

Web Application for the Voiland Food Pantry

Project Testing and Acceptance Plan

FIZ, VCSS



11-FA25-SP26-F-WEB

Jodie Butterworth

Kaitlyn Cornish

Matthew Hill

Alex Langland

November 28, 2025

TABLE OF CONTENTS

I.	Introduction	3
I.1.	Project Overview	3
I.2.	Test Objectives and Schedule	3
I.3.	Scope	3
II.	Testing Strategy	3
II.1	Testing Approach	4
II.2	Testing Process	4
II.3	Test Flow The testing flow is organized as follows:	4
II.4	CI/CD Considerations	5
III.	Test Plans	5
III.1.	Unit Testing:	5
III.2.	Integration Testing	6
III.3.	System Testing	6
III.3.1.	Functional Testing	6
III.3.2.	Performance Testing	7
III.3.3.	User Acceptance Testing	7
IV.	Environment Requirements	8
IV.1.	Hardware Requirements	8
IV.2.	Software Requirements	8
IV.3.	Network and Database Configuration	8
IV.4.	Hardware Integration Setup	9
IV.5.	CI/CD and Build Execution	9
V.	Glossary	9
VI.	References	9
VII.	Appendix-A	10

I. Introduction

I.1. Project Overview

The software associated with this project includes a webpage consisting of a landing page, which leads to pages for student and staff/admin login and dashboards, inventory management, and volunteer hour tracking. Testing will be crucial for these major elements to ensure they function properly to keep the Voiland Food Pantry operating smoothly during its business hours.

I.2. Test Objectives and Schedule

The objectives of our testing plan all work towards ensuring this application functions as expected of our stakeholders and future users. For this purpose, we will employ several kinds of testing methods, focusing our attention on integration testing due to the interconnected nature of this application. Manual testing will be utilized, allowing our team to personally test application operation and edge cases to ensure regular users do not encounter unintended errors with the system. This will be important for verifying the project's expected performance and ultimately, the final product we will deliver to our clients.

For this, both hardware and software resources will be required, which will be outlined later in greater detail. These resources will assist our team in our testing approach, which will facilitate our planned schedule of testing activities.

These activities include major work activities such as developing our testing strategy, setting up a testing environment, creating test cases for the system, executing tests, and recording the results of such tests. With this, we plan on delivering a functional system to our clients by the end of this semester so that they may interact with the application and its current features. Along the way, major milestones will be reached, such as approving our testing strategy, completing our test cases with results to highlight any improvements that must be made, and receiving feedback from our clients about said results.

I.3. Scope

The purpose of this document is to outline our team's testing strategy, along with the approach we will take to employ it. This testing strategy will be specifically designed to validate the Voiland Food Pantry web application performance with both the FIZ and VCSS's requirements for the application in mind. Along with this, we will define both hardware and software requirements required for operation, as well as testing criteria that must be met before application deployment. Different types of testing, including unit, integration, functional, performance, and user acceptance, will be performed to guarantee this project's reliability in an active usage situation.

II. Testing Strategy

The overall testing approach for the Voiland Food Pantry web application focuses on manual, exploratory testing of all currently implemented system components immediately after development. Testing is guided by the functional and non-functional requirements defined in the System Requirements Specification. The team aims to ensure that core features—such as user login, inventory management, and device integration—function correctly in realistic usage scenarios.

II.1 Testing Approach

Manual testing is performed for each module in isolation first (unit-level checks), then in combination with related modules (integration), and finally across the entire system (system testing). Each test session involves executing typical user workflows to verify that the system behaves as expected and handles invalid inputs or errors gracefully. Key aspects currently tested include:

- **User Login:** Verifying card swipe and website login for customers and administrators, including error handling for unregistered users.
- **Inventory Management:** Adding, updating, and removing items via barcode scanning and manual entry, ensuring accurate inventory counts.
- **Device Integration:** Testing barcode scanner and card reader input to ensure seamless interaction with the web interface.
- **Error Handling:** Ensuring appropriate messages appear for invalid operations, such as unregistered users attempting login or scanning invalid inventory codes.

Modules planned for future implementation next semester, such as volunteer hour logging and administrative reporting, are acknowledged but are not yet included in testing. Once implemented, these features will undergo the same manual testing process to ensure accurate data tracking and report generation.

