Ram Logic | Web Application for Voiland Food Pantry

Project Report

SPONSORS
Washington State University
Frank Innovation Zone

PREPARED FOR
Maynard Siev

PREPARED BY
Cristobal Escobar
Alan Sun

I. PROJECT DESCRIPTION

I.1 Introduction

The Voiland Food Pantry & Wellness Center serves as a critical resource for Washington State University students, providing access to food, hygiene products, and wellness support. With increasing demand and a growing variety of inventory, maintaining accurate records of pantry usage, tracking inventory levels, and collecting meaningful data has become a key operational challenge. Manual data entry processes are prone to errors and can make it difficult to quickly generate reports, limiting the pantry's ability to analyze trends and make informed decisions regarding procurement and volunteer scheduling.

The Data Tracker project is intended to address these challenges by designing and implementing an integrated system that collects, stores, and visualizes pantry data in a reliable and efficient manner. The system will include a database that records client check-ins, item distribution, and volunteer activity. Hardware peripherals such as a barcode scanner or card reader will be integrated to streamline check-in and inventory management processes, reducing manual input and improving accuracy.

Accurate and timely data collection is vital for the pantry's mission. Usage statistics can inform decisions about stocking patterns, predict surges in demand (e.g., during midterms or finals), and provide justification for continued funding and expansion. By presenting data in a centralized, easy-to-use interface through a WordPress-based web application, the system aims to empower pantry staff and volunteers to monitor operations in real time and generate reports for stakeholders.

This project is positioned within the domain of information systems for community services, combining database design, hardware integration, and web development to improve service delivery. The database team, newly assembled for this project, is responsible for developing the system from the ground up, beginning with the creation of an Entity-Relationship diagram to define key data relationships. The resulting platform will not only support the day-to-day operation of the pantry but also contribute to its long-term sustainability and impact on student well-being.

I.2 Background and Related Work

The pantry's current system is based on paper lists and spreadsheets, as well as disjointed digital methods, to organize pantry inventory and volunteer sign-ups. These methods are susceptible to human error, not very scalable, and lack analytics for data-driven decision-making. Food pantry programs usually require dedicated software systems that handle either inventory management or volunteer coordination but integrated solutions that manage both functions comprehensively remain scarce. Our front-end system will be developed on WordPress because it supports established web technologies and allows us to build a customized solution that meets the pantry's requirements.

The system to be constructed will operate together with the pantry's newly acquired and function-tested hardware components comprising a card reader and barcode scanner. These devices will support both inventory tracking and client registration processes for the pantry.

I.3 Project Overview

The core Food Pantry Data Tracker project is underway, with key specifications defined and an initial Entity-Relationship diagram created for the database. The team has selected WordPress as the platform for the web application and confirmed that hardware peripherals (barcode scanner and card reader) are available and functional. Key development areas include the following:

1. Database implementation and integration with WordPres
2. Front-end interface design with client-focused default view and volunteer sign-in option
3. Hardware integration for automated check-in and inventory management
4. Real-time data visualization and reporting capabilities
5. User interface refinements for accessibility and ease of use
6. Development of documentation, tutorials, and user guidance

By completing these tasks, this project aims to improve the accuracy, efficiency, and reliability of pantry operations, enabling staff to make data-driven decisions and better serve the student community.

I.4 Client and Stakeholder Identification

The project sponsor is the Voiland Food Pantry & Wellness Center represented by Maynard Siev. A data management system consisting of both software and hardware components is needed by the pantry to manage food distribution operations alongside volunteer scheduling and client support functions.

The pantry is a part of the larger operation of nonprofit and community needs resource management. Food banks, resource pantries, wellness centers, and similar entities typically have similar needs to track their services, volunteers, and client interactions. By having a solution that meets the specified use cases of barcode scanning, role-based interfaces, and database-powered reporting, this system can be applied to larger or similar efforts within the community.

Pantry customers, volunteers, and administrators are stakeholders that depend on the resource availability and can help guide its use. The use cases of inventory management, volunteer availability, client-facing availability, and recent hardware interface should provide a platform that is scalable, reliable, and easy to use in a broader context of nonprofits and similar use cases.

II. TEAM MEMBERS & BIOS

Cristobal Escobar

Cristobal Escobar is a Computer Science student with interests in machine learning, full-stack development, and data-centric system design. In this project, Cristobal contributed to the design of the MySQL database schema, the creation of the entity–relationship model, and the implementation of backend logic for managing inventory, student data, and volunteer workflows. He also helped develop the Flask-based prototype used for testing database functionality and collaborated with the client to refine requirements related to data collection and reporting. His work supported the team's goal of establishing a reliable backend foundation for integration with the WordPress front-end in the next project phase.

Alan Sun

Alan Sun is a Computer Science student interested in backend development and system security. His role in the project focused on the technical implementation of the backend logic and system integration. Alan was responsible for developing the Flask web application prototype, implementing the secure user authentication and session management systems, and creating comprehensive SQL seeding scripts to facilitate testing. He also led the efforts in verifying the system's CRUD functionality and ensuring data integrity across the student and administrative interfaces.

III. REQUIREMENTS AND SPECIFICATIONS

III.1 Introduction

This section outlines the data specifications for the Voiland Food Pantry's backend database system which manages inventory levels along with student and volunteer information.
A local, lightweight Flask front-end has been developed in order to test database functionality and ensure proper data validation and constraint enforcement before being exposed to the partner team's WordPress-based website.
The database design should preserve data organization and accuracy while positioning it for easy future import into the website environment.
The resulting platform will include a database to record client check-ins and item distribution, with integration of hardware peripherals (a Cougar Card reader and barcode scanner) to streamline check-in and inventory management. Accurate data collection is vital, as usage statistics will inform decisions on stocking patterns, predict demand, and provide justification for continued funding and expansion of the pantry's mission. The web application will be accessible via a WordPress domain, with a centralized, easy-to-use interface for staff and volunteers.

III.2 System Requirements Specification

III.2.1 Use Cases

Student Check-in (Client):

Pre-condition	Application running, on the clientele-facing page.
Post-condition	Pantry usage recorded (StudentId, VisitDate, TotalItems).
Basic Path	User scans Cougar Card. **OR** User uses the backup digital form sign-in.
Related Requirements	Student must be in the Students table. Requires hardware integration.

Volunteer/Admin Sign-in:

Pre-condition	Application running, on the home page.

Post-condition	User directed to their respective role-based interface.
Basic Path	User selects a separate sign-in option for Volunteers/Admins.
Related Requirements	Separate data models must exist for different user types.

View Inventory:

Pre-condition	Application connected to the database.
Post-condition	Display of all items, including Name, Category, Quantity, and Expiration Date.
Basic Path	User navigates to the Inventory view.
Related Requirements	Must retrieve data from the Items table.

Add Inventory Item:

Pre-condition	Application connected to the database.
Post-condition	New item recorded in the Items table.
Basic Path	User navigates to Add Item page. Enters Name, Category, Quantity, Expiration Date, and optional Barcode. Clicks "ADD ITEM".
Related Requirements	All fields except Barcode are required.

Edit Inventory Item:

Pre-condition	Inventory item exists and is displayed.

Post-condition	Item details (Name, Category, Quantity, Exp Date) updated in the database.
Basic Path	User clicks 'Edit' for an item. Enters new parameters and clicks 'Save'.
Related Requirements	Must be able to retrieve the item by its ItemId.

