Design Document

Fridge Manager

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

---

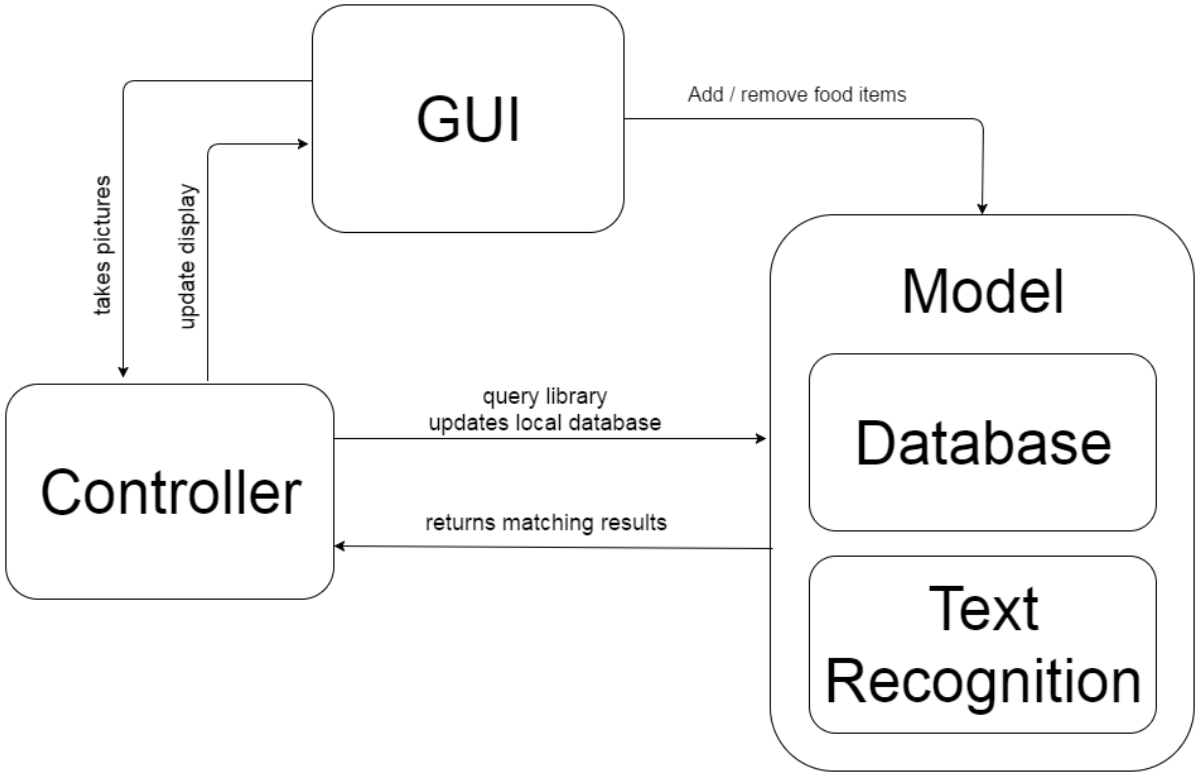
Introduction

Fridge Manager is a mobile application designed to help people manage contents in their fridges with convenience and efficiency. Nowadays, people are busy and lacks the time to manage their food in their fridge. With a click of a button, the user can scan their grocery bills and have their food item organized and stored onto a foodstock database on their phone. This application alleviates the issue of food wastage by reminding users when their food are about to expire.

Architecture and Rationale

The Android Application consists of three main components:

1. Model: Local and Main Library Database
2. View: User Interface
3. Controller : Optical Character Recognition, Database Interaction, and Algorithms



Database: Javascript Object Notation (JSON)

The mobile application will require a local foodstock database to hold the list of items in the fridge (or elsewhere such as freezer, or pantry box) and a library database containing all valid food items with attributes like unit and expiry date. Additionally, the application will require a stack implementation to perform allow undos of up to 10 past moves (such as delete and decrement of food item). This stack needs to exist outside the Java class, inside a file-format, so that the stack remains even after closing the application. These databases will be implemented in a simple fashion with JSON. The primary reason for choosing JSON over other databases formats is that it is light-weight. Our intent was that since other components of the application, such as the text recognition, will require some memory space, it is preferable to limit other components. In fact, our database will not be storing an exponentially increasing amount of food, so the JSON format will be sufficient. JSON gives minimal complexity, as well as portability in the case when we want to move the database to different environment in future updates.

