Food-For-You

Software Requirements Specification

Linnea Gilius (lg), Krishna Patel (kp), Jerry Pi (jp), Katherine Smirnov (ks),
Lauren Van Horn (lvp) | 3-12-2023 | v2

1. SRS Revision History

Date	Author	Description
2-23-2023	lg, ks	created the initial document
2-23-2023	lg, ks	added ConOps and specific requirements
2-25-2023	ks	added sections 3.1 and 3.2
2-26-2023	lg	added sections 2.6, 3.3, 3.4, and 3.5
2-26-2023	ks	edited sections 3.1 and 3.2
2-28-2023	lg	edited sections 2.6, 3.3, 3.4, and 3.5
3-11-2023	lg, ks	revision of all sections, incorporating suggestions from Professor Hornof

2. The Concept of Operations (ConOps)

The goal of this project is to build a system called *Food-For-You* that people can use to streamline the process of getting food from and donating food to a food bank. The system would allow the food bank to publish what they currently need so donors can view that information. Recipients of food would also be able to view what items each food bank has so they can go to the location that has what they need.

2.1. Current Situation

There are three main users that could benefit from this system: food bank staff, food donor, and food recipient.

Through interviews with food bank staff, we learned that it is not easy for food banks to communicate their constantly changing needs with other food banks, and view what food is available in other food banks. Food pantries currently order from a spreadsheet, which can create a delay in getting food to food pantries. The distribution of food items across food banks is not known, so when sending items to food banks, it is not known if the food bank is in dire need of new items compared to other food bank.

The food banks currently have a website located at https://foodforlanecounty.org/wp-content/uploads/ 2021/03/Most-Wanted-2020.pdf that shows the most wanted foods. Though, this document doesn't display which locations, or food items would be in need. From an interview with a food donor, they stated they were willing to drive up to 30 minutes to donate food. If the donor was able to know which location was in the most need of food, this would help streamline the food items to the correct food banks.

The closest resource they have is a website at https://foodforlanecounty.org/find-a-food-pantry/ that shows where different food pantries around Eugene are. This doesn't allow food recipients to know what specific food is available and where.

2.2. Justification for a New System

Food-For-You would allow donors to view which donation centers would be in need of food, and which specific foods are in need of donations. This would streamline the needed foods to the needed areas. The users who may be in need of certain foods would be able to see which centers have their desired foods, which would allow users to know which food bank to go to when they are in need of a certain item. This system would allow more efficient donations and more accessible food.

2.3. Operational Features of the Proposed System

The new system would indicate what specific foods the food bank needs and would show what they have. This would allow donors to see what the food bank is currently short on and would allow those who are in need of food to see what the food bank has. Additionally, all users would be able to filter by location so they can conveniently see what is closest to them.

2.4. User Classes

The three main user classes include:

1. Food recipients who need resources and trying to receive food from a food bank.
2. Donors who are trying to donate food into a food bank.
3. Food bank staff who can update the food bank database.

2.5. Modes of Operation

There are two main modes of operation: one for food bank staff and one for food donors and recipients. The mode of operation for food banks allows each food bank to update the Food Resource Database with the items they have received and the items they have distributed. The mode of operation for the donors and recipients allows them to view the inventory at all food banks or the inventory at each food bank. It also allows them to see the various food banks and their locations, which would help them locate their nearest food bank.

2.6. Operational Scenarios (Also Known as “Use Cases”)

Use Case: food recipients look for food banks to receive food from

Brief description: This use case describes how a food recipient would look for a food bank to receive food from.

Actors: a food recipient

Preconditions:

1. The food recipient is in need of food.
2. The food recipient has access to a computer.

Steps to Complete the Task:

1. The food recipient runs the food recipient program (RecipientUI.py) in the command line.
2. The food recipient determines the neighborhood that would be the most convenient for them to go to, or decides to view all neighborhoods.
3. The food recipient determines the food category (if any) that they are most in need of.
4. The food recipient selects the neighborhood and food category they want from the interface. The food recipient also selects if they only want to view the food banks that are currently open, as well as if they want to view food banks’ addresses, hours, and/or phone numbers.

Postconditions:

The food recipient has the locations of food banks which best fit their preferences.