Delete Inventory Item:

Pre-condition	Inventory item exists and is displayed.
Post-condition	Item removed from the Items table.
Basic Path	User clicks 'Delete' for an item. Confirms the deletion.
Related Requirements	User must have Admin/Volunteer privileges.

Manage Volunteer Shifts:

Pre-condition	Application running, connected to the database.
Post-condition	Shifts are recorded in the Shifts table.
Basic Path	Admin/Volunteer inputs a Shift Date, Start Time, and End Time.
Related Requirements	Must support time and date data types.

Generate Expiration Report:

Pre-condition	Application connected to the database.

Post-condition	A report or list of items nearing their ExpDate is displayed.
Basic Path	Admin/Staff requests the Expiration Report.
Related Requirements	Must support reporting functionality.

Record Items Taken:

Pre-condition	Student check-in/usage has been recorded (UsageId).
Post-condition	ItemsTaken table is populated, linking UsageId to ItemId and QuantityTaken.
Basic Path	Volunteer/Staff records the items and quantities distributed during the visit.
Related Requirements	Must deduct quantity from the Items table.

III.2.2 Functional Requirements

III.2.2.1 Inventory Item Management

Description	The system should store and manage information about all types of food pantry items, including name, category, quantity, expiration dates, and barcode.
Source	Requested from our client Maynard
Priority	0

III.2.2.2 Auto-Generated Identifiers

Description	Each table in our database should have a unique auto-increment primary key to maintain data integrity and simplify relationships between tables.

Source	Suggested by our team and approved by our client.
Priority	0

III.2.2.3 Student Visit Logging

Description	The system should record the student's food pantry visits, it should store student ID, name, major and total numbers of items taken each visit
Source	Requested by our client Maynard.
Priority	0

III.2.2.4 Volunteers Hours Tracking

Description	The database should allow Volunteers and Shifts tables to be linked through the VolunteerHours and record the total hours worked by a volunteer with the corresponding shift.
Source	Requested by our client Maynard.
Priority	1

III.2.2.5 Referential Integrity and Validation

Description	The system should enforce data consistency using primary and foreign key constraints. Input validation needs to ensure proper data types.
Source	Suggested by our team and approved by our client.
Priority	0

III.2.2.6 Data Export for Website Integration

Description	The database should support exporting pantry data (items, students, volunteers, visits) into a MySQL- or CSV-compatible format for use by the WordPress website team.
Source	Suggested by our team and approved by our client.
Priority	0

III.2.2.7 Flask Testing Interface

Description	A simple Flask web application should connect to the local MySQL database to perform CRUD operations for testing and demonstration purposes before integration with WordPress.
Source	Suggested by our instructor and approved by our client.
Priority	0

III.2.2 Non-Functional Requirements

Compatibility

- The database must be available and integrated with the front-end application being developed on a WordPress domain.
- The system must integrate with the client's existing, function-tested hardware (Cougar Card reader and barcode scanner).

Usability

- The default state of the application must be the clientele-facing page.
- The design must be a user-friendly system that motivates students to sign in and supports simple data entry.

Performance

- The simplicity of implementation should be a top concern, factoring in budget constraints.

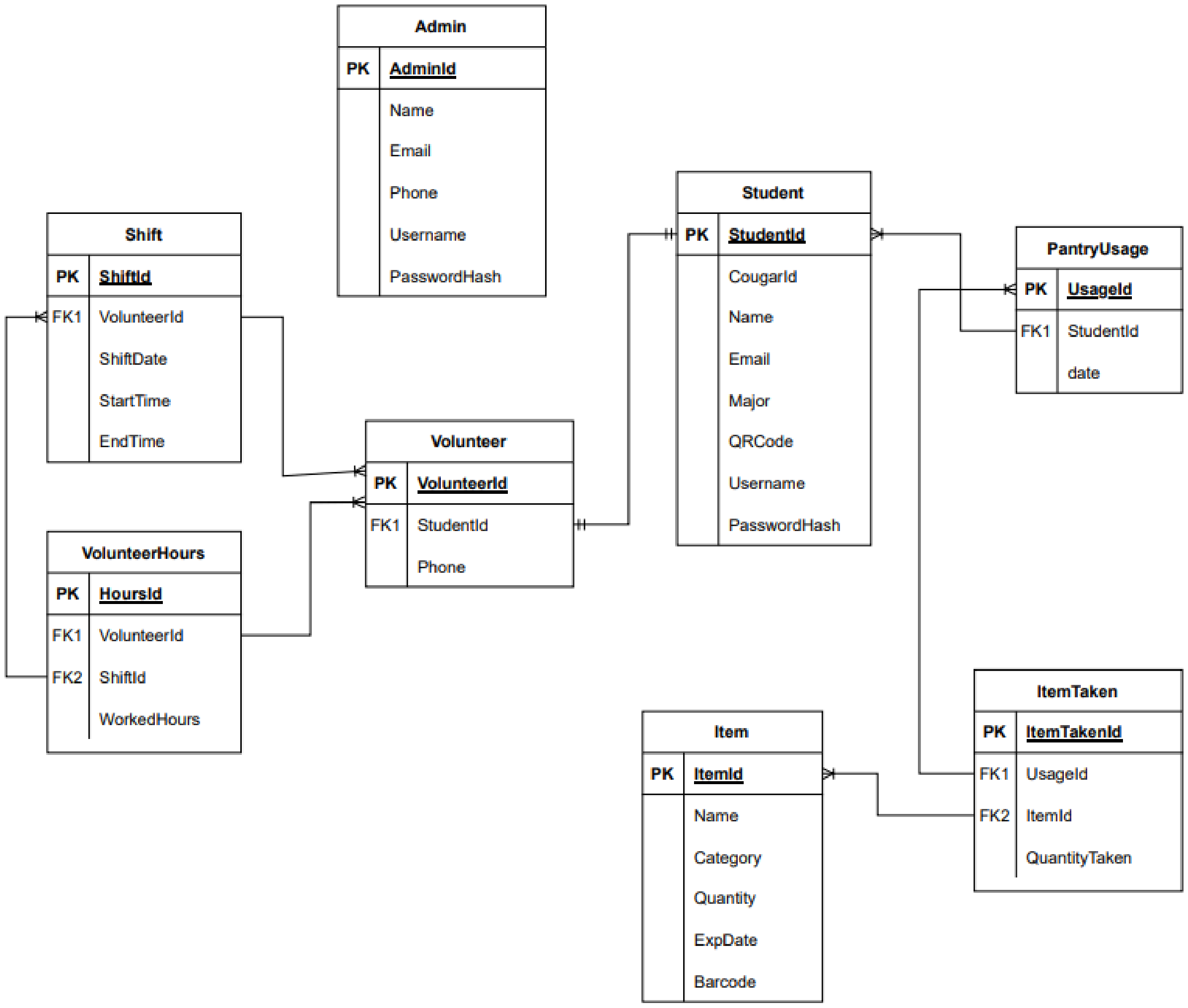
Scalability

- The database must be flexible and easy to expand to multiple departments if desired.
- The system should be applicable to larger or similar efforts within the context of non-profits and community needs resource management.

Maintainability

- The system must be able to support a workflow for syncing progress with the website team.

