System Verification and Validation Plan for Software Engineering

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

November 6, 2025

Revision History

Date	Version	Notes
Oct 27 2025	1.0	Initial Document

Contents

1 Symbols, Abbreviations, and Acronyms **iv**

2 General Information **1**
- 2.1 Summary 1
- 2.2 Objectives 1
- 2.3 Challenge Level and Extras 2
- 2.4 Relevant Documentation 2

3 Plan **3**
- 3.1 Verification and Validation Team 3
- 3.2 SRS Verification 4
  - 3.2.1 SRS Verification Approaches 4
  - 3.2.2 SRS Verification Checklist 5
- 3.3 Design Verification 6
  - 3.3.1 Design Verification Techniques 6
  - 3.3.2 Design Verification Checklist 7
- 3.4 Verification and Validation Plan Verification 8
  - 3.4.1 Verification and Validation Plan Verification Approaches 8
  - 3.4.2 Verification and Validation Plan Verification Checklist 9
- 3.5 Implementation Verification 10
- 3.6 Automated Testing and Verification Tools 11
- 3.7 Software Validation 12

4 System Tests **12**
- 4.1 Tests for Functional Requirements 13
  - 4.1.1 Data Filtering and Browsing 13
  - 4.1.2 Behavioral Analysis and Visualization 15
  - 4.1.3 Data Downloading and Accessibility 16
- 4.2 Tests for Nonfunctional Requirements 17
  - 4.2.1 Look and Feel Requirements 17
  - 4.2.2 Usability and Humanity Requirements 19
  - 4.2.3 Performance and Reliability Requirements 21
  - 4.2.4 Maintainability and Support Requirements 23
  - 4.2.5 Security and Compliance Requirements 24
- 4.3 Traceability Between Test Cases and Requirements 25

5 Unit Test Description **27**
5.1 Unit Testing Scope 27
5.2 Tests for Functional Requirements 27
5.2.1 Module 1 27
5.2.2 Module 2 28
5.3 Tests for Nonfunctional Requirements 28
5.3.1 Module ? 29
5.3.2 Module ? 29
5.4 Traceability Between Test Cases and Modules 29

6 Appendix **30**
6.1 Symbolic Parameters 30
6.1.1 Performance, Robustness, and Scalability Requirements 30
6.2 Usability Survey Questions? 33

1 Symbols, Abbreviations, and Acronyms

Table 1: List of Symbols, Abbreviations, and Acronyms

Symbol / Acronym	**Description**
T	Test
SRS	System Requirements Specification
V&V	Verification and Validation
FR	Functional Requirement
NFR	Nonfunctional Requirement
MIS	Module Interface Specification
MG	Module Guide
CI	Continuous Integration
UI	User Interface
FRDR	Federated Research Data Repository
API	Application Programming Interface
PR	Pull Request
ID	Identification
NLP	Natural Language Processing
SQL	Structured Query Language
CSV	Comma-Separated Values
GTmetrix	Grand Touring Metrix (web performance analysis tool)
APP	Application
EOU	Ease of Use
LEA	Learning
UAP	Understandability and Politeness
ACC	Accessibility
ARIA	Accessible Rich Internet Applications
SAL	Speed and Latency
ROFT	Robustness or Fault Tolerance

Continued on next page

Symbol / Acronym	**Description (continued)**
CAP	Capacity
SOE	Scalability or Extensibility
MAI	Maintainability
SUP	Supportability
INT	Integrity
IMM	Immunity
OWASP ZAP	Open Web Application Security Project Zed Attack Proxy
LEG	Legal
STA	Standards Compliance
ST-FR	System Test – Functional Requirement
ST-NFR	System Test – Nonfunctional Requirement
LON	Longevity
EPE	Expected Physical Environment
WE	Wider Environment
IWAS	Interfacing with Adjacent Systems
PROD	Productization
REI	Release
ADA	Adaptability

This section defines the symbols, abbreviations, and acronyms used throughout the plan. The purpose of this section is to make sure clarity and consistency remains when referring the plan.