II.2 Testing Process

The manual testing process follows these steps:

1. Identify the component or feature to test based on the functional requirements.
2. Prepare test scenarios reflecting common user workflows and potential edge cases, including valid and invalid inputs.
3. Execute the test by performing actions through the UI or device input, observing system responses.
4. Record results, noting success, failure, or unexpected behavior.
5. Document issues for later review, including steps to reproduce, observed behavior, and potential cause.
6. Verify fixes after bugs are resolved, repeating the tests to ensure the system works as intended.

II.3 Test Flow

The testing flow is organized as follows:

1. **Unit Checks (Manual):** Each currently implemented function is exercised individually in the local development environment to confirm correct behavior.
2. **Integration Testing (Manual):** Related modules are tested together to verify correct interactions.
3. **System Testing (Manual):** End-to-end workflows are tested using realistic scenarios, ensuring the complete system meets requirements.
4. **User Acceptance Testing (Manual):** Key stakeholders perform guided use cases on currently implemented features to confirm readiness for operational use.

II.4 CI/CD Considerations

At this stage, automated testing and CI/CD are not employed. The team has opted for manual testing due to the small scale of the project, limited number of implemented features, and reliance on interactive device input (card reader and barcode scanner), which are challenging to simulate in automated tests. This approach ensures that the team can observe real-time behavior and address user interface or integration issues immediately.

III. Test Plans

The testing plan for the Voiland Food Pantry web application focuses on verifying the correctness and robustness of the currently implemented features, as well as establishing procedures for testing future modules once implemented. Testing is conducted manually, following structured steps for each type of testing.

III.1. Unit Testing:

We will follow traditional unit testing practices by identifying the smallest testable units within the application and testing them in isolation from the rest of the system.

Since our project currently relies heavily on manual testing, unit testing will be conducted by:

- Running individual endpoints, login verification and inventory CRUD operations, directly after implementation.
- Verifying that each endpoint returns the correct HTTP responses, error messages, and state changes.
- Testing edge cases, such as:
 - Invalid barcode scans
 - Nonexistent user IDs
 - Negative or zero inventory quantities
 - Duplicate item creation
- Observing how the backend behaves when the card reader or barcode scanner provides unexpected input.

Typical units under test include:

- **User Login Handler**
Tests verification of registered vs. unregistered users, admin authentication, and card swipe input.
- **Inventory Service Functions**
Tests adding, updating, removing, and querying inventory items individually.
- **Device Input Handlers**
Tests how raw card reader and barcode scanner data is parsed and validated.

III.2. Integration Testing

Integration testing focuses on verifying that multiple components work together correctly. After unit-level behavior is confirmed, we gradually combine related modules to ensure they interact without errors.

Integration testing for this project includes:

Component Groups Tested

- **Login + Inventory Access**
Ensures only authenticated users with correct roles can access administrative inventory pages.
- **Scanner Input + Inventory Update Logic**
Confirms that scanning a barcode directly updates the right item, quantity, and timestamp.
- **Card Reader + User Lookup Process**
Tests whether scanned card data is correctly translated into a valid user login session.

Testing Approach

Integration testing is entirely manual at this stage. The sequence typically follows:

1. **Select a pair of components to test together** (login + inventory editing).
2. **Set up realistic data** such as sample users and sample inventory items.
3. **Trigger the integrated workflow** using the UI, scanner, or card reader.
4. **Check for correct interactions**, including:
 - a. Database updates
 - b. Correct routing to protected pages
 - c. Proper role restrictions
 - d. UI consistency
5. **Add more components** (login → inventory → admin dashboard) once no new faults appear.

III.3. System Testing

III.3.1. Functional Testing

Functional system testing is currently performed manually by the development team. Each major requirement in the project specification is matched to a corresponding functional test scenario.

Current functional testing includes:

- Logging in with a customer card, admin account, or invalid credentials
- Scanning inventory items and verifying correct behavior

- Updating quantities using the UI
- Attempting restricted actions as a non-admin user
- Ensuring error messages appear when expected

Because volunteer tracking and reporting are not yet implemented, they will be added to functional testing during the next development cycle.

Failures are documented along with the steps to reproduce the issue, and the responsible developer applies fixes before retesting.

III.3.2. Performance Testing

Although extensive performance testing is not required for the current scope of the project, we still evaluate basic performance characteristics during system-level testing to ensure the application remains responsive under typical usage. This includes:

- Login response time
- Inventory update latency when scanning or editing items
- UI responsiveness under typical administrative workflows

Because the current system is small and uses local databases, performance testing is largely qualitative—based on developer observations during usage. As the application grows to include analytics, reporting, and increased inventory volume, more formal performance testing will be introduced.