Use Case: food donors look for food banks to donate food to

Brief description: This use case describes how a food donor would look for a food bank to donate food to.

Actors: a food donor

Preconditions:

1. The food recipient has food to donate.
2. The food recipient has access to a computer.

Steps to Complete the Task:

1. The food donor runs the food donor program (DonorUI.py) in the command line.
2. The food donor determines the neighborhood that would be the most convenient for them to go to deliver food to.
3. The food donor determines the food category (if any) that they have the most excess of.
4. The food donor selects the neighborhood and food category they want from the interface. The food recipient also selects if they only want to view

the food banks that are currently open, as well as if they want to view food banks' addresses, hours, and/or phone numbers.

Postconditions:

The food donor has the locations of food banks which best fit their preferences.

Use Case: food bank staff update the database with new inventory

Brief description: This use case describes how a staff member at a food bank would update the Food Resource Database inventory.

Actors: food bank staff member

Preconditions:

1. The food bank staff member has access to a computer.
2. The food bank staff member needs to modify an item (update, move, delete, add).

Steps to Complete the Task:

1. The food bank staff member runs the food bank staff program (staffUI.py) in the command line.
2. The user may choose to filter by the data by inserting the desired filters in the input boxes. The user then selects a "Search" button.
3. The user selects the row in the table which they choose to modify.
4. The user views the "update" window and selects from the top dropdown an action: update, move, delete (defaults to update).
5. The user inputs the required fields, and selects a "Save Changes"/"Confirm deletion" button.

Postconditions:

The Food Resource Database is updated to reflect the modified inventory. The table in the program displays such changes.

Use Case: food bank staff update the database with new inventory

Brief description: This use case describes how a staff member at a food bank would add new items to the Food Resource Database.

Actors: food bank staff member

Preconditions:

1. The food bank staff member has access to a computer.
2. The food bank staff member has an item to add to the database.

Steps to Complete the Task:

1. The food bank staff member runs the food bank staff program (staffUI.py) in the command line.
2. The user selects an “add item” button.
3. The user fills in all prompted fields.
4. The user selects a “Save Changes” button.

Postconditions:

The Food Resource Database is updated to reflect the new inventory. The table in the program displays such changes.

Use Case: food bank staff update the database with new inventory

Brief description: This use case describes how a food bank administrator member at a food bank would add a new food bank to the Food Bank Table.

Actors: food bank administrator

Preconditions:

1. The food bank staff member has access to a computer.
2. The food bank staff member has the necessary information about the new food bank: Open and closing times, food bank location, food bank name, neighborhood, phone number, and inventory data in a CSV file (optional).
3. The table in the Food Bank tab is empty.

Steps to Complete the Task:

1. The food bank staff member runs the admin program (AdminView.py) in the command line.
2. The user selects the “Add food bank” tab.
3. The user selects a “Add food bank” button.
4. The user inputs the information about the food bank (see preconditions).
5. The user uploads a CSV file of new inventory (optional).
6. The user selects a “Save Changes” button.

Postconditions:

The Food bank tab displays the uploaded inventory data as a table.

Use Case: food bank staff update the database with new inventory

Brief description: This use case describes how a food bank administrator member at a food bank would view and search the Outgoing table from the database.

Actors: food bank administrator

Preconditions:

1. The food bank staff member has access to a computer.

Steps to Complete the Task:

1. The food bank staff member runs the admin program (AdminView.py) in the command line.
2. The user may choose to filter by the data by inserting the desired filters in the input boxes. The user then selects a “Search” button.

Postconditions:

The data from the outgoing table is displayed in the “Outgoing tab”.

3. Specific Requirements

3.1 Donor Display

3.1.1. External Interfaces (Inputs and Outputs)

I. Food Category (string): Allows the user to search the database based on food categories. User selects from a dropdown of food categories.
II. Neighborhood (string): Allows the user to display the food banks in the given neighborhood. User selects from a dropdown of neighborhoods.
III. Display filters:
    A. Phone Number (Boolean): Allows the user to view food banks’ phone numbers.
    B. Street Address (Boolean): Allows the user to view food banks’ addresses.
    C. Today’s Hours (Boolean): Allows the user to view food banks’ hours for today.
    D. Open Now (Boolean): Allows the user to only view the food banks that are currently open.
IV. Standard Output: Outputs the filtered data to Standard Output, filtered based on (I-III).
V. File: Contains the filtered data which was outputted to Standard Output as described in (IV).