2 General Information

2.1 Summary

The Gouda Engineers team built this entire software system. Their core aim? To deliver a platform—one that's both robust and lightning-quick—that doesn't just meet but exceeds every single functional and non-functional requirement laid out in the official SRS.

This document looks into the system's VnV process. This process is essential as it guarantees the software behaves exactly how it's supposed to and, crucially, completely satisfies what the stakeholders want. We will be checking that every single part of the development puzzle—the initial SRS, the intricate design blueprints, the lines of code itself—perfectly aligns with its own set of specifications and validation, is the final check.

Our rigorous VnV approach pulls together a variety of methods: automated testing provides speed, while painstaking manual inspections and collaborative peer reviews inject human judgment to evaluate the system's correctness, performance, and general usability. These activities are what fundamentally build confidence in the software's ultimate quality and unwavering reliability, simultaneously rooting out any lingering risks or pesky issues long before the final, official release.

2.2 Objectives

The main objective of this plan is to build confidence in the accuracy, reliability and stability of the software we are developing. The system works as a connection between researchers and the data set of rat trials. Our focus is on providing a program that retrieves correct data and works smoothy with no errors. AN objective is that the layout and information presented is usable and doesn't provide any friction in it use. However, we are not able to do user testing on a wide population and are beyond the scope of the project. WE assume that the underlying database and external libraries are already validated.

The plan wants to ensure that the core functions of the interface like data access, filtering and display are accurate and reliable, long-term performance testing is beyond the scope of this phase.

2.3 Challenge Level and Extras

The project would be advanced because it uses many complex software engineering ideas and principles. Creating a visually appealing UI, integrating NLP, implementing the store front display while also ensuring that the performance is good. These features require to make both the desing and verification of the system very in depth to fit the project.

The project includes two extras, a performance report and a user manual. The performance report identifies ares of the code where the software can be optimized to make it faster and more effecient, and the user manual offers the user guidance through the program.

2.4 Relevant Documentation

- Perry, Nathan, Aidan Goodyer, Jeremy Orr, et al. "OCD-Rat-Infrastructure/Docs/Problemst at Main - OCD-Rats-Capstone/OCD-Rat-Infrastructure." GitHub. Accessed 27 Oct. 2025. https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/tree/main/docs/ProblemStatementAndGoals

- Perry, Nathan, Aidan Goodyer, Tim Pokanai, et al. "OCD-Rat-Infrastructure/Docs/SRS-Volere at Main - OCD-Rats-Capstone/OCD-Rat-Infrastructure." GitHub. Accessed 27 Oct. 2025. https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/tree/main/docs/SRS-Volere

- Perry, Nathan, Leo Vugert, Tim Pokanai, Jeremy Orr, et al. "OCD-Rat-Infrastructure/Docs/Hazardanalysis at Main - OCD-Rats-Capstone/OCD-Rat-Infrastructure." GitHub. Accessed 27 Oct. 2025. https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/tree/main/docs/HazardAnalysis

- Perry, Nathan, Tim Pokanai, Aidan Goodyer, Jeremy Orr, et al. "OCD-Rat-Infrastructure/Docs/Design at Main - OCD-Rats-Capstone/OCD-Rat-Infrastructure." GitHub. Accessed 27 Oct. 2025. https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/tree/main/docs/Design

- Perry, Nathan, Tim Pokanai, Jeremy Orr, Aidan Goodyer, et al. "OCD-Rat-Infrastructure/Docs/Developmentplan at Main - OCD-Rats-Capstone/OCD-Rat-Infrastructure." GitHub. Accessed 27 Oct. 2025. https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/tree/main/docs/DevelopmentPlan

?

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.

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

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?