III.2.3 Entity and Relationship Requirements



The Entity–Relationship (ER) diagram defines the logical structure of the Voiland Food Pantry database and serves as the foundation for the implemented schema.The core entities implemented were Items, Students, Volunteers, PantryUsage, Shifts, and VolunteerHours. The relationships between these entities were added to allow for functionality for all of the major system operations. These entities were chosen because they correspond to physical items and operations discussed and identified in the meeting with the client. Additional entities can be easily added to the database as it is developed and tested further, such as tables for tracking donors and user authentication, for example. The finalized ERD will likely continue to be modified and extended as future testing and front end integration is done with WordPress and other components.

III.3 System Evolution

This section documents the current assumptions and identified risks that will influence the project's development path.

Assumptions

- **Hardware Functionality:** The client-provided hardware (Cougar Card reader and barcode scanner) is functional, tested, and will be IT-approved shortly for integration.
- **Platform Integration:** The database solution can be successfully integrated and made available to the website team's **WordPress** front-end application.
- **Scope Division:** The division of work between the database team (responsible for the back-end) and the website team (responsible for the front-end) is clearly established.
- **Entity Relationships:** The final Entity-Relationship (ER) diagram accurately captures the required data relationships (inventory, usage, volunteer hours, and shift scheduling).

Anticipated Changes and Risks

Risk/Unfinished Work	Implication/Mitigation	Source
Database Integration	Finalizing the ER diagram, beginning SQL schema creation, and defining data exchange protocols are required to integrate with the 421 (website) team.	Incomplete Issues
Coordination	Improving coordination and establishing a workflow with the website team is necessary for smooth integration.	Needs Improvement
Data Specification	The team still needs detailed specifications	Needs Improvement

	from the client on exact reporting requirements and data fields for usage, inventory, and volunteer tracking.	
Access Control	Precise access permissions and restrictions for each user type need to be defined and documented.	Needs Improvement
Expiry Alerts	Logic for generating reports on soon-to-expire items needs to be implemented (e.g., automated expiry alerts or reporting scripts).	Needs Improvement

IV. SOLUTION APPROACH

IV.1 Introduction

This section describes the technical solution and system design for the Food Pantry Database Project. The components include:

System Architecture: Summary of the solution's overall structure, integration points, and process flow between the database and the end-user facing application.

Data Design: The database schema, relationships, and planned future growth.

User Interface Design: Initial views created with Flask to prototype database functionality, and in preparation for integration with the final website front-end.

This document will help developers as well as the client understand the:

- Database schema and relations.
- Permissions and user-facing logic.

- Integration with barcode/QR scanner, to be developed.
- Foundation for automated reports and web deployment.

The specifications in project documentation guide these decisions to ensure system compatibility with client requirements and support subsequent implementation phases by the website Team.

IV.2 System Overview

The main goal is to create the Voiland Food Pantry Tracker, a system to facilitate pantry operations, specifically inventory and usage management, student tracking, volunteer scheduling, and basic reporting. The system has two logically distinct, cooperating parts:

- Database Subsystem (Our Team)
    - A MySQL database to house inventory, student visitation, volunteer hours, user information, and supporting metadata.
    - A Flask-based "test" application that allows admins to directly CRUD tables and test functionality in a sandboxed environment before connection to the live website.
    - Ability to support barcode/QR scanning for adding/updating inventory items.
- Website User Interface
    - A WordPress (or custom web?) front-end for pantry users and staff.
    - Handles the task of displaying an end-user friendly interface where pantry visitors, volunteers, and others can check in, view inventory, or log activities.
    - Connects to the MySQL backend via common import/export data formats to keep information in sync.

The database subsystem serves as the application foundation where information is kept in a permanent structure that supports reporting. The project setup has been constructed to ensure adaptability and extensibility while being prepared to handle potential future requirements such as donor tracking and item-level imagery.

The workflow that the system is expected to support:

- Guest users looking at available pantry items.
- Students signing up and applying to be volunteers.
- Admin logging in and performing management of students, volunteers, shifts, and inventory.
- Automated creation of monthly usage and inventory reports.

The split design means that each team should be able to independently develop and test the parts, and then interface requirements can be designed later in the project timeline.

IV.3 Architecture Design

IV.3.1 Overview

The Voiland Food Pantry Tracker project will be based on client-server architecture, where the data layer (database/backend) and the interface layer (webapp/frontend) will be separate. We will design the project to be scalable and modular, where each of the database/frontend components can be developed by each team in isolation and then fit together.

Project will include the following architectural components:

Backend (Database Layer)

- MySQL database: the persistent datastore for students, volunteers, inventory, shifts, administrators, and pantry use.
- Flask app: light weight API, and a development/testing interface, which can implement CRUD database operations. The database team can use this interface to independently test data logic before it is integrated with the live web app.

Frontend (Web Application Layer)

- Prototype with vanilla JavaScript and local server environment will be implemented by the website team.
-
- Migration to WordPress, or other web-hosting environment, to be determined by the final hosting platform.
- Interact with backend via SQL export/import or API calls

Hardware Integration:

- Barcode scanner, and Cougar Card reader will be used to automatically input items, or check-in students.
- Devices function as input peripherals, which feed data into the frontend interface, which then pushes data into the backend for persistence.

Resulting architecture will ensure that:

- Database is a single source of truth, and is independent of interface technology.
- Frontend can change independently, and consume the backend via data sync protocols.

IV.3.2 Subsystem Decomposition

The following sections describe the primary subsystems of Food Pantry Tracker and their functions in the architecture. Each subsystem offers a particular set of capabilities that ensures that the system runs efficiently and in a modular fashion.

IV.3.2.1 User Management Subsystem

Description:

Manages all user-related data and permissions including students, volunteers, and admins.

Subsystem Responsibilities:

- Register and authenticate users (students and admins).
- Support future volunteer application and approval workflows.
- Enforce role-based access in the application such as guest, student, volunteer or admin.

Concepts and Algorithms Generated:

- Session-based user authentication using Flask sessions.
- Password hashing for admin and student credentials.
- Future integration with WordPress authentication.

Interface Description:

Services Provided:

Service Name	Service Provided To	Service Description
Login	UI	Validates user and creates session.
Logout	UI	Clears session and logs out user.
Register	UI	Adds student/admin user to

		database.

Services Required:

Service Name	Service Provided From
Get user credentials	Database

IV.3.2.2 Inventory Management Subsystem

Description:

CRUD operations to pantry inventory. Store item information upon barcode scan while monitoring weight and expiration date.

Subsystem Responsibilities:

- Add, update, delete, view items in the pantry.
- Store information Name, Category, Quantity, Weight, Expiration Date, UPC, optional Comments.
- Interact with barcode scanners to lookup or add items.

Concepts and Algorithms Generated:

- CRUD implementation (Flask + MySQL)
- Field validation (required name, positive qty, valid expiration date, etc.)
- UPC lookup flow: if exists increment quantity, otherwise prompt to add details.

Interface Description:

Services Provided:

Service Name	Service Provided To	Service Description
Get all items	UI	Displays list of all inventory items.
Add item	Admin UI	Insert new item into database.
Edit item	Admin UI	Update item info into

		database.
Delete item	Admin UI	Removes item from database.

Services Required:

Service Name	Service Provided From
Get database connection	Database

IV.3.2.3 Volunteer Management Subsystem

Subsystem Responsibilities:

Associates volunteer work with students, and records their sign up and approvals.