3.1.2. Functions

I. Receiving Inputs: User inputs are accepted from an interface and type-validation is performed.
II. Searching: Database is searched based on the user’s inputs.
III. Writing to File: The results are processed and filtered based on the user's display filters (outlined in 3.1.1.III). The results are then written to a file.
IV. Displaying: The results, which were processed in (III) above, are displayed to Standard Output.

3.2 Recipient Display

3.1.1. External Interfaces (Inputs and Outputs)

I. Neighborhood (string): Allows the user to display the food banks in the given neighborhood. User selects from a dropdown of neighborhoods.
II. Food Category (string): Allows the user to search the database based on food categories. User selects from a dropdown of food categories.
III. Display filters:
    A. Open Now (Boolean): Allows the user to only view the food banks that are currently open.
    B. Today's Hours (Boolean): Allows the user to view food banks' hours for today.
    C. Address (Boolean): Allows the user to view food banks' addresses.
    D. Phone Number (Boolean): Allows the user to view food banks' phone numbers.

IV. Standard Output: Outputs the filtered data to Standard Output, filtered based on (I-III).
V. File: Contains the filtered data which was output to Standard Output as described in (IV).

3.1.2. Functions

I. Receiving Inputs: User inputs are accepted from an interface and type-validation is performed.
II. Searching: Database is searched based on the user's inputs.
III. Writing to File: The results are processed and filtered based on the user's display filters (outlined in 3.1.1.III). The results are then written to a file.
IV. Displaying: The results, which were processed in (III) above, are displayed to Standard Output.

3.3 Staff view: displaying, updating and viewing the data.

3.3.1. External Interfaces (Inputs and Outputs)

I. Update existing row: Allows users to change the quantity or units of a row. Quantity can be zero or a positive integer. There will be a text input box in which these actions can be inserted. Updates are sent to the Food item table. If the quantity is decremented, such difference is sent to the Outgoing table.

II. Move an item: The user may input a quantity move and the location to move to. These will be text inputs and dropdowns respectively. The dropdown for the locations will pull all the locations currently in the food item database. The quantity input must be a positive integer less than the current quantity. The updates are sent to the Food item table and outgoing table.
III. Delete an item: The user may choose to delete a row in the database.
IV. May filter the data which will send queries to the database:
    A. Search by item by a text input
    B. Search by item ID by text input
    C. Search by location by dropdown
    D. Sort by ascending quantity
V. New Item: Allows a new row to be inserted in the Food Item database. The user must input category and location by dropdown, item, quantity and units by text entry. Quantity must be zero. The location and category dropdowns will only allow selection of those that are already in the Food Item database. This new item is queried into the Food Item database.

3.3.2. Functions

I. Update an existing row:
    A. If the data check is successful, calls a query to change the row to have the inputted quantities and units of the item.
    B. If the quantity decreases, the difference is logged in the Outgoing table, where all columns are the same, except the difference is the quantity.
II. Move an item
    A. If the inputted quantity is less than the current quantity, then the inputted quantity is to subtract the moving quantity from the current item, and increase the such quantity to the food item with such item name, unit and inputted location. If no food item exists at the new location, a new entry in the created.
III. Delete an item
    A. The item is deleted from the food bank table.
IV. Filter by data: Calls SQL query and populates tkinter table.
V. New Item: Checks if there is an existing row in the table with such item name, units and location. If so, combine these rows by adding the quantities, else will create a new row.

3.4 Admin View:

3.4.1. External Interfaces (Inputs and Outputs)

I. Add new Food bank button: The user must input the following below (except file upload). Such data is sent to the Food Item table and the Food Bank table
   A. File upload: Allows a mass insertion of data. Users can load a CSV file through a file uploader. Inserts such data into the database. CSV must have headers of “food item”, “category”, “quantity”, “units”, with the food item, category and units being non-empty strings, quantity being non-negative digits.
   B. Insert food bank times: Time widget to allow user to input military time for open and closing time of each day. The close time must be after the open time.
   C. A text entry for street address, food bank name, neighborhood, and phone number. Phone number must be in the format (XXX) XXX-XXXX.