3.4 Verification and Validation Plan Verification

3.4.1 Verification and Validation Plan Verification Approaches

- **Peer Reviewers:** Our first approach for our Verification and Validation Plan verification is reviewing the peer review issues created by our classmates. As mentioned in the earlier section, this is not the most rigorous or systematic but allows our peers to provide insight from an outside perspective and gives us the chance to evaluate these criticisms and determine if changes need to be made.
- **Structured Internal Team VnV Review:** Our second approach for Verification and Validation Plan Verification is to do a structured team review, identical to that mentioned in the SRS verification section. As a team, we will go over the checklist for the VnV plan and each member will be given the chance to critique the section, identify things that should be changed and personally evaluate sections they may have not written or had the chance to inspect closely.
- **Mutation Testing:** The third approach is the only strictly novel approach being used for verifying the Verification and Validation Plan. Mutation tests will be used to validate the unit tests once they are developed and as many system testing modules as are applicable (mutation testing may not be appropriate for some mutation). Each test that is subject to our mutation testing should satisfy the requirements denoted in the checklist written in the next subsection.

3.4.2 Verification and Validation Plan Verification Checklist

Mutation Testing Specific Checks:

- Do all relevant test modules have at least one mutation test associated with them?
- For all test modules, are all mutants caught by the test suite for that module?
- In the case that a mutant is not caught, can a convincing reason be developed for why this is okay? If not, new tests must be created.

General VnV Checklist:

- Are all the generated tests reasonably valuable and worth the time to implement?
- Are all the generated tests feasible to implement?
- Do the tests cover the main anticipated error points that the team can come up with for each module?
- Are there any glaring and potentially likely failure cases not covered by the test?
- Does each component of the system contain a test suite to an acceptable level of granularity?
- Does each test have reasonable and easily justifiable connection to a specific requirement(s)?
- Does the scope of our testing include all necessary components and are all out of scope components clearly justifiable? Can any team member identify a component that may need to be moved from in scope to out of scope or vice versa?
- Do our tests cover every applicable requirement from our SRS? Are there any requirements which can be reasonably tested within the scope of the VnV that are not being tested against?

3.5 Implementation Verification

The implementation verification will be used to guide the system's adherance to the specifications as defined in the SRS. Verification will be done using both static and dynamic methods. The team will employ several static tools including linters and code quality analyzers to enforce a high standard and detect vulnerabilities.

One major source of verification will be our unit test suite, which will be defined for both frontend and backend components of the system and integrated into the CI process. Test cases will target functional requirements as defined in the

Unit Testing:

Unit Test cases will target functional requirements as defined in the System Tests (see Section 3.7). These tests will be important drivers of system correctness, ensuring functionality works according to specification. Correctness of querying behaviour (**FR2**) is one example of a critical requirement to verify.

Performance Testing:

Performance testing will be used to validate several important benchmarks impacting usability including:

- Query Speed (**NFR8**)
- Fault Tolerance (**NFR9**),
- Scalability (**NFR10**).

Performance testing will utilize end-to-end methods to validate metrics like query speed, where the response time can be measured using an automated UI test.

Static Analysis:

Static analysis tools will be employed to maintain a consistent quality threshold throughout the development lifecycle, helping adhere to style guidelines and protecting against defects early on. Specific tools are defined in section 3.6.

Code Walkthroughs and Inspections

- **Peer Review:** Pull Requests will go through a mandatory peer review process prior to any PR being merged. A minimum of 2 reviewers will be required prior to merging.

Manual Reviews:

To verify many of the non-functional requirements, the team will employ manual reviews and user studies to ensure the sytem meets look, feel, and usability requirements. Walkthroughs will be useful for several of the following:

- NFR3 UI Familiarity
- NFR4 Ease of Use
- NFR5 Learning Curve Assessment

3.6 Automated Testing and Verification Tools

The following tools are to be used during the verification and validation process to facilitate automated testing and verification. Broadly, these tools are separated into frontend and backend categories:

Frontend:

- Jest: Facilitates frontend unit testing and coverage
- ESLint: A JavaScript/TypeScript linter with support for JSX
- Puppeteer: End to end testing and automated UI testing.

