Fridge Manager

Design Document

Introduction

Fridge Manager is a mobile application designed to help people manage their fridges more conveniently and efficiently. Nowadays, people are busier and are lacking the time to manage their food. This application would alleviated 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. Local and Main Library Database
2. User Interface
3. Optical Character Recognition

Database: Javascript Object Notation (JSON)

The mobile application will require a local database to hold the list of items in fridge (or elsewhere) and a library database of possible food items. These databases will be implemented in a simple fashion using 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 app, such as the text recognition, require some memory space, it is preferable to limit others. In fact, our database will not be storing exponentially increasing amount of food, so the JSON format will be sufficient in our case. 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 and the average expiry time of that particular food. The only time this database would be changing is 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 Database

The local database holds all of the current food that the user have in their fridge or the 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 database). This database also holds the name of the food, when the food was bought, and along with the expiry date that is retrieved from the main database.

By having two databases, we are able to parse through the database more efficiently and “clever” designs are avoided. Minimizing the complexity means that we are parsing each database separately rather than developing a single algorithm to parse one single 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 database. Fundamentally, separating the local database and the main library database is more organized and makes it less confusing for the developers.

User Interface

The user interface is set up using separate Android Activities, 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

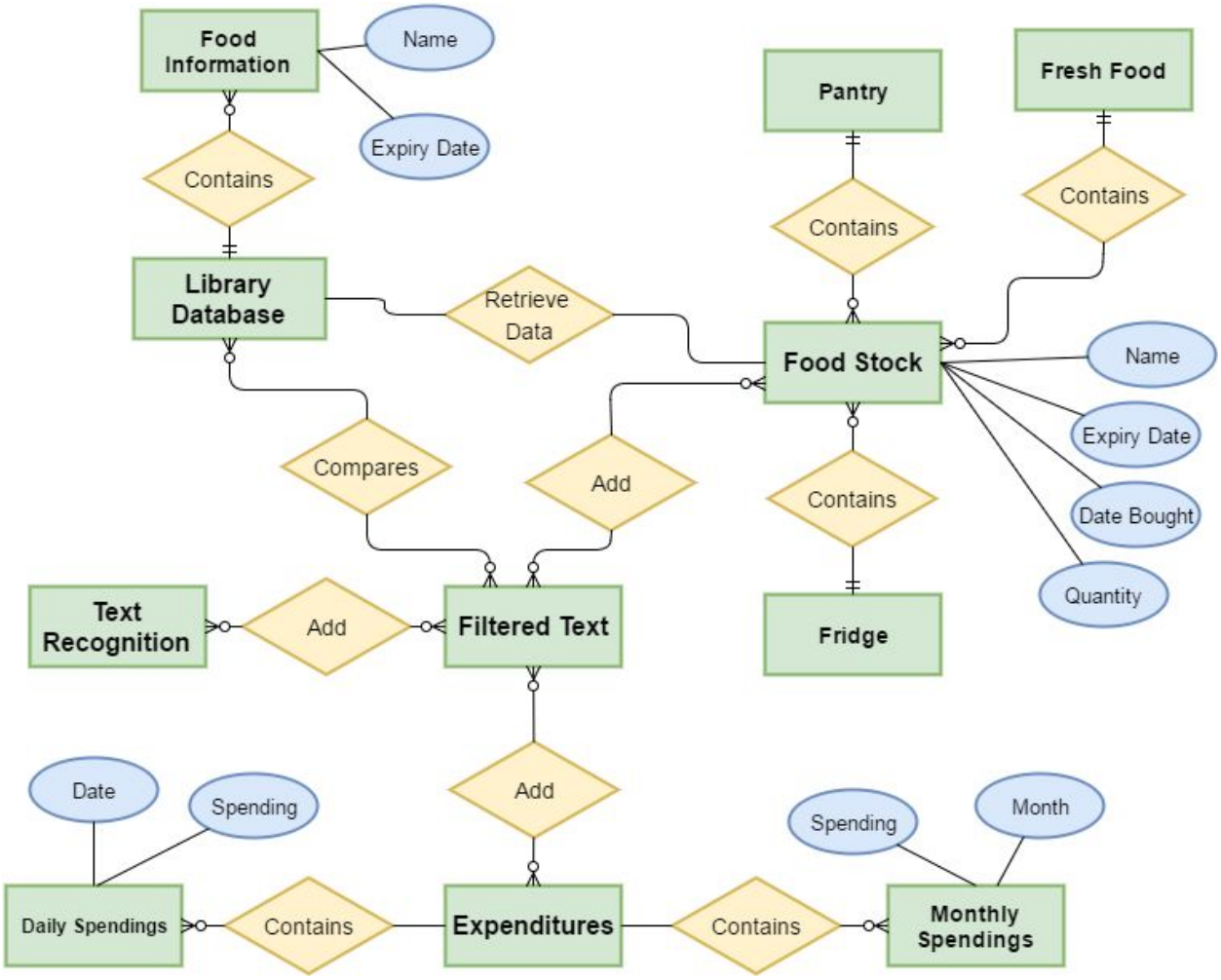
analyzed and matched with the database. This is the basic functionality required for text recognition.

