Requirements, Vision, and Scope

Fridge Manager

Authors: Xuewei (Vivy) Wang, Jason Wang, Raphael Strebel, Sangeetha Kamath, Kazuki Fukushima, Jeffrey Chen

---

Product Vision Statement

Motivation / Opportunity

Our goal is to implement an android app to solve the problem of not knowing what food is currently at home and when it expires. The advantage of our product over pre-existing apps is that users are able to scan the grocery bill with their smartphone and the app will provide them with a list of all foods the users purchased and automatically add them to their local food database.

Problem Statement

The problem of	Food wastage caused by expired food
affects	Consumers looking to reduce waste and maintain an overview of groceries available to be used without having to spend time on tracking grocery purchases.
The impact of which is	Solving the problem of food waste by reminding users of the expiry date for each product. An example case would be meat that spoils quickly and is expensive which might have been forgotten by the user.
A successful solution would be	A program that provides an overview of your groceries currently at home, in the fridge, pantry, or freezer and then provides information on nutritional value and sets reminders for the expiry dates of the said items.

Product Position Statements

For	All consumers who make regular grocery purchases.
Who	Use the app to manage food items and prevent food wastage
Our System	Android app; minimum v4.0.3 (ICE_CREAM_SANDWICH_MR1)
That	Automatically scans receipts, adds food to local food database, and reminds the user of food about to expire.
Unlike	Other fridge manager apps such as fresh box, fridge pal, best before, or grocery hero.
Our Product	Allows users to scan receipts, automatically adds the food items into foodstock database, and provides updates to expiry date information.

Users: User of our app is anybody who wants to have a clear overview of the current contents in the food stock (fridge / freezer / pantry / etc.).

Feature List:

- Ability to take a picture of grocery receipts and recognize and analyze text
- Search through database for expiry date of food item and input groceries onto list and display expiry date through UI
- Visual interface of food stock; allows user to add, remove and decrease groceries from food stock to keep track of quantities remaining
- Ability to remove items and manually add items
- Categorize items by physical location they might be placed (items that might go in the fridge, freezer, or pantry)
- Ability to notify soon-to-expire foods in stock

Constraints: App has to be running on Android devices with minimum version 4.0.3, and built-in camera support.

Scope and Limitations: App doesn't provide any recipe suggestions for the food in the food stock or suggestions on which food should be bought in the future. It also does not provide the ability to scan barcodes and provide information about the item.

Assumptions and Dependencies: We expect that the text recognition service, Google Mobile Vision, allows us to scan grocery bills and output text, including blocks, lines, and words.

Use Cases

1. **Analyze and Read Text from Receipt**
    a. Primary Actor : End user
    b. Stakeholders : End user
    c. Precondition : App must be running on an Android device with Camera support and the receipt must be in English, with minimal quality damage
    d. Postcondition : All food items will be extracted from the receipt
    e. Main Success Scenario :
        1. User takes a picture of the receipt
        2. App receives the image and processes words using OCR Reader code from Google Mobile Vision
        3. Google Mobile Vision processes the image to retrieve a list of lines on the image
        4. App receive the lines and retrieves one food item per line if exists
        5. App displays the list of food items it was able to identify from the receipt
        6. User confirms that recognized items are linked with the right food items.
    f. Extensions and Alternative Flows :
        1. Image was unreadable or some food was not recognized
            1. App prompts the user to make a picture again or manually enter items
        2. Food item on one line of receipt was unrecognized

1. User enters the corresponding food manually
2. App updates the library with unknown item and entered name

2. Search Expiry Date Database for Expected Expiry Date

a. Primary Actor : App
b. Stakeholders : End user
c. Precondition : Food item must be a valid item contained in the library
d. Postcondition : Returns an expiry date for the corresponding food
e. Main Success Scenario :
    1. App accesses library to retrieve number of days until expiry for all food items
    2. App searches through the list of days to retrieve the day of corresponding food item
    3. App calculates the actual expiry date from the current date from the phone and days until expiry from the library
f. Extensions and Alternative Flows :
    1. App gives a wrong expected expiry date
        1. User selects the food item for which the expiry date was wrong
        2. User selects the correct expiry date from the calendar

3. Manually Add Food to Foodstock Database

a. Primary Actor : End user
b. Stakeholders : End user
c. Precondition : The food item must be composed of valid ASCII characters
d. Postcondition : Food item is added into the foodstock database and can be viewed by the end user
e. Main Success Scenario:
    1. User selects add food and manually enters a food name. The app can suggest auto-complete word from the library
    2. User enters any abbreviations for the food, quantity, food location, the unit of the food, and the food's expiry date
    3. App verifies the name so that it contains valid characters
    4. App adds the food item to the foodstock database
    5. App updates the changes to the foodstock database so that it is visible to the user
f. Extensions and Alternative Flows
    1. Food item already exists in the foodstock database with same expiry date
        1. App increases the quantity of the corresponding food item
    2. Quantity entered was 0 or smaller, or the expiry date entered was past
        1. App prompts a message suggesting to enter a valid quantity and/or expiry date
    3. User did not enter a food name
        1. App prompts a message to enter a valid food name