Subsystem Responsibilities:

- Relate students to volunteers.
- Enable admins to approve volunteers.
- Hold volunteer status for authorization.

Concepts and Algorithms Generated:

- JOINs of Students and Volunteers in SQL.
- Status of volunteer applications

Interface Description:

Services Provided:

Service Name	Service Provided To	Service Description
Get volunteers	Admin UI	Show the list of volunteers
Approve volunteer	Admin UI	Update volunteer status as approved.
Denial volunteer	Admin UI	Update volunteer status as denied.
Apply as volunteer	Student UI	Create volunteer

		application.

Services Required:

Service Name	Service Provided From
Get student/volunteer info	Database
Get database connection	Database

IV.3.2.4 Shift Scheduling Subsystem

Description:

Admin volunteers are assigned to pantry shifts. Admin records hours for a monthly report.

Subsystem Responsibilities:

- Keep track of shift date and time.
- Associate shifts to volunteer hours.

Concepts and Algorithms Generated:

- SQL CRUD with time and date fields.
- Calculate time difference.

Interface Description:

The Results Interpreter Subsystem integrates the result non-proprietary CAD file with a respective PCB non-proprietary CAD file.

Services Provided:

Service Name	Service Provided To	Service Description
Get all shifts	Admin UI	Display list of shifts.
Add shift	Admin UI	Create a new shift.
Edit Shift	Admin UI	Update a shift.
Delete Shift	Admin UI	Remove shift from database.

Services Required:

Service Name	Service Provided From
Get database connection	Database

IV.3.2.5 Reporting and Exporting Subsystem

Description:

Automatically gathers and exports use reports for assisting with grant applications and budgeting decisions.

Subsystem Responsibilities:

- Gather information about students using the pantry.
- Output reports in CSV/Excel format.
- Schedule automatic monthly reports.

Concepts and Algorithms Generated:

- SQL GROUP BY and joins to build reports.
- Python scheduling libraries.
- Flask route for manual, on-demand exports.

Interface Description:

Services Provided:

Service Name	Service Provided To	Service Description
Generate monthly report	Admin UI	Create CSV with monthly reports.
Export data	Admin UI	Export students, items, etc as CSV files.

Services Required:

Service Name	Service Provided From
Get usage data	Database

Get inventory data	Database

IV.4 Data design

The proposed data model for the Voiland Food Pantry system is a relational schema implemented in MySQL. The schema was built with data consistency, referential integrity, and scalability in mind to account for the intended integration with both a Flask test front end, and the eventual website interface. The database entities and relationships were previously outlined in the ER diagram included in the Requirements Analysis portion of this report.

This data model accounts for multiple types of users (guests, students, volunteers, and admins), but normalizes tables appropriately in order to prevent duplication of information.

Schema Details:

- Students: User personal information is stored here, as well as whether they are a pantry user.
- Volunteers: Tracks whether a student applied or is approved to volunteer, linked directly to a student's ID to prevent duplication.
- Admins: Stores administrative user credentials which can access the system with full access and management privileges.
- Items: Stores pantry item information such as category, quantity, expiration date, and weight, with an optional comment field. A UPC system allows for scanning of items.
- Shifts: Tracks the work shifts volunteers take as well as allows the administrator to assign and modify shifts.
- Visits and ItemsTaken: Tracks which students visit the pantry and what items they take to later allow for automated monthly reports.

Foreign key constraints are used to ensure referential integrity, and timestamp fields were included to allow for future reporting and auditing capabilities

The Food Pantry Tracker data layer was designed with a MySQL relational database along with the Flask-based prototype front end in order to facilitate core pantry workflows, as well as provide the development team a means to verify functionality prior to delivery with a final web front end.

Back End Database

MySQL Schema core tables include:

- Students: basic student information; Cougar ID, name, email, major. Primary pantry users.
- Volunteers: student volunteers; active after approval. Foreign key to Student Table.
- Admins: users who can perform all functions of the system (add inventory, approve volunteers, etc).
- Items: pantry inventory; name, category, weight, count, expiration date, UPC, comments.
- Shifts: volunteer shift sign-up; date and time window.
- Pantry Visits & Items Taken: will be used to track item-level use of the pantry and generate use reports/logs

Note: all tables are constructed with relational design patterns such as:

- Auto-incrementing primary key
- Foreign keys
- Required field constraints
- Standard SQL types (VARCHAR, DATETIME, INT, DECIMAL, etc.)

Note: Entity-Relationship Diagram for this data layer is available in the Requirements tab.

Frontend Testing Layer (Flask App)

A basic Flask front end was constructed for the purposes of:

- Testing connectivity to DB & performing validation on CRUD operations
- Prototyping login and session flow
- Validating field-level constraints (expiration date, required, etc.) on form submissions
- Mock-up of role-based capabilities (guest, admin)
- Aiding in demo prep & DB troubleshooting

This app is intended to serve as a local testing sandbox to confirm that various workflows work prior to integration and data ingestion/exchange with the website team's external app, likely in JavaScript or WordPress based on the final choice.

IV.5 User Interface Design

The Food Pantry System will have a user interface (UI) in order to be useful for testing and eventual implementation. The database team developed a temporary Flask-based UI which enables them to test primary system functions including user access permissions and basic CRUD mechanisms before the website team starts building the final front-end interface. The current prototype interface is not intended to be the final product. It will, however, be used to validate database queries and behavior and will be useful for agile development with the client.

The final UI to be used by users will be developed by the website team and is expected to be based around scanning work flows for the majority of inventory and client interactions. The UI will include student checking, volunteer contributions and reporting/admin work as a simple, friendly user experience with appropriate frontend styling.

Two phases of the user interface (UI) for Food Pantry will be implemented:

1. Temporary UI – Flask Prototype (Local Testing Environment)

The UI to be used during development will be a lightweight Flask-based interface for testing and validating the database components. While this is not the final system, it should allow for basic CRUD operations and support:

- CRUD operations for:
    - Inventory items
    - Volunteers
    - Shifts
- Form-level validation (required values, formats, etc.)
- Manual data entry / admin-based database changes
- Session-based login for admin access and guest browsing
- Implemented Templates (Python Flask)
- home.html – Landing page; content varies for admin vs. guest
- inventory.html – List of items; admin can add/edit/delete
- add_item.html, edit_item.html – Item forms for admin only
- volunteers.html – View / manage volunteer applications / status
- shifts.html – CRUD support for volunteer shift scheduling
- login.html – Simple user authentication (student vs. admin modes)

This testing interface will be provided to the database team for them to have a fully functioning environment in which they can validate their schema and database logic. It will also be used to “prepare the plumbing” for data access when the website team develops the final UI.

2. Final UI – To Be Implemented by Website Team (CPTS 421)

The final UI that students will interact with is to be built by the partner website team for CPTS 421. This will be scanner-driven first and foremost, so the priority features are as follows:

- Barcode / QR scanner integration for rapid inventory updates
- User-facing checkout process; students scan Cougar Card to record pantry usage
- Volunteer portal for clocking in/out and managing shifts
- Admin dashboard with inventory, approve volunteers, and reporting
- Automatic reporting (e.g. monthly summaries of pantry usage / inventory flow)

The database team will support this effort by creating either a REST API for data access or some form of export (CSV, MySQL dumps) that allows for data to be imported directly.

Platform update: Although WordPress was planned for the final UI, the final hosting platform (local vs. hosted) is still being determined.