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.

We chose this API because it is fast. Character recognition happens in real time with the live view camera, and is almost instantaneous. The sample classes can also be modified to analyze static images taken by the camera. It is simple and powerful and provides all the basic functionalities we need for character recognition.

Data

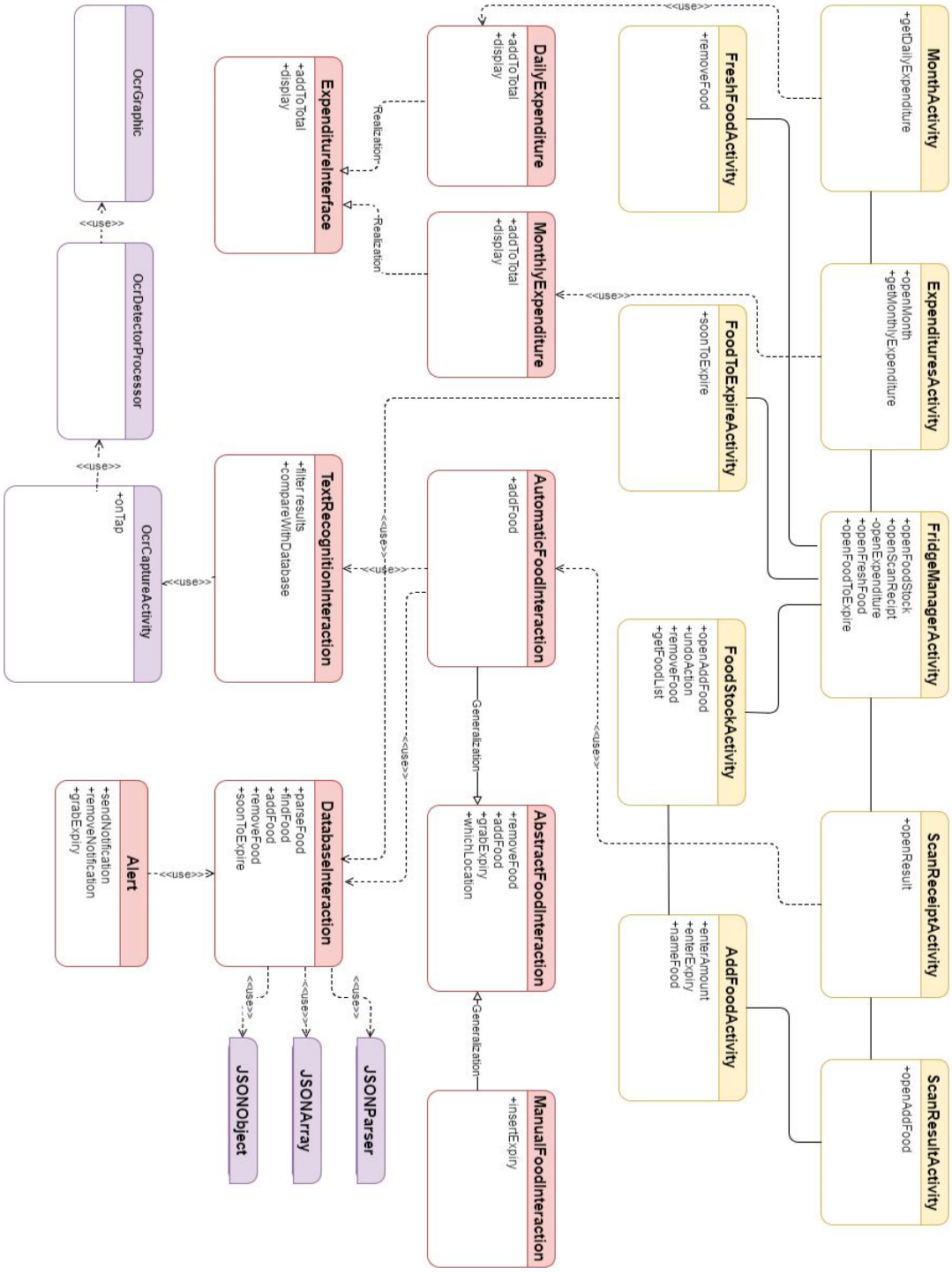
Fridge Manager's Entity Relationship Diagram looks as follows:



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.

Expenditures

Under the expenditures, the spendings of a full year are listed. By clicking on a specific month you end up on the detail list of the selected month (e.g. current month screen).

Fresh Food

This list includes the food which doesn't have an expiry date because it is considered as fresh food (ex. fruits or vegetables) and is displayed.

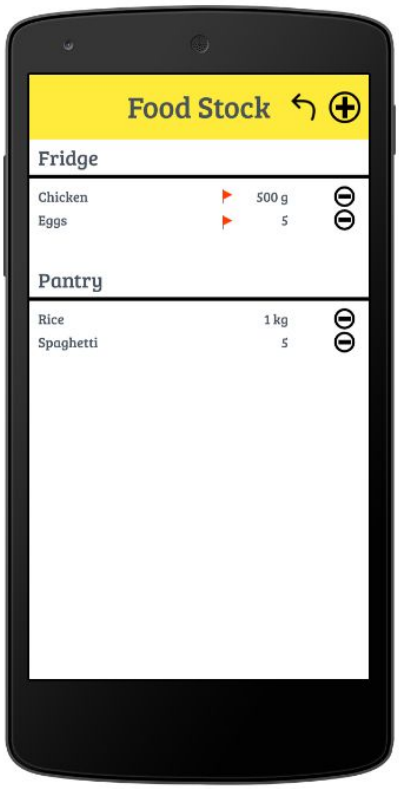
Recognized Food

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.