II. Table displaying newly inserted data (if new food bank was imported).

VI. View the Outgoing table
   A. May filter the data which will send queries to the database: search by item and item ID by text input, location by dropdown, and sort by ascending quantity.

VII. May export data in the outgoing table to a CSV file.

3.4.2. Functions

I. New food Bank: File Upload
   A. User inserts a file to the file uploader, and data checking is performed:
      1. Verifies file is .csv extension. For incorrect extension, throws an error message asking for a csv only, and returns to the file uploader.
      2. Verifies the file has the correct headers: “food item”, “ounces”, “quantity”, and “location”. For incorrect header, throws an error message displaying the necessary headers, and returns to the file uploader.
      3. Verifies all the quantities are non-negative integers. For incorrect data, throws an error message displaying the necessary quantity types and which row was incorrect, and returns to the file uploader.
      4. Verifies all rows/columns contain data. For incorrect data, throws an error message detailing how each food item row must be filled and

displaying which row was incorrect, and then returns to the file uploader.

B. Upon successful file uploading, the data is inserted to the table, with the location inputted . Will compare existing rows with new rows based on equivalent food item and location. If two rows are the equivalent (same food item, and units), the quantities are summed together, and if not, will create a new row of the new data.
C. If no file is uploaded, no data is added to the Food Item table.
D. Food Bank details are added to the Food Bank Table.
    1. Comparing Times: if two times are equal, no time is inputted for such day.

II. Fetch Data from the Food Bank table filtered by the newly inserted location.
III. View the Outgoing table: Calls SQL query from outgoing table and populates tkinter table.
IV. Export Data: Converts database to CSV file.

3.5 Usability Requirements

- For all modules, the user has the functionality of searching by a criteria; the user should be able to search by item name, item ID, location, and sort by quantity in 20 seconds.
- The staff view should allow staff to modify (update, move, delete or add) an item in no longer than 20 seconds.
- If a user provides an input that is not the correct data type, the program will throw an error which states the given input and the expected input.
- The admin view should allow staff to insert a new food bank in under a minute.

3.6 Performance Requirements

- For all modules, the user has the functionality of displaying the food item table; the system should not take longer than one second filter and show the data matching the user's preferences. The system should be able to handle at least 10 users at a time and should be able to process inventory and distribution lists of at least 100 items, across all modules.
- It should not take longer than 100 ms for the database to be updated after a food bank staff member submits their changes using the user interface.

3.7 System Requirements

The Food Resource Database should follow the ACID properties (atomicity, consistency, isolation, and durability). Therefore, when a food bank staff member uploads an inventory or distribution CSV file, the database should be able to complete all of the updates specified by the file or none of them. If one update fails, the database is rolled back to its previous state. Furthermore, each update should be isolated from other updates that are executing concurrently. This means that the changes made by a later update should not be visible until the first update has completed. Finally, once an update has been submitted, it should be permanent and should survive any subsequent failures in the database.

4. References

IEEE Std 1362-1998 (R2007). (2007). IEEE Guide for Information Technology–System Definition–Concept of Operations (ConOps) Document. https://ieeexplore.ieee.org/document/ 761853

IEEE Std 830-1998. (2007). IEEE Recommended Practice for Software Requirements Specifications. https://ieeexplore.ieee.org/document/720574

ISO/IEC/IEEE Intl Std 29148:2011. (2011). Systems and software engineering — Life cycle processes — Requirements engineering. https://ieeexplore.ieee.org/document/6146379

ISO/IEC/IEEE Intl Std 29148:2018. (2018). Systems and software engineering — Life cycle processes — Requirements engineering. https://ieeexplore.ieee.org/document/8559686

Faulk, Stuart. (2013). *Understanding Software Requirements*. https://projects.cecs.pdx.edu/attachments/download/904/Faulk_SoftwareRequirements_v4.pdf

Oracle. (2007). White Paper on “Getting Started With Use Case Modeling”. Available at: https://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf

van Vliet, Hans. (2008). *Software Engineering: Principles and Practice*, 3nd edition, John Wiley & Sons.

5. Acknowledgements

We used information from interviews conducted with a food donor, a food receiver, and a food bank staff member.
We incorporated feedback from Professor Anthony Hornof, received on March 8, 2023.