V. TESTING AND ACCEPTANCE PLANS

V.1 Introduction

V.1.1 Project Overview

The Voiland Food Pantry Web Application aims to provide a robust backend system for managing pantry operations. The core functions of this project include:

- Managing inventory (tracking items, categories, quantities, and expiration).
- Tracking student pantry usage (logging visits and items taken).
- Managing volunteer information and scheduling shifts.
- Providing role-based access for students, volunteers, and administrators.
- Enabling data export for reporting and analysis.

The aim of our test process is to ensure the database schema is logically sound, all functional requirements are met, and the system's data integrity is absolute before integration with the final WordPress front-end.

V.1.2 Test Objectives and Schedule

The primary objective of our testing plan is to verify the correctness and reliability of the database backend. We will generate test coverage for the following:

- Database Schema Testing: Low-level testing to verify table structures, data types, constraints (NOT NULL, UNIQUE), and foreign key relationships (especially ON DELETE CASCADE).
- Unit Testing (Backend Logic): Testing individual functions and application routes (e.g., adding an item, editing a student) to ensure they perform the correct SQL operation and handle data as expected.
- Integration Testing (Flask UI): Testing the complete workflow of a user action, from submitting a form in the Flask prototype to verifying the data is correctly and persistently stored in the MySQL database.
- System & Acceptance Testing: High-level testing to confirm the system meets all client requirements and that the data is ready for consumption by the partner website team.

Our team adopted a "prototype-driven" testing approach. A temporary Flask front-end was developed concurrently with the database specifically to act as a test harness.

V.1.2.1 Required Testing Software Resources

To facilitate end-to-end testing of the backend, our team utilized the following:

- MySQL Server (8.0+): To host the food_db database.
- Python 3.x: As the runtime for the testing application.
- Flask Framework: To build the temporary web interface (app.py) for interacting with the database.
- mysql-connector-python: To handle the connection between the Flask application and the MySQL database.
- Standard Web Browser: To access the Flask prototype and perform manual tests.

V.1.2.2 Milestones and Deliverables

With software requirements and our high-level plan, the deliverables for testing are as follows:

- A validated schema.sql file.
- A populated seed_data.sql file to create a consistent test environment.
- A functional Flask testing application (app.py and associated templates/) capable of performing all required CRUD operations.
- A final test report (this document) and a successful client demonstration.

V.1.3 Scope

This section provides our team's test procedure, which is designed to ensure database quality, detect bugs in logic and data integrity, and provide a clear validation that the project requirements from our client, Maynard Siev, have been met.

V.2 Testing Strategy

The testing strategy is centered on our purpose-built Flask web application (app.py), which functions as a local, interactive testing interface. This approach allows the database team (Ram Logic) to perform comprehensive, end-to-end testing of the backend logic independently of the partner website team's (CPTS 421) front-end development.

This strategy allows us to isolate and confirm backend functionality, ensuring that any bugs that arise during final integration are not related to the database's core logic.

The team generated tests for the following core components (as defined in schema.sql):

- Inventory Management (Items table): All CRUD operations (Create, Read, Update, Delete) for pantry items.
- User Management (Admins, Students, Volunteers tables): Registration, login, and validation of different user roles. Verifying that a Volunteer must first be a Student.
- Shift Management (Shifts, VolunteerHours tables): Creation and deletion of shifts and the association of volunteers to those shifts.
- Pantry Usage Logging (PantryUsage, ItemsTaken tables): Simulating a student check-in, recording the items they take, and verifying that this data is correctly linked.
- Hardware Integration (Simulation): Using the Flask app's forms to simulate hardware input (e.g., manually typing a CougarId to simulate a card swipe).

V.3 Test Plans

V.3.1 Database Schema & Unit Testing

Unit tests were performed by validating the schema.sql and seed_data.sql files.

Constraint Testing: We confirmed that the database correctly rejects invalid data.

- Test: Attempt to add an Item with NULL for Name.
- Expected Result: The database throws a NOT NULL constraint violation.
- Test: Attempt to add a Student with a Username that already exists.

- Expected Result: The database throws a UNIQUE constraint violation.

Foreign Key Testing: We confirmed that relationships between tables are enforced.

- Test: Attempt to add a Volunteer record for a StudentId that does not exist in the Students table.
- Expected Result: The database throws a foreign key constraint violation.

Cascade Delete Testing: We verified that deleting a parent record correctly cascades to child records.

- Test: Delete a Student from the Students table who is also a Volunteer.
- Expected Result: The corresponding record in the Volunteers table is automatically deleted.

V.3.2 Integration Testing (Flask Prototype)

Integration testing was performed manually using the Flask web application to test the full path from user input to database persistence.

Test Case: Add a New Inventory Item.

1. Log in as "admin" / "password" on the /login page.
2. Navigate to the /inventory page.
3. Click "Add Item".
4. Fill out the add_item.html form with new item details (e.g., Name: "Test Cereal", Quantity: 50).
5. Submit the form.
6. Verify: The user is redirected to the /inventory page and "Test Cereal" is now visible in the list.
7. Verify (DB): Check the food_db.Items table directly in MySQL to confirm the new row exists with the correct data.

Test Case: Edit a Shift.

1. Log in as "admin".
2. Navigate to the /shifts page.
3. Click "Edit" on an existing shift.
4. Change the StartTime from "09:00" to "10:00" on the edit_shift.html form.
5. Submit the form.
6. Verify: The user is redirected to the /shifts page and the updated "10:00" time is displayed.

V.3.3 System Testing

V.3.3.1 Functional Testing

System-level functional testing involved executing every Use Case specified in Section III.2.1 using the Flask prototype. This confirmed that all required business logic was implemented and functioning correctly.

V.3.3.2 Hardware Simulation Testing

The Flask application's forms were used to simulate the input from the hardware peripherals.

- Cougar Card Reader Test: The "Student Check-in" use case was tested by manually typing a valid CougarId (from the Students table) into a dedicated form. The system successfully identified the student and logged their visit in the PantryUsage table.
- Barcode Scanner Test: The "Add Inventory Item" use case was tested by manually typing a UPC/barcode number into the "UPC" field of the add_item.html form. The system correctly stored this value, proving the data pipeline is ready for the real scanner.

V.3.3.3 User Acceptance Testing (UAT)

User Acceptance Testing was planned in two phases:

- Internal UAT (Completed): The database team and our client, Maynard Siev, reviewed the Flask prototype. The client confirmed that the database structure, data fields (e.g., Items.Weight, Items.Comment), and workflows (e.g., linking Students to Volunteers) successfully met the project's operational requirements.
- External UAT (Pending): The final acceptance test is the successful integration by the partner website team (CPTS 421). Our backend system is considered "accepted" when they can successfully connect their WordPress front-end to our MySQL database and perform all necessary data operations to power their user interface.

V.4 Environment Requirements

Our testing and development environment for the Voiland Food Pantry Data Tracker required specific hardware and software configurations to ensure proper functionality and replicability.

Physical Characteristics & Hardware:

- Server/Host Machine: We utilized standard computers (Windows, macOS, and Linux) capable of running a local MySQL server and Python runtime to host the application.
- Peripherals (Target Environment):
    - Barcode Scanner: A handheld USB or wireless barcode scanner is required for the rapid entry of item UPCs during inventory management.
    - Cougar Card Reader: A magnetic stripe or NFC reader compatible with Washington State University student IDs is necessary to simulate and perform client check-ins.