Main Library Database

The main database holds all of the characteristic of food item such as the name of the food, the average expiry time of that particular food, the abbreviation of the food name, the unit associated with the food, and the location where the food is stored. The default library database exists in the assets folder of the application project, meaning that it will never be changing during runtime. When the application opens for the first time, a copy of the default database is moved to resources directory (which allows overwriting files) of the application. By doing so, it allows for the application to keep a secure working copy as well as a copy that can be extended by the user. This database can be extended by the user if the character recognition encounters an unrecognized food. In this scenario, the user will have the choice of adding the unrecognized food to the main database with its characteristic so that in the future, the application will then be able to recognize the previously unrecognized food.

Local Foodstock Database

The foodstock database holds all of the current foods that the user have in their fridge, freezer, fresh box, or pantry. This database will be updated constantly whenever the user wants to input more food or if current food has expired (in which the item would then be removed from the foodstock database). This database also holds the name of the food, when the food was bought, and along with the pre-assumed average expiry date, the associated unit that is retrieved from the main database.

By having two databases, we are able to parse through each database more efficiently and “clever” designs with complex algorithm are avoided. Minimizing the complexity means that we are parsing each database separately rather than developing a single algorithm to parse one single long database. In addition, ease of maintenance comes into play when dealing with two databases as updates to one database will not affect the other. If the main library database becomes too large, we can move it to an online server without causing a big change to the local foodstock database. Fundamentally, separating the local foodstock database and the main library database is more organized and makes it less confusing for the developers.

Undo Stack File

The undo stack file is a stack designed using JSON format. Although JSON does not natively support stack, we simply implemented it using existing functions of JSON. The primary reason for choosing JSON format over formal types of stack or databases is because utility methods for communication with JSON format files already existed in our project. Since the stack will only contain at most 10 past moves and will not regularly be popped, code reuse was preferred over formal design.

User Interface

The user interface is set up using separate Android Activities and Fragments, each one representing a single screen with a user interface, similar to a window or frame of Java. We have one activity for each action that involves the user. We used the minimal amount of buttons possible to do the functions, with the main Activity having buttons which link to other activities such as taking photo of receipt, adding food items, and viewing food information.

The user interface is set up so that the user can communicate with the app without knowing the actual layers existing beneath it. It has extensibility, allowing it to be changed in future updates without causing damage to other components of the system. Maintenance and minor updates on the user interface can be done easily.

Optical Character Recognition

This is achieved using the Google Mobile Vision OCR classes and library. This library provides two main functionalities.

The first functionality is recognizing text from a live view camera, which is accomplished in real time, on system. TextBlock objects are created, each representing a single word captured by the live view camera. These TextBlock objects can be manipulated to