Backend:

- flake8: Style guide enforcement for Python backend
- PyTest: Unit testing and integration testing.

Continuous Integration:

We will use GitHub actions for CI purposes, which will be configured to run the complete test suite for the frontend and backend upon commit, and serve as a quality gate prior to PR merges. Jobs will also be configured to run static analysis and linting checks, as well as generate coverage reports.

3.7 Software Validation

This section outlines our plan for validating the system to ensure it meets the needs of our stakeholders and users. Our core goal is to confirm the system provides an accessible and non-technical interface for researchers to analyze the rat behavioural data. The project is not defined in terms of specific performance metrics, but instead by its utility as an exploratory platform for researchers. Therefore, our validation strategy focuses on researcher utility and data integrity.

The following methods will be used to engage our project stakeholders and elicit feedback throughout the project lifecycle.

- **Revision 0 Demonstration:** The revision 0 demonstratation will be an important validation milestone for our software requirements. We will present this revision of the system to our supervisors to demonstrate the core features of the system. This will allow us to validate that the implementation aligns with the client goals and needs effectively.
- **Iterative Supervisor Feedback:** The team will leverage our weekly scheduled meetings with Dr. Szechtman and Dr. Dvorkin-Gheva to elicit ongoing feedback during the project's development. These meetings will provide continuous validation and will help us ensure that the project is aligned with our stakeholder expectations.
- **User Testing:** Towards the end of the developnent cycle we will conduct user testing. The user will be asked to perform tasks like executing a natural language query, filtering the data, or producing a visualization. We will use this user feedback to validate that the system is ready for its intended users.

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 non-technical 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 2 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 2: 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
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

Continued on next page

Requirement ID	Associated Test Case(s)
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
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:

   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

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.

Test 7: Accessibility Audit

- **Related Requirement:** ACC.R.1
- **Test ID:** NFR7-Accessibility
- **Type:** Static, Manual (Accessibility Review)
- **Initial State:** Fully implemented UI with all interactive elements deployed.
- **Input/Condition:** Users and automated accessibility tools evaluate the system.
- **Output/Result:** WCAG 2.1 AA compliance met; all interactive elements accessible via keyboard; screen readers correctly announce content.
- **How test will be performed:** Use accessibility testing tools (e.g., WAVE, Axe) and manual inspection; verify color contrast, tab navigation, and ARIA attributes.

6.1.1 Performance, Robustness, and Scalability Requirements

Test 8: Query Performance Benchmark

- **Related Requirement:** SAL.R.1
- **Test ID:** NFR8-Performance
- **Type:** Dynamic, Automatic
- **Initial State:** System deployed with representative dataset.

- **Input/Condition:** Run standardized queries on backend.
- **Output/Result:** Average query response time < 2 seconds for typical queries; < 4 seconds for complex queries.
- **How test will be performed:** Use automated scripts and CI tools to record response times; compare against thresholds.

Test 9: Fault Tolerance

- **Related Requirement:** ROFT.R.1
- **Test ID:** NFR9-FaultTolerance
- **Type:** Dynamic, Automatic
- **Initial State:** System deployed with simulated failure scenarios.
- **Input/Condition:** Induce backend failure, network disruption, or unexpected user input.
- **Output/Result:** System continues operation with minimal disruption; errors logged and no data loss occurs.
- **How test will be performed:** Simulate failures in test environment; monitor system responses and error handling.

Test 10: Scalability Testing

- **Related Requirement:** SOE.R.1
- **Test ID:** NFR10-Scalability
- **Type:** Dynamic, Automatic
- **Initial State:** System deployed in staging with incremental load testing.
- **Input/Condition:** Gradually increase number of simultaneous users and dataset size.
- **Output/Result:** System maintains acceptable performance; response time remains within defined thresholds.

- **How test will be performed:** Use load testing tools (e.g., JMeter, Locust) to simulate increasing load; monitor performance metrics.