System Software & Communications:

- Operating System: The system is compatible with Windows 10/11, macOS 11+, or standard Linux distributions (e.g., Ubuntu 20.04).
- Database Engine: We used MySQL Server version 8.0 or newer for all data storage.
- Runtime Environment: Python 3.7 or newer is required to execute the backend logic.
- Network: Development was conducted on localhost; however, an internet connection is required for installing dependencies and accessing the remote repository.

Software Dependencies & Special Tools:

- Flask: The web framework we selected to serve the application interface.
- mysql-connector-python: The database driver we used to establish connectivity between our Python application and the MySQL database.
- Web Browser: A modern web browser (Google Chrome, Mozilla Firefox, or Safari) is required to interact with the user interface.
- Git: Used for version control and repository management throughout our development cycle.

V.3. Test Results

- Our prototype testing yielded the following results based on our "Test Case Specifications" and the current status of the project:
- Database Schema Validation: Our unit tests confirmed that the MySQL schema correctly enforces data integrity. Primary keys auto-increment as expected, and

foreign key constraints successfully prevent orphan records (e.g., we confirmed a Volunteer record cannot exist without a linked Student record).

- CRUD Functionality: Integration testing of our Flask prototype confirmed successful Create, Read, Update, and Delete operations for the Items, Students, and Admins tables. We verified that forms correctly validate required input fields before submission.
- User Authentication: We successfully tested the session-based authentication system. Users can register, log in, and log out without issues. Our access control logic correctly differentiates between 'Admin', 'Volunteer', and 'Student' roles, restricting access to sensitive pages effectively.
- Data Persistence: We verified that data entered via the web interface (e.g., adding a new inventory item like "Canned Beans") is correctly persisted in the MySQL database and remains available after a server restart.
- Hardware Simulation: We simulated hardware integration tests successfully using manual input. Entering a UPC into the "Barcode" field correctly associated the data with the item, verifying the system's readiness for physical scanner integration.

VI. Projects and Tools Used

Tool/Library/Framework	**Quick note on what it was for**
Flask	A micro web framework written in Python we used to build the front-end testing interface and handle HTTP requests.
MySQL	The relational database management system we used to store and manage all persistent data (inventory, users, shifts).
mysql-connector-python	A Python driver for communicating with the MySQL database, enabling us to execute SQL queries directly from the app.
Jinja2	The templating engine used by Flask to render dynamic HTML pages populated with our database content.
Git / GitHub	Version control system and remote repository for our source code management and team collaboration.
Visual Studio Code	The primary Integrated Development Environment (IDE) we used for writing and debugging code.
MySQL Workbench	Visual tool we used for database design, SQL development, and database administration.

Languages Used in Project

Language	**Usage**
Python	Core backend logic, route handling, and database interaction (app.py).
SQL	Database schema definition and data seeding .
HTML5	Structure of the user interface web pages (Templates).
CSS	Styling of the user interface (implied via static assets).

VII. Description of Final Prototype

Our final prototype is a functional web-based data tracking system designed to manage the core operations of the Voiland Food Pantry. It features a responsive user interface powered by a Python Flask backend and a MySQL database.

User Manual & Key Workflows:

1. System Setup:
   - Ensure MySQL is running and the food_db database is initialized using our provided schema.sql and seed_data.sql files.
   - Launch the application by running python -m flask run in the terminal.
   - Access the system via http://127.0.0.1:5000/.
2. User Authentication:
   - Login: Users navigate to the Login page. Administrators, Volunteers, and Students enter their username and password. Our system detects their role and redirects them to the appropriate dashboard.
   - Registration: New students can create an account via the Registration page. We created a dedicated registration route for Admins to ensure security.
3. Inventory Management (Admin/Volunteer):
   - View Inventory: The "Inventory" tab displays a table of all items, including Name, Category, Quantity, Weight, and Expiration Date.
   - Add Item: Admins can click "Add Item" to input new stock. This form supports manual entry or barcode scanning into the UPC field.
   - Edit/Delete: We enabled functionality to update existing items (e.g., changing quantity) or remove them entirely from the database via the respective actions in the inventory list.
4. Volunteer Management:

    - Admins can view a list of all volunteers, their approval status, and contact information. This allows for efficient management of the student workforce.

Prototype Visuals:

- Database Schema: Our backend is structured around relational entities including *Students*, *Items*, *Shifts*, and *PantryUsage* as visualized in our ER Diagram.
- User Interface: We designed the interface using clean, form-based layouts to facilitate easy data entry, which is critical for high-traffic pantry environments.

VIII. Social Responsibility and Broader Impacts

Social Responsibility: In developing this system, we made informed judgments concerning data privacy and accessibility. Although the current system is a prototype, we handled student data (names, IDs, emails) by structuring the database to minimize the exposure of sensitive personal information. We also implemented distinct user roles (Admin vs. Student) to ensure that sensitive administrative data is not accessible to general users. One challenge we faced was balancing ease of access with security; while we prioritized a frictionless login to encourage pantry usage, we recognize the ethical obligation to protect student anonymity in production environments.

Broader Impacts: Our work directly aligns with the goal of benefiting society by addressing food insecurity on university campuses. By digitizing the Voiland Food Pantry's operations, our system reduces administrative overhead, allowing staff to focus more on serving students rather than managing paperwork.

- Efficiency: Automated tracking ensures that popular items are restocked promptly, ensuring resources are available to those in need.
- Inclusivity: We designed the system to be simple and user-friendly, accommodating volunteers and students with varying levels of technical literacy.
- Scalability: By releasing this documentation and code, our project serves as an open-source template that other campus food pantries or non-profit resource centers can adapt, extending the impact beyond WSU.

IX. Product Delivery Status

We have demonstrated the project prototype to our client, Maynard Siev, and the partner website team. The codebase and documentation are hosted in a shared repository for future development and integration.

Project Location Information:

1. Source Repository: https://github.com/WSUCptSCapstone-S25-F25/-mda-unity3dapp- (Located in the Fiz Project directory).
2. Materials for Rebuild:
   - schema.sql: Located in Fiz Project/App/DB Files/ - Required to build the database structure.
   - seed_data.sql: Located in Fiz Project/App/DB Files/ - Required to populate the database with initial test data.
   - app.py: Located in Fiz Project/App/ - The main application entry point.
   - requirements.txt: Lists Python dependencies (flask, mysql-connector-python).

Installation Instructions: To install and set up the project, future users should:

1. Install Python 3 and MySQL Server 8.0.
2. Install dependencies: pip install flask mysql-connector-python.
3. Configure the database connection string in app.py to match local MySQL credentials.
4. Run the application: python -m flask run.

X. Conclusions and Future Work

X.1. Limitations and Recommendations

Limitations:

- Security: Our current prototype stores passwords as simple hashes or plain text in the seed_data, which is not suitable for a production environment.
- Hosting: The system currently runs on a local Flask server (localhost). It is not yet deployed to a live web server or integrated into the WordPress environment as originally targeted.
- Reporting: While data is collected, the generation of monthly Excel/CSV reports is not fully automated in the user interface.