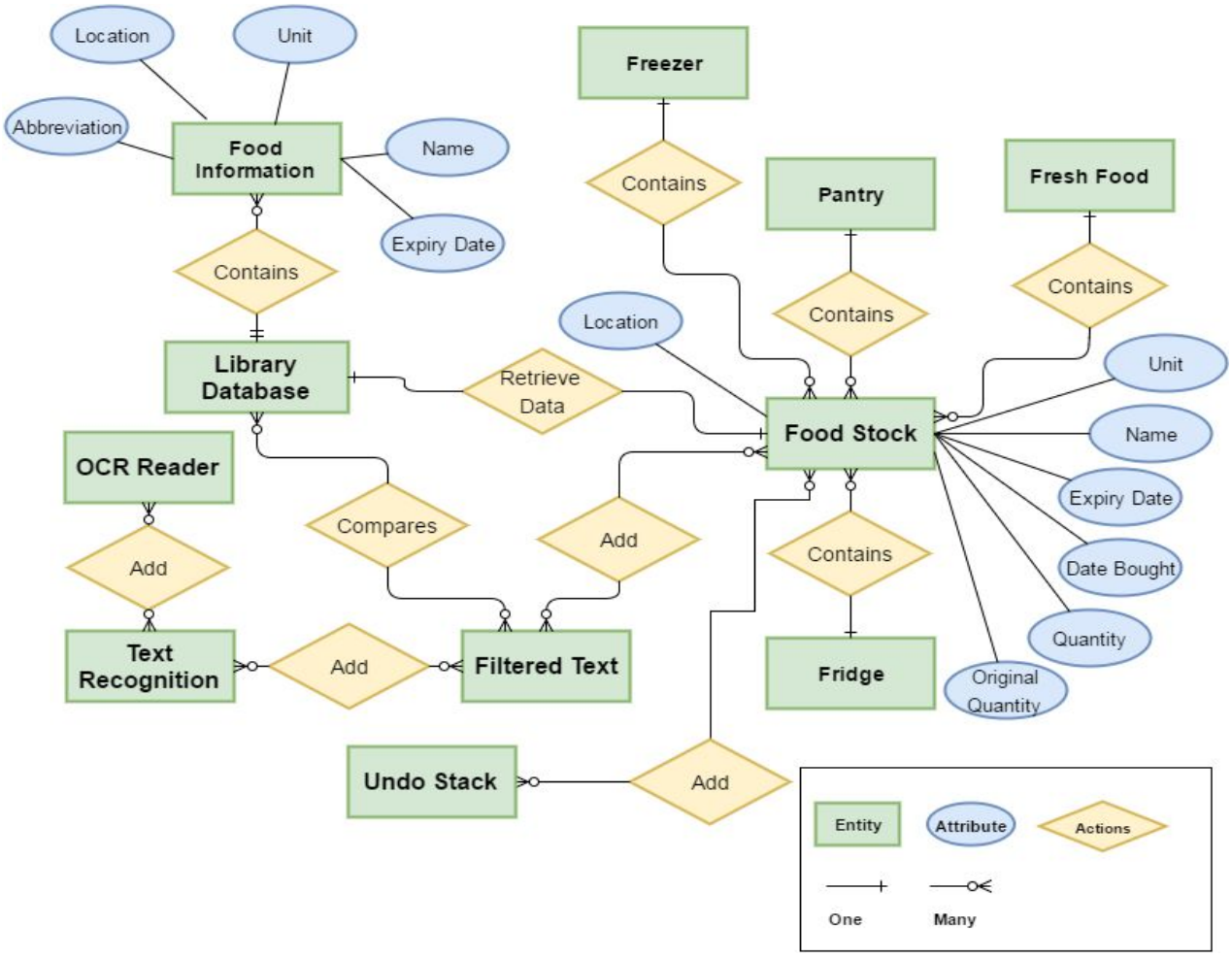
obtain the individual words as Text objects, and can be converted to Strings to be analyzed and matched with the database. This is the basic functionality required for text recognition.

The second functionality is recognizing the height, width, and position of individual words. This will be useful when we are analyzing and differentiating between the individual food items on the receipts, and help us sort and display the recognized items at the correct position.

We chose this API because it is fast. Character recognition happens in real time with the live view camera, and is almost instantaneous. It is combined with our string matching algorithm (Levenshtein Algorithm) to match items detected on the receipts with our library of food items, adding all items that are recognized.