4. Manually Edit Expiry Date of Food Item

a. Primary Actor : End user
b. Stakeholders : End user

c. Precondition : Food item already exists in the food stock
d. Postcondition : Expiry date for the food item is updated or created
e. Main Success Scenario :
    1. User selects a food from food stock user interface
    2. User manually enters the expiry date for that food item through a calendar interface
    3. App verifies that the date is a valid day
    4. App links the entered expiry date with the food item
    5. App adds the expiry date specified from the user to the corresponding food, and updates new change on the user interface
f. Extensions and Alternative Flows :
    1. The expiry date is in the past
        1. App alerts the user of past expiry date
        2. App prompts the user to input new expiry date

5. Remove / Decrease Food from Foodstock Database

a. Primary Actor : End user
b. Stakeholders : End user
c. Precondition : Food item already exists in the foodstock database
d. Postcondition : Quantity of the food item decreases by specified ammount or is deleted
e. Main Success Scenario :
    1. User access the foodstock
    2. User presses decrease or delete button
    3. App decrements by specified amount or deletes item
f. Extensions and Alternative Flows :
    1. Quantity reaches 0 upon decrementing
        1. Corresponding food is removed from foodstock database
        2. No alert will be sent to the user

6. Alert User of Food About to Expire

a. Primary Actor : App
b. Stakeholders : End user
c. Precondition : A food item exist in the foodstock database and is about to reach its expiry date within 3 days
d. Postcondition : User receives a mobile notification from the app
e. Main Success Scenario :
    1. App finds a food item that is expiring within 3 days in the foodstock database
    2. App sends a notification to the user at specified time of the day (by default 18:00) and displays a notification when the food expires
    3. The user confirms the message with the OK button
f. Extensions and Alternative Flows
    1. The smartphone was off, when the message should have been displayed and the food already expired
        1. The user gets informed when the food item expired once the phone is turned back on

7. **Scanning Long Receipt**
    a. Primary Actor : End user
    b. Stakeholders : End user
    c. Precondition : Reciept is longer than the range of the camera view
    d. Postcondition : All food items will be extracted from the receipt
    e. Main Success Scenario :
        1. User takes a picture of the receipt
        2. App receives the image and proccesses words using OCR Reader code from Google Mobile Vision
        3. App displays the list of food items it was able to identify from the receipt
        4. User can press “add more” to take another picture of the rest of the receipt
    f. Extensions and Alternative Flows
        1. List of food items contains duplicates due to user taking a picture of the same food item again
            1. User manually edit the amount of food item to the actual amount

Non-functional Requirements

Performance requirements:

- Finishes processing each photo and adding food items under 4 seconds ensuring that the app will faster than writing down all the food items shown on receipt
- Any additional buttons (eg. Undo) should have an instantaneous-like (0.1s) response time ensuring the user experience is of quality and no lag time is experienced

Software Quality Attributes:

- Ease of use over ease of learning; the app environment should be easy to pick up due to common UI and should not be difficult to learn anything unfamiliar
- Reliability, availability, scalability, and recoverability (RASR) due to database usage.
    - The database should be reliable by being free of technical errors and have consistent performance. It is always readily available to be used as database are stored locally on the phone. As the database is being populated, it should not be hindering the app’s performance significantly. Lastly, the database should be able to retain/recover its information even if the app crashes unexpectedly

Security Requirement:

- Personal information from receipts (store location/credit card information) processing should be kept private to users only

User Demographics

End Users: Only Android users will have access to this mobile app. They are expected to have experience with an android UI. They aren’t expected to have any programming skills.

Rationale for Changes

Constraints

We added in an additional constraint information which is that the android device running this app must have a minimum version of 4.0.3. This is important as the device must be able to run libraries with an API level of 15 or the app will not run properly on the user's device.

Use Cases

Many of our original use cases were either unclear or were more suitable as an alternate flow for existing use cases instead of a use case. The use cases were modified to be more informative and concise but relevant.

Many of the original use cases still remains in this updated document. However, we introduced one more use case: Scanning Long Receipt. This use case is important because the length of the receipt is dependent on how many food items the user purchases. If the length of the receipt exceeds the screen size that the camera can capture, the user should be able to finish scanning the rest of the receipt and handle any duplicated food item with ease.

In use case 3, we included the possibility to add abbreviations to an existing food item. This is important because many receipts only show abbreviations for food items. The items would not be recognized even if they were in the library. Abbreviations that are linked to a specific food item in the library makes it possible to recognize most food items. Its alternative flows now includes an prompted message for invalid input to make sure that the app always works with valid data.

In use case 5, we added the possibility to decrease the amount of food. This gives the user a better overview of his current food stock.

Non-functional Requirements

Non-functional requirements are now elaborated with reasonings on why the requirements were proposed. Some requirements from the original document were removed or modified because they were either incorrect or no longer relevant to our product.