Recommendations:

- We recommend implementing robust password hashing (e.g., using bcrypt or Argon2) before live deployment to ensure student data security.
- The hosting environment should be transitioned to a university-managed server or cloud instance to allow remote access for all pantry staff.
- A dedicated "Reports" dashboard should be developed in the UI that executes the SQL queries for monthly usage stats and offers a one-click download.

X.2. Future Work

The immediate future work involves the full integration of our backend system with the partner team's WordPress front-end. This will likely involve creating RESTful API endpoints within the Flask app to serve data to the WordPress site. Additionally, the physical integration of the barcode scanner and Cougar Card reader needs to be finalized, moving from manual entry fields to direct hardware input. Long-term, we believe the project could be commercialized or packaged as a "Pantry-in-a-Box" open-source solution for other universities facing similar data tracking challenges.

XI. Acknowledgements

We would like to thank the following individuals for their support and guidance throughout this project:

- Maynard Siev, our primary client, for his consistent feedback and dedication to the Voiland Food Pantry.
- Ananth Jillepalli, our coach, for his weekly feedback, tips, and dedication to supporting our team.
- Gary Offerdahl, Lisa Carmack, and Alena Hume, for their valuable insights during joint team meetings.
- The Website Team (CPTS 421), for their collaboration on the front-end integration planning.
- Washington State University EECS Department, for providing the resources and opportunity to work on this capstone project.

GLOSSARY

Admin

An Administrator user with full access and privileges to manage the entire system, including user roles, inventory, and reporting.

Barcode Scanner

A hardware peripheral used to read item barcodes for streamlined inventory tracking and distribution recording.

Cougar Card Reader

A hardware peripheral used to scan Washington State University student ID cards to facilitate client check-in and usage tracking.

Database Team

The team responsible for designing, implementing, and maintaining the project's back-end data structure and logic (Ram Logic).

ER Diagram

Entity-Relationship Diagram; a visual model of the database structure defining the relationships between data entities (tables).

Inventory

The collection of food and hygiene products currently available and tracked by the Voiland Food Pantry.

Voiland Food Pantry

The client organization and subject of the Data Tracker project (Voiland Food Pantry & Wellness Center).

WordPress

The content management system (CMS) selected as the platform for the front-end web application.

REFERENCES

R.1 Client Meeting Agenda and Minutes (9/3/2025, 9/10/2025, 10/2/2025).

R.2 Project Report: *Ram Logic

R.3 Initial Database Schema Definition (*Fiz Project/App/DB Files/schema.sql*).

R.4 Sprint Report 5 (Future work and coordination items).

R.5 Entity-Relationship Diagram Draft (*ER_FoodPantry_DiagramDraft2.drawio.pdf*).

XIV. Appendix A – Team Information

Team Ram Logic

Alan Sun

- Degree Plan: Computer Science
- Project Role: Backend Developer & Security Lead. Responsible for Flask application development, implementing user authentication, and ensuring secure data handling practices.
- Technical Interests: Backend web development, cybersecurity, and system integration.

Cristobal Escobar

- Degree Plan: Computer Science
- Project Role: Database Architect & Full-Stack Developer. Contributed to designing the MySQL database schema, developing the entity–relationship model, and implementing backend logic for data handling and validation. Also assisted in building the Flask prototype, creating SQL schema and seed files, and coordinating requirement updates with the client.

- Technical Interests: Machine learning, software engineering, and data-driven system design.

XV. Appendix B - Example Testing Strategy Reporting

1) Requirement Identified:

- Requirement: The system must restrict access to administrative functions (e.g., adding inventory, managing volunteers) to users with the 'Admin' role.
- Test Case: Attempt to access the /add_item route while logged in as a 'Student' user.

2) System Testing Results:

- Test Action: Logged in with student credentials (username: asmith) and navigated to http://localhost:5000/add_item.
- Result: The system correctly identified the unauthorized role and redirected the user to the home page with a flashed message or restricted view.

3) User Testing Feedback:

- Feedback Source: Client demo session (10/29/2025).
- Quote/Discussion: "The login flow is simple, but we need to make sure volunteers can easily see their shift status immediately after logging in."
- Action Taken: The home page template (home.html) was updated to conditionally display relevant buttons based on the user's role (Admin, Volunteer, or Student).

XVI. Appendix C - Project Management

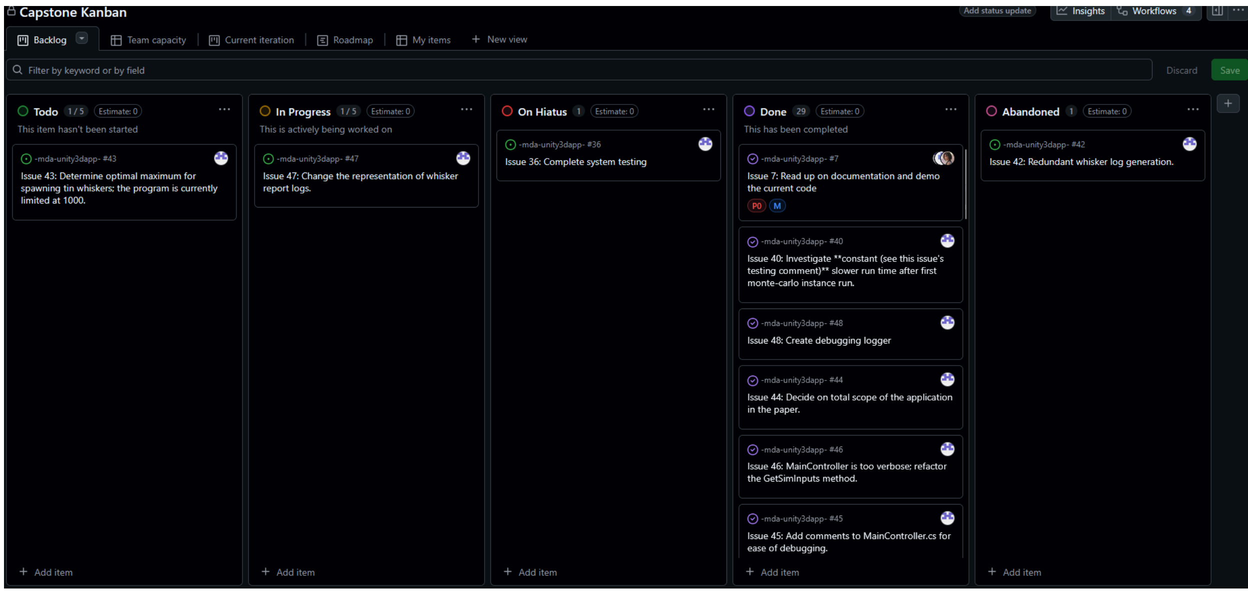
Weekly Schedule:

- Weekly Meetings with Mentor (Ananth Jillepalli):
    - Purpose: To review high-level progress, discuss potential blockers, and ensure the project aligns with course requirements and academic standards.
    - Routine Agenda: Status update on current sprint deliverables, review of documentation drafts, and guidance on upcoming milestones (e.g., poster submission, final report).
- Weekly Meetings with Client (Maynard Siev):
    - Purpose: To demonstrate new features, clarify requirements, and receive direct feedback on the prototype's functionality.

    - Routine Agenda: Demo of the latest build (e.g., "Here is the new inventory page"), discussion of specific features (e.g., "Do we need a 'weight' field?"), and setting goals for the next sprint.
- Internal Team Meetings (2-3 times per week):
    - Purpose: To coordinate development tasks, debug code together, and prepare for client/mentor presentations.
    - Routine Agenda: Code reviews, merging branches in GitHub, updating the issue tracker, and dividing upcoming tasks.