Data

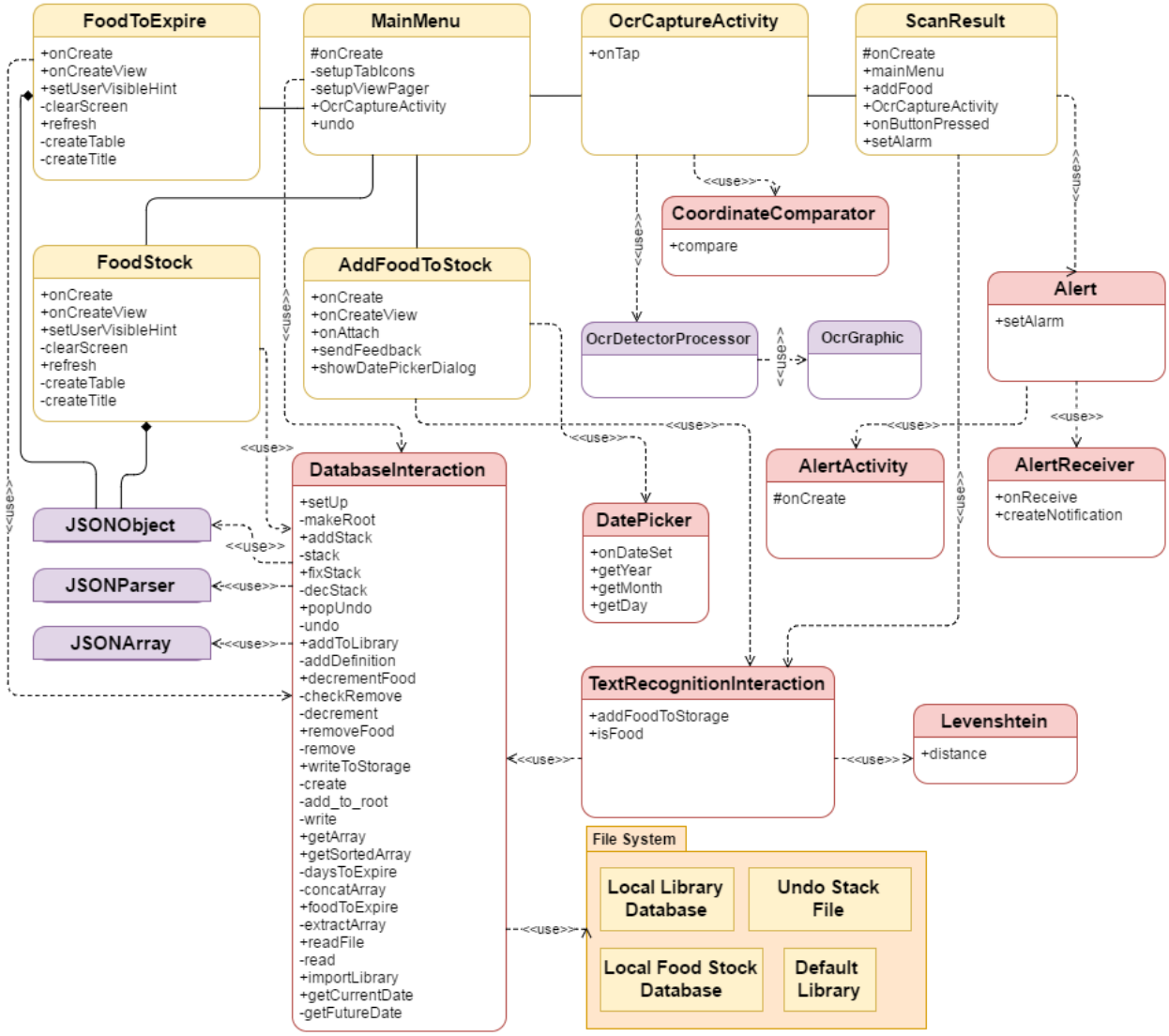
Fridge Manager's Entity Relationship Diagram looks as follows:



Detailed Design

Class Diagram

Fridge Manager's class diagram looks as follows:



GUI

Validation

One of the main focus of the Fridge Manager Application is incorporating a user-friendly interface. This means that user feedback is heavily taken into account in the UI design process. The design progress is presented to our team and to randomly selected testers on a bi-weekly basis. User feedback from these demonstration sessions will help guide our design so that the application would be tailored to user's needs.

Main Menu

From the main menu, you can reach all important pages with one click, these are: Food Stock, Scan Receipt, Expenditures, Food to Expire and Fresh Food.