Test 11: Maintainability and Supportability

- **Related Requirement:** MAI.R.1, SUP.R.1
- **Test ID:** NFR11-Maintainability
- **Type:** Static, Manual (Code Review)
- **Initial State:** Source code in version control with documentation.
- **Input/Condition:** Review code, documentation, and deployment scripts.
- **Output/Result:** Code conforms to style guides; sufficient comments and documentation for maintenance; modular design observed.
- **How test will be performed:** Conduct peer code reviews and documentation walkthroughs.

Test 12: Compliance and Legal Checks

- **Related Requirement:** LEG.R.1, STA.R.1
- **Test ID:** NFR12-Compliance
- **Type:** Static, Manual
- **Initial State:** System fully deployed; policies and regulations reviewed.
- **Input/Condition:** Evaluate compliance with data protection, licensing, and applicable legal standards.
- **Output/Result:** System meets all regulatory and legal requirements; no unlicensed content used.
- **How test will be performed:** Conduct legal and standards compliance audit; document verification steps.

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)?

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?

    - What went well writing this deliverable was our groups collaboration and finishing in a timely mannaer for the most part. For example, this past week we had two midterms that took away from our time to complete the deliverable. However, we all managed our time and finished the VnV plan in time.

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

    - A pain point we experienced during the deliverable was Leo falling ill before submission and our midterms from the last week. We also have another midterm coming up Wednesday, this made us intredibly nervous for time management, but the group submitted on time by focusing and collaborating with each other. Some would take extra parts and make the process better for others.

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.

- **Static analysis Approaches:** A very basic approach to testing but often an effective, especially when there is a large team to review each other's code. It will be very useful to review the most relevant static analysis approaches both for this test plan and so that we can generally improve our ability to review each other's code. This can include tools like SonarQube or can just be manual static analysis.
- **JMeter or other Load Testing Tools:** We will need to test the performance of our web application to identify how it performs under increased loads. JMeter allows us to create HTTP requests and measure how the system responds. This will be especially useful in testing the system response under increased user load or a request with a large data result.
- **Selenium or other test automation approaches:** Selenium is a good example of an automated web application testing tool. This is a tool that we expect to use, given that we will be creating a web application for the front-end of our system. The use of Selenium or a similar tool allows us to create repeatable and autonomous web application inputs that can be used to effectively test our front-end interface.
- **PyTest or other unit testing approaches:** Although not yet composed, the VnV plan will eventually contain an extensive set of unit tests which need to be implemented in our system. Reviewing or learning about a relevant unit test implementation approach will be crucial in effectively implementing robust unit tests in our project.

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?

   - **All team Members - Static Analysis:** As mentioned above, this is an important skill for general code review as well as our testing plan. As a result, it is important for all team members to review this. Internet tutorials such as a SonarQube walkthrough and reviewing the SFWRENG 3S03 class material are both good approaches for this.

- **Nathan and Leo - JMeter:** Both these team members have experience with load testing and thus it would be most effective for them to refresh their skills in JMeter. We expect the testing implementation to be largely compartmentalized so it would be best for each team member to play into their already existing strengths. Good tools to gain knowledge on this topic would be an internet tutorial such as one from tutorialspoint. Additionally, Nathan has a Kubernetes test-bed with JMeter on his device from an old project which would be useful to refresh knowledge.
- **Jeremy - Selenium:** Jeremy has done reasonably extensive work with Selenium and thus it makes the most sense for him to focus on this aspect of the testing technologies. A refresher from the internet, again from tutorialspoint would be useful. Alternatively, Jeremy can review old projects from his time on Coop to refresh his skills.
- **Tim and Aidan - PyTest:** Much like the other technologies, these team members will be taking this knowledge area because it plays into their already existing strengths. Many online tutorials about this can be found. Again, tutorialspoint is a good option (they seem to have tutorials on everything). Additionally, PyTest was covered in our SWFRENG 3S03 class so that is another good option.