Type:** Static, Manual (Content Review)

- **Initial State:** Interface text finalized.

- **Input/Condition:** Non-technical users read through all on-screen text.

- **Output/Result:** At least 90% of all interface text rated "understandable" by users.

- **How test will be performed:** Conduct terminology walkthrough; classify all words/phrases as technical or intuitive. Replace technical terms as necessary.

Test 7: Accessibility Audit
- **Related Requirement:** ACC.R.1

- **Test ID:** NFR7-Accessibility

- **Type:** Static + Automated

- **Initial State:** Frontend deployed.

- **Input/Condition:** Run Lighthouse, Axe, or WAVE accessibility audit tools.

- **Output/Result:** Accessibility score $\geq 90/100$; ARIA labels and alt-text coverage 100%.

- **How test will be performed:** Automated scan using accessibility tools; manual confirmation of ARIA compliance and alt-text for media.

4.2.3 Performance and Reliability Requirements

These tests verify latency, throughput, fault-tolerance, and data integrity.

Test 8: Query Speed and Latency Benchmark

- **Related Requirements:** SAL.R.1, SAL.R.2

- **Test ID:** NFR8-Performance

- **Type:** Dynamic, Automatic

- **Initial State:** Server and database running under normal load.

- **Input/Condition:** Execute 100 queries of varying sizes up to 5,000 records.

- **Output/Result:** Average query time $<$ 2 seconds; request latency $<$ 100 ms.

- **How test will be performed:** Automated benchmark using JMeter or Locust. Generate graph of response time vs. query size.

Test 9: Fault-Tolerance and Logging Verification

- **Related Requirements:** ROFT.R.1, ROFT.R.2

- **Test ID:** NFR9-Robustness

- **Type:** Dynamic, Automatic

- **Initial State:** Backend running.

- **Input/Condition:** Submit invalid input payloads and simulate network interruptions.

- **Output/Result:** Errors logged without system crash; user receives appropriate feedback message.

- **How test will be performed:** Automated injection of malformed JSON; verify log entries and system stability.

Test 10: Capacity and Scalability Evaluation

- **Related Requirements:** CAP.R.1, CAP.R.2, SOE.R.2

- **Test ID:** NFR10-Capacity

- **Type:** Dynamic, Automatic

- **Initial State:** System deployed in cloud environment.

- **Input/Condition:** Simulate 250 concurrent users performing queries.

- **Output/Result:** Throughput $\geq$ 5,000 transactions/hour; performance efficiency $\geq$ 80%.

- **How test will be performed:** Use load testing tools (Locust, K6) to simulate concurrent access and measure throughput metrics.

4.2.4   Maintainability and Support Requirements

Test 11: Maintainability Code Review

- **Related Requirements:** MAI.R.1, MAI.R.2

- **Test ID:** NFR11-Maintainability

- **Type:** Static (Code Walkthrough)

- **Initial State:** Source code and documentation complete.

- **Input/Condition:** Independent developer reviews repository structure and documentation.

- **Output/Result:** Reviewer able to build and extend system within 1 hour using provided documentation.

- **How test will be performed:** Conduct peer code walkthrough; independent participant attempts new dataset integration per SRS instructions.

Test 12: User Manual Review

- **Related Requirement:** SUP.R.1

- **Test ID:** NFR12-Documentation

- **Type:** Static, Manual

- **Initial State:** User manual drafted.

- **Input/Condition:** Review manual completeness and clarity.

- **Output/Result:** 90% of user study participants rate the manual as "clear" or better.

- **How test will be performed:** Conduct documentation walkthrough with usability test participants; collect feedback ratings.

4.2.5 Security and Compliance Requirements

Test 13: Data Integrity and Abuse Prevention

- **Related Requirement:** INT.R.1, IMM.R.1, IMM.R.2

- **Test ID:** NFR13-Security

- **Type:** Dynamic, Automatic

- **Initial State:** Backend deployed behind HTTPS.

- **Input/Condition:** Attempt SQL injection, code injection, and rate-limit violations.

- **Output/Result:** All attacks blocked; valid error message returned; rate-limiting enforced at 100 requests/minute.

- **How test will be performed:** Perform penetration testing using OWASP ZAP; verify all malicious attempts logged and mitigated.

Test 14: Standards and Licensing Compliance

- **Related Requirements:** LEG.R.1, STA.R.1, STA.R.2

- **Test ID:** NFR14-Compliance

- **Type:** Static, Manual Review

- **Initial State:** Code repository finalized.

- **Input/Condition:** Review dependency list and license headers.

- **Output/Result:** All third-party libraries under compatible licenses; repository includes MIT license and citation requirements.

- **How test will be performed:** Conduct static code inspection; verify license documentation for all dependencies.

4.3 Traceability Between Test Cases and Requirements

Table 1 provides a traceability matrix showing the correspondence between each Software Requirements Specification (SRS) requirement and the test cases defined in Sections 4.1 and 4.2. Each requirement identifier (FUNC.R.#, APP.R.#, etc.) is mapped to one or more test case identifiers that verify its implementation or satisfaction.