Food Stock

On the food stock screen, you can see all products which are currently at home sorted by their location. Here, it is possible to manually add new food items and decrease them upon consumption. Once the food stock is populated, the food items are categorized as one of the following:

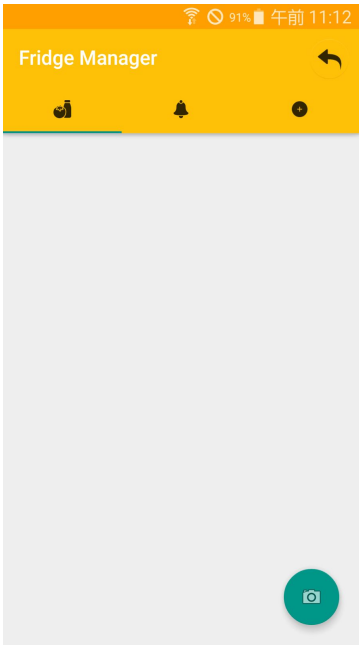
- Freezer
- Fridge
- Pantry
- Fresh (this includes any food without an expiry date)

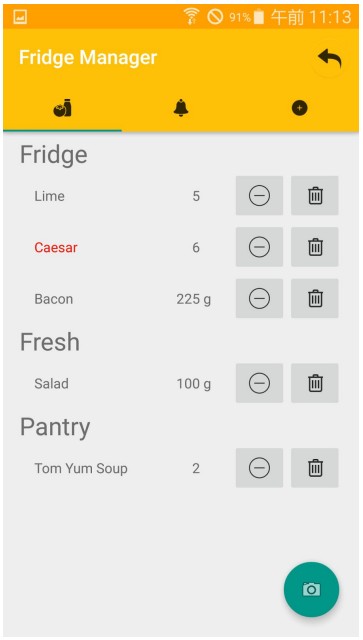
Recognized Food

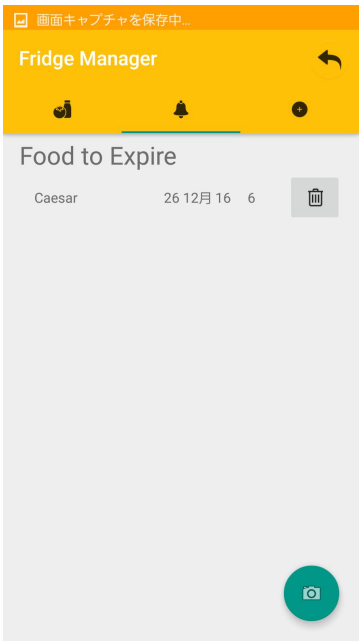
On this screen, the food recognized on the receipt is categorized into recognized food for elements which were found in the database and unrecognized food for products which weren't found. There are two possible reasons for why a product could not be found. Either, it is missing in the database (e.g. cherimoya) and the food has to added manually or the used acronym on the receipt (e.g. strwbr) is unknown. In this case, you only have to link the new acronym to an already existing product in the database.