Beneficial Activities: The weekly client meetings were the most beneficial activity. They allowed us to pivot quickly when requirements changed (e.g., adding the 'Donors' table concept) and ensured we were building a tool that would actually be useful to the pantry staff, rather than just guessing at their needs.

Planning Documents & Tools:

- GitHub Projects & Issues: We used GitHub Issues to track individual tasks (e.g., "Create database schema," "Design login page"). We organized these into Sprints using the Projects board view to visualize our workflow.
    - 

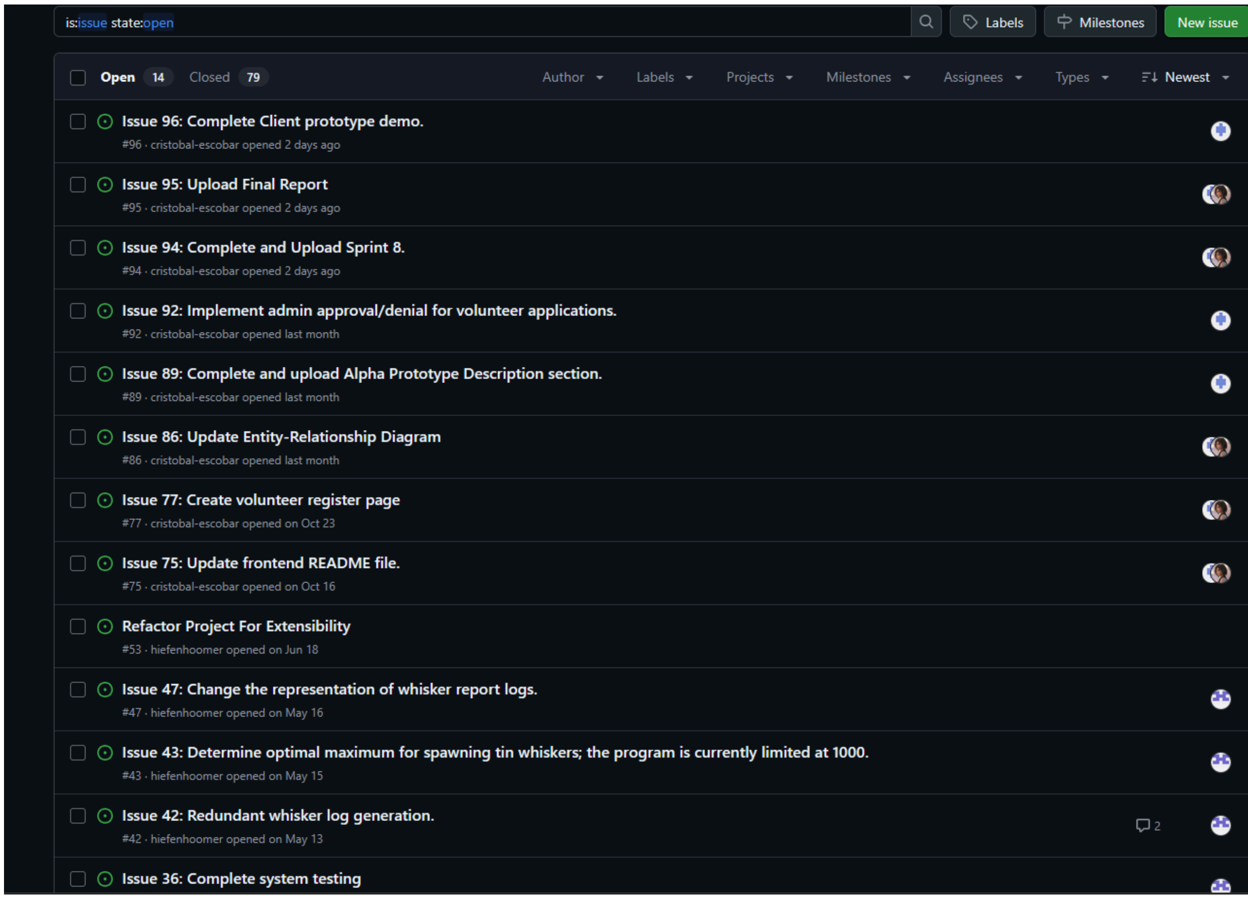
Todo 1 / 5 Estimate: 0
This item hasn't been started
-mda-unity3dapp- #43
Issue 43: Determine optimal maximum for spawning tin whiskers; the program is currently limited at 1000.

In Progress 1 / 5 Estimate: 0
This is actively being worked on
-mda-unity3dapp- #47
Issue 47: Change the representation of whisker report logs.

On Hiatus 1 Estimate: 0
-mda-unity3dapp- #36
Issue 36: Complete system testing

Done 29 Estimate: 0
This has been completed
-mda-unity3dapp- #7
Issue 7: Read up on documentation and demo the current code
P0 M
-mda-unity3dapp- #40
Issue 40: Investigate **constant (see this issue's testing comment)** slower run time after first monte-carlo instance run.
-mda-unity3dapp- #48
Issue 48: Create debugging logger
-mda-unity3dapp- #44
Issue 44: Decide on total scope of the application in the paper.
-mda-unity3dapp- #46
Issue 46: MainController is too verbose; refactor the GetSimInputs method.
-mda-unity3dapp- #45
Issue 45: Add comments to MainController.cs for ease of debugging.

Abandoned 1 Estimate: 0
-mda-unity3dapp- #42
Issue 42: Redundant whisker log generation.

◦ 

is:issue state:open

Labels | Milestones | New issue

Open 14 Closed 79 — Author, Labels, Projects, Milestones, Assignees, Types, Newest

- Issue 96: Complete Client prototype demo.
  #96 · cristobal-escobar opened 2 days ago
- Issue 95: Upload Final Report
  #95 · cristobal-escobar opened 2 days ago
- Issue 94: Complete and Upload Sprint 8.
  #94 · cristobal-escobar opened 2 days ago
- Issue 92: Implement admin approval/denial for volunteer applications.
  #92 · cristobal-escobar opened last month
- Issue 89: Complete and upload Alpha Prototype Description section.
  #89 · cristobal-escobar opened last month
- Issue 86: Update Entity-Relationship Diagram
  #86 · cristobal-escobar opened last month
- Issue 77: Create volunteer register page
  #77 · cristobal-escobar opened on Oct 23
- Issue 75: Update frontend README file.
  #75 · cristobal-escobar opened on Oct 16
- Refactor Project For Extensibility
  #53 · hiefenhoomer opened on Jun 18
- Issue 47: Change the representation of whisker report logs.
  #47 · hiefenhoomer opened on May 16
- Issue 43: Determine optimal maximum for spawning tin whiskers; the program is currently limited at 1000.
  #43 · hiefenhoomer opened on May 15
- Issue 42: Redundant whisker log generation. 2
  #42 · hiefenhoomer opened on May 13
- Issue 36: Complete system testing

- Team Tools Used:
  - Microsoft Teams: Used for all video conferencing with the client and for quick chat/file sharing between team members.
  - Discord: Used for informal, rapid communication and screen sharing during pair programming sessions.
  - Visual Studio Code (Live Share): Utilized for collaborative coding sessions, allowing both members to edit files simultaneously.