Table 1: Traceability Between Test Cases and Requirements

Requirement ID	Associated Test Case(s)
FUNC.R.1	ST-FR1.1 Filter Accuracy Test
FUNC.R.2	ST-FR2.1 NLP Query Response Test
FUNC.R.3	ST-FR3.1 Predefined Dataset Browsing Test
FUNC.R.4	ST-FR4.1 Behavioural Categorization and Metrics Test
FUNC.R.5	ST-FR5.1 Visualization Generation Test
FUNC.R.6	ST-FR6.1 Data Download and Export Verification Test
FUNC.R.7	ST-FR7.1 Global Accessibility and Latency Test
APP.R.1	ST-NFR1.1 Interface Familiarity Inspection

Requirement ID	Associated Test Case(s)
APP.R.2	ST-NFR1.2 Visual Simplicity Inspection
APP.R.3	ST-NFR1.3 Filter Interface Familiarity Test
EOU.R.1	ST-NFR2.1 Ease of Use (Low Floor) Usability Study
EOU.R.2	ST-NFR2.2 Ease of Use (Low Ceiling) Usability Study
LEA.R.1	ST-NFR3.1 Learning Curve Evaluation Test
LEA.R.2	ST-NFR3.2 Visualization Guidance Test
UAP.R.1	ST-NFR4.1 Technical Abstraction Inspection
UAP.R.2	ST-NFR4.2 Terminology Familiarity Review
ACC.R.1	ST-NFR5.1 Accessibility Compliance Test
SAL.R.1	ST-NFR6.1 Query Response Time Benchmark
SAL.R.2	ST-NFR6.2 Network Latency Benchmark
POA.R.1	ST-NFR7.1 Data Accuracy Cross-Check
POA.R.2	ST-NFR7.2 Database Consistency Test
ROFT.R.1	ST-NFR8.1 Input Error Logging Test
ROFT.R.2	ST-NFR8.2 Backend Fault Tolerance Test
CAP.R.1	ST-NFR9.1 Concurrent User Load Test
CAP.R.2	ST-NFR9.2 Data Volume Stress Test
SOE.R.1	ST-NFR10.1 Module Integration Test
SOE.R.2	ST-NFR10.2 Performance Under Load Test
LON.R.1	ST-NFR11.1 Reliability and Maintenance Longevity Test
LON.R.2	ST-NFR11.2 Cross-OS Compatibility Test
EPE.R.1	ST-NFR12.1 Desktop Environment Performance Test
EPE.R.2	ST-NFR12.2 Multi-OS Execution Verification Test
EPE.R.3	ST-NFR12.3 Environmental Condition Simulation
WE.R.1	ST-NFR13.1 Browser Compatibility Test
IWAS.R.1	ST-NFR14.1 API Communication Protocol Test
PROD.R.1	ST-NFR15.1 Docker/Kubernetes Deployment Test
PROD.R.2	ST-NFR15.2 Repository Update Verification Test
REL.R.1	ST-NFR16.1 Versioning Convention Test

Requirement ID	Associated Test Case(s)
REL.R.2	ST-NFR16.2 Release Consistency Verification Test
MAI.R.1	ST-NFR17.1 Documentation Maintainability Inspection
MAI.R.2	ST-NFR17.2 Dataset Integration Maintenance Test
SUP.R.1	ST-NFR18.1 User Manual Verification Test
SUP.R.2	ST-NFR18.2 Self-Support Functionality Test
ADA.R.1	ST-NFR19.1 Platform Adaptability Test
INT.R.1	ST-NFR20.1 Data Integrity Protection Test
IMM.R.1	ST-NFR21.1 Security Attack Resistance Test
IMM.R.2	ST-NFR21.2 Rate Limiting and HTTPS Enforcement Test
LEG.R.1	ST-NFR22.1 License Compliance Review
STA.R.1	ST-NFR23.1 FRDR Standards Compliance Test
STA.R.2	ST-NFR23.2 Ethical Use of Data Verification Test

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:

25

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?

The following is a list of straightforward and open ended questions, good for both clear answers and to allow room for discussion, all of which will be considered for a usability survey.

- Were the system's features and functionalities easy to find and intuitively placed?

- Did the system's functionalities respond quickly to your input?

- Were you aware of what the system was doing at all times and did you receive feedback after performing an action?

- Were the search and webstore functionalities easy to use?

- How well do the search filters satisfy your needs?

- Are the system's information display and data visualization clear and well organized?

- What were your least favourite parts of using the system to complete a workflow?

- On a scale of one to five, how easy was it to learn how to use the system (one meaning impossible and five meaning easy)?

- On a scale of one to ten, how was your experience using the system's UI (one meaning not user-friendly at all and ten meaning very user-friendly)?

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

Appendix — Reflection

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

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

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

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

1. What went well while writing this deliverable?

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

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

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?