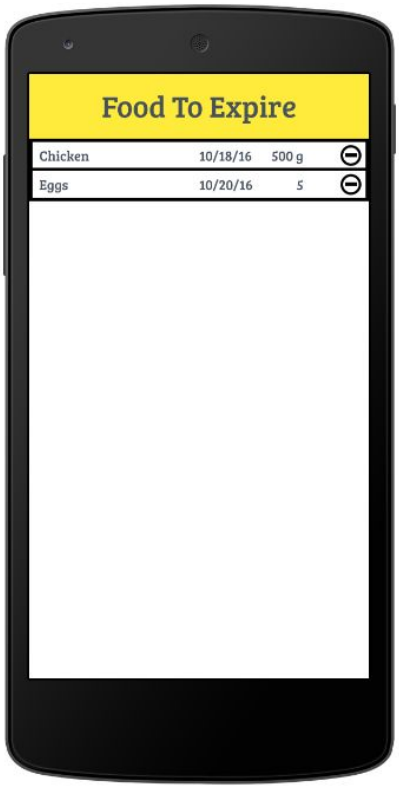
Add Food

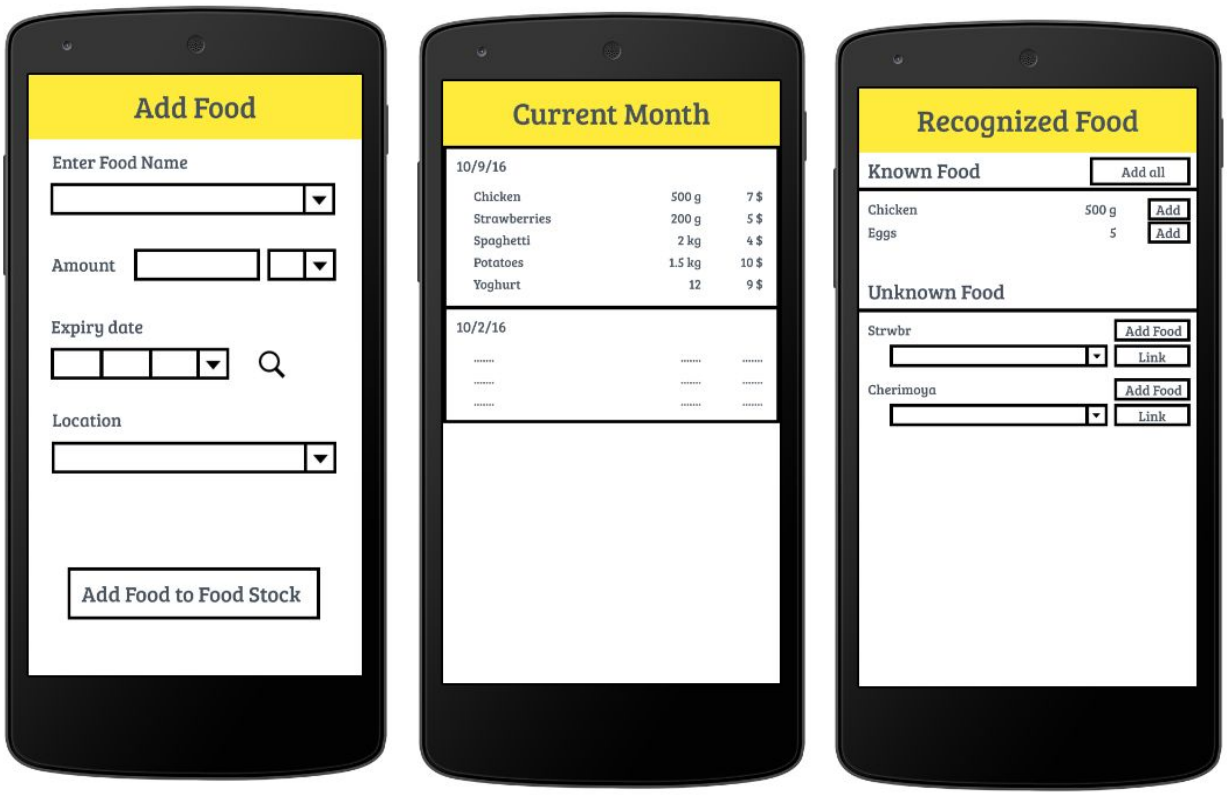
This screen is used to manually create a new entry to the food stock. With the drop down bar, you can select already existing items or you can type in an unused name to create a new entry in the database. Furthermore, you can specify the amount, the unit, the expiry date and where the product will be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

All screens are shown below and the following link is an interactive version of the GUI layout: https://marvelapp.com/2g7h7h0/screen/15894023



This Month	101$
September	120$
August	86$
July	113$
June	53$
May	179$
April	76$
March	98$
February	75$
January	154$
December	134$
November	111$
Total	1300$




Add Food
Enter Food Name
Amount
Expiry date
Location
Add Food to Food Stock
Current Month
10/9/16
Chicken 500 g 7 $
Strawberries 200 g 5 $
Spaghetti 2 kg 4 $
Potatoes 1.5 kg 10 $
Yoghurt 12 9 $
10/2/16
Recognized Food
Known Food
Add all
Chicken 500 g Add
Eggs 5 Add
Unknown Food
Strwbr
Add Food
Link
Cherimoya
Add Food
Link