III.3.3. User Acceptance Testing

UAT involves testing the application from the perspective of pantry administrators and volunteers to ensure the system matches their real-world needs. Our User Acceptance Testing plan includes:

1. Preparing the Test Environment

- A working deployment of the web application
- Sample users, inventory items, and test data
- Feedback forms or a shared document for collecting comments

2. Testing Activities

- Administrators will test login, inventory updates, and barcode scanning.
- Pantry workers will test scanning workflows as they would during check-in/out.
- Users will attempt invalid actions (scanning an unregistered card) to verify error handling.

3. Feedback and Revision

- The development team reviews all feedback.
- Necessary improvements are prioritized and assigned.
- A final verification round takes place after changes are implemented.

User acceptance testing ensures the system meets user expectations and is ready for real-world operational use. As more features are added next semester, additional user acceptance testing cycles will be conducted.

IV. Environment Requirements

Specify both the necessary and desired properties of the test environment. The specification should contain the physical characteristics of the facilities, including the hardware, communications and system software, the mode of usage (for example, stand-alone), and any other software or supplies needed to support the test. Identify special test tools needed.

IV.1. Hardware Requirements

- Developer laptops/workstations with 8–16GB RAM and 4+ CPU cores for running the Spring Boot backend, MySQL server, and IDE tools simultaneously.
- Optional testing servers (university-hosted or local VM) with 16GB RAM and expandable storage for database snapshots and deployment testing.

IV.2. Software Requirements

- **Operating Systems:** Windows, macOS, or Linux — all team members used local environments compatible with Java and MySQL.
- **Java Development Kit (JDK 17+):** Required for Spring Boot 3.x features, annotation processing, and application compilation.
- **Spring Boot 3.x Framework:** Used to build REST controllers, services, dependency injection, and authentication logic.
- **Maven Build System:** Manages dependencies, plugins, and build automation for consistent project setup.
- **Thymeleaf Template Engine:** Enables server-side rendering of HTML pages and form binding for the pantry UI.
- **MySQL Server 8.x:** Used as the primary RDBMS for persistent data storage (inventory, volunteers, cards, logs).
- **MySQL Workbench / CL:** Used for schema validation, querying, and debugging database issues.
- **IDE Tooling:** IntelliJ IDEA, Eclipse, or VS Code
All configured with Spring Boot, Java, and Maven extensions.

IV.3. Network and Database Configuration

- Local MySQL instance configured with a shared schema name.
- Database connection set in application.properties:

- Local backend accessible at:
<http://localhost:8080/>
- Common development pages:
 - /login
 - /cardReader
 - /inventory
 - /volunteer

IV.4. Hardware Integration Setup

- **Magstripe Card Reader:** Tested and verified to behave as a keyboard device. Automatically injects card number data into input fields.
- **Barcode Scanner:** Reads UPC codes directly into HTML form inputs without additional drivers or software.

These devices required no additional middleware, ensuring smooth integration with the web application's input forms.

IV.5. CI/CD and Build Execution

- **GitHub:** Used for collaboration, version control, and branching strategy (feature branches for each milestone).
- **Application start commands:**
 - IDE Run Configuration, or
 - Command line: mvn spring-boot:run
- **Standardized setup steps:**
 - Clone project repository
 - Configure MySQL
 - Update credentials
 - Run backend locally

V. Glossary

CI/CD: Continuous Integration/Continuous Delivery.

CRUD: Create, Read, Update, and Delete.

FIZ: Frank Innovation Zone.

IDE: Integrated Development Environment.

VCSS: Voiland College Student Success

WSU: Washington State University.

VI. References

No references were used in the origination of this document. This may change in future iterations, for which we will update this section accordingly.

VII. Appendix-A

Example Testing Strategy:

1. Identify the requirements to be tested. All test cases shall be derived using the current Software Requirements Specification.
2. Identify which particular test(s) will be used to test each module.
3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
4. Identify the expected results for each test.
5. Document the test case configuration, test data, and expected results.
6. Perform the test(s).
7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the revised Test Plan document.
8. Successful unit testing is required before the unit is eligible for component integration/system testing.
9. Unsuccessful testing requires a bug form to be generated. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.
10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.