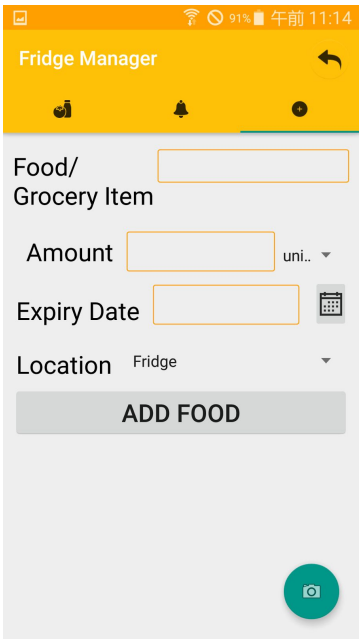
Add Food

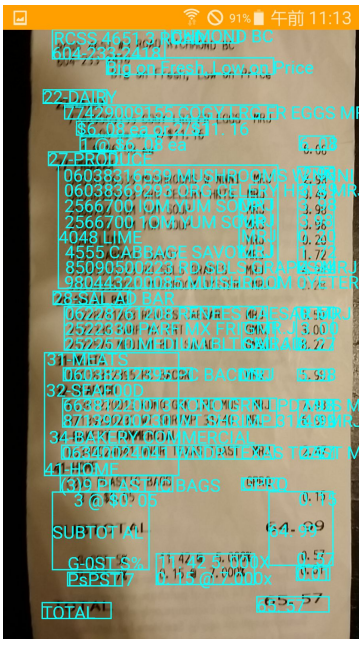
This screen is used to manually create a new entry to the food stock. With the drop down bar, you can select already existing items or you can type in an unused name to create a new entry in the database. Furthermore, you can specify the amount, the unit, the expiry date and where the product will be stored.

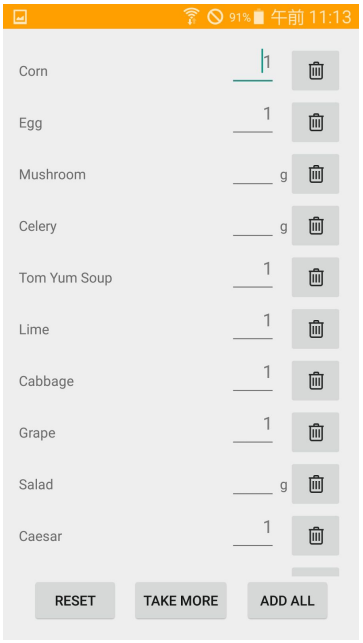

91% 午前 11:12
Fridge Manager


91% 午前 11:13
Fridge Manager
Fridge
Lime 5
Caesar 6
Bacon 225 g
Fresh
Salad 100 g
Pantry
Tom Yum Soup 2


画面キャプチャを保存中...
Fridge Manager
Food to Expire
Caesar 26 12月 16 6


91% 午前 11:14
Fridge Manager
Food/
Grocery Item
Amount
uni..
Expiry Date
Location
Fridge
ADD FOOD


91% 午前 11:13


91% 午前 11:13
Corn 1
Egg 1
Mushroom g
Celery g
Tom Yum Soup 1
Lime 1
Cabbage 1
Grape 1
Salad g
Caesar 1
RESET
TAKE MORE
ADD ALL

Design Validation

Procedure and Planning

Our aim was to create a GUI that focuses on minimizing user effort and enhancing the overall user experience.

Verification Process

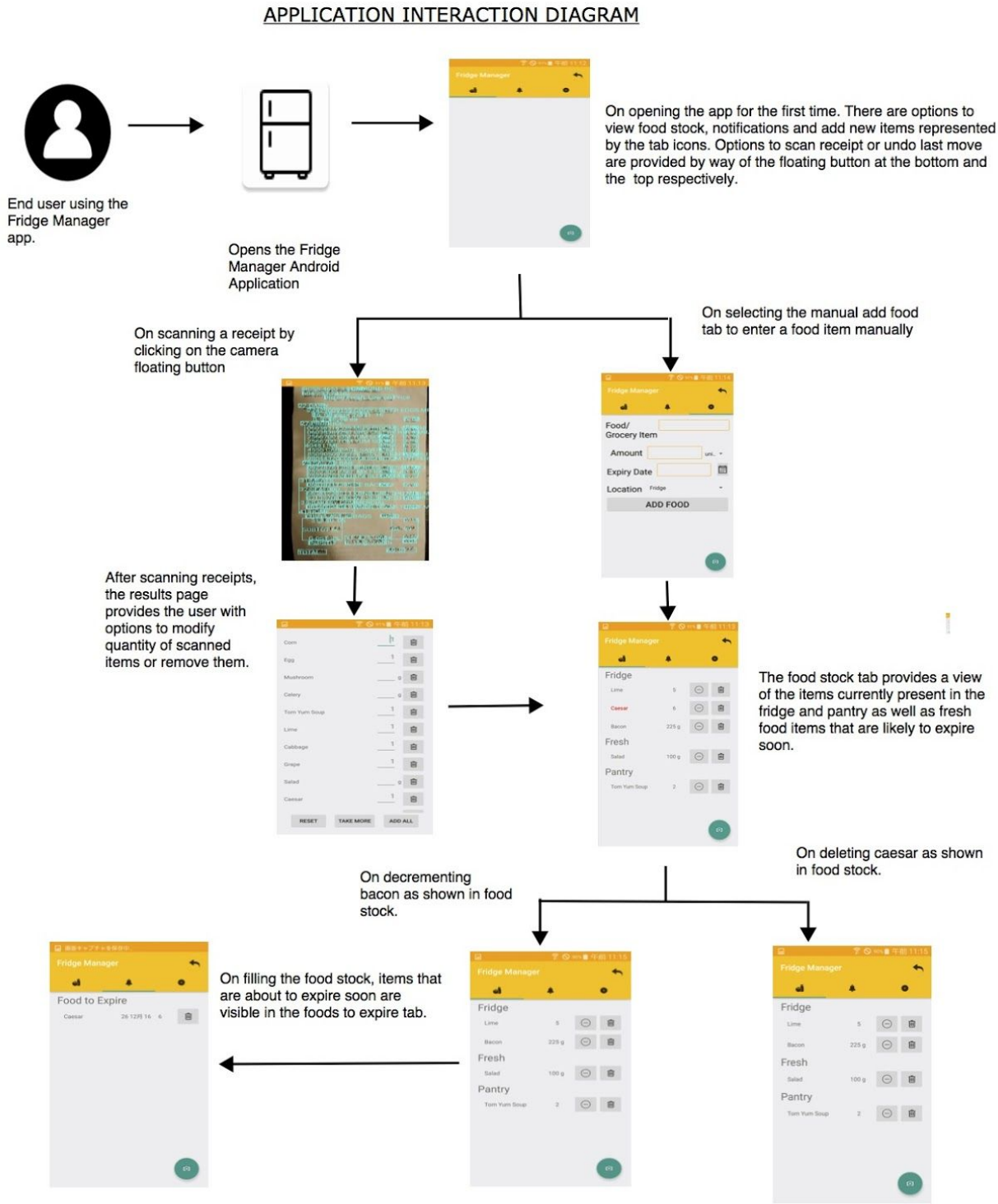
We verified our designs with repeated demonstrations to a group of target users and followed up on their observations by inspecting our designs accordingly. Initially, we had separate pages for every action with specific buttons leading from one page to another. Users at large expressed the opinion that this style was outdated. On inspection of the applications currently in the market, we improved our design layout by including a simple tab layout with the ability to swipe between the main fragments which includes the current foodstock, foods that are about to expire, and a manual food entry tab. Options to scan the receipts and undo a recent delete/decrement are included as floating buttons placed aesthetically in this tab layout.

Results

As of now, content awareness is maintained by means of icons used as metaphors for different use cases and actions. Expandable lists are used to display search results. Entry fields are surrounded by borders and the use of spinners and pop-up dialogs maintain clarity. Consistency of the interface is maintained by using a banner in the toolbar. Thus, the user is ensured a user-friendly experience and is provided by a straightforward interface to exploit full potential of all use cases.

Dynamic View

Fridge Manager's dynamic view looks as follows:



Developer-Specific Information

How to find your source code?

- All files under FridgeManager/app/src

How to check it out?

- git clone https://github.com/strebelr/FridgeManager

How to build the project?

- Be sure that Android Studio has already been installed on your computer
- Open the project in Android Studio
- Sync project with gradle file
- Run the application on an emulator or your Android device

How to run tests?

- Be sure your project is synchronized with Gradle by clicking Sync Project in the toolbar
- Run your test in one of the following ways:
    - To run a single test, open the Project window, and then right-click a test and click Run
    - To test all methods in a class, right-click a class or method in the test file and click Run
    - To run all tests in a directory, right-click on the directory and select Run tests

Structure of the source code directory?

- src/main/assets ----- Application/Library assets (Default Library Database)
- src/main/java ------- Application/Library sources
- src/main/res -------- Application/Library resources
- src/test/java ------- Unit Tests

Design patterns used?

- Structural Design Pattern
- Strategy Design Pattern