Test Plan Document

Fridge Manager

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

---

Introduction

Fridge Manager is an Android based mobile application that allows users to take photos of their grocery bills and automatically store the food items into specific categories. Furthermore, it keeps track of the different food items and their expiry dates, aiding the user in managing their food in the fridge and notifies the user when the food are about to expire.

Verification Strategy

The application must reliably grab information from a receipt, store and display this information in such a way that the user can easily read and understand the information.

When testing the ability of the camera to grab information from the receipt, we will mostly be conducting tests using actual receipts from different grocery stores. Our current method of testing incorporates the Android Debug Bridge(ADB). Group members take photos of receipts from different grocery stores and the result is immediately displayed in another activity so that we can see what key words were successfully matched to our database of food items. For this strategy, we rely heavily on the ADB, which allows us to test any changes on an actual android device quickly and efficiently.

Further in the development process we plan to create an APK file so that users can install Fridge Manager onto their android devices. This way, we can have multiple testers that can provide feedback on the mobile application regarding their experience with the UI, and how useful the app is in general, as well as any suggestions.

Non-Functional Testing and Results

The non-functional requirements and corresponding tests of this application are as follows:

Performance requirements

- Finishes processing each photo and adding food items under 2 seconds:
- Any additional buttons (eg. Undo) should have an instantaneous-like (0.1s) response time

In order to achieve the above requirements, we would first test it within our group multiple times by recording the time spent in performing these actions. After we ensure that the average time spend is within the desirable range, we would ask other people (potential app users) to try these actions and rate how satisfied they are about the processing time. As for testing additional buttons, as long as their response feels instantaneous, the test is sufficient. This test will be done on different android devices and versions so that the requirement meets for various devices.

Software Quality Attributes:

- Ease of use over ease of learning
- Reliability, scalability, and recoverability due to database usage

Reliability of the database will be tested in accordance to its accuracy and data retrieval time. Because the library database will be containing all relevant information including expiry date regarding food item, the information should be correct. Under the assumption that the libary hold legitimate information regarding the food item, the only reasonable way to test accuracy is comparing the output of the local storage database to the library. In addition, as our library begins to populate, the data retrieval time should still be quick and any noticible lag will need to be fixed. Scalability can be tested by creating a script that automatically adds elements to our database. By adding numerous elements, we can check up to how large of a data the database can handle. To test recoverability of the database, we could intentionally make the app crash, and then check if database still retains information.

Security Requirement:

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

Security requirement can be tested by observing permissions that the application asks upon installation. Internet access is the largest threat to a secure application and since our current implementation does not require Internet access, we must observe that the app does not ask for Internet access permission. This test will be done to verify that our implementation in Android Manifest, which is in charge of setting up permissions, was done correctly.

Functional Testing Strategy

We will be performing unit testing because we want to ensure that the behaviours of an individual module is what we are expecting. Unit testing will be performed whevener a method has been implemented so that any additional code will not be influenced by the previous implementation. The unit testing will include both black-box testings (ex. database will require stubs before it is populated) and

white-box testings. Black box testing is required because the process will not be influenced by components being tested and will allow for testers without having to be familiar with the code. White box testing is also required to allow the exploration of alternate paths with knowledge of the implementation.

Once working classes are made, some integration tests will be performed to ensure that closely related units are working as expected. An integration test will be performed whenever a major component of the app is integrated with the rest of the app. For example, an integration test will be required for testing the communication between text recognition block and the database block, so that database block works with inputs from actual text source rather than mocked test cases. Another example would be testing whether the manual input field can be matched with the existing database, so that unrecognized food can be added by the user.

In addition, system testing will be performed because this helps ensure that the system as a whole is working properly. This means that any possible scenarios and requirements should be satisfied. A system test will be performed whenever a code source for scenarios or requirement is implemented. Performance is a key factor of our product so performance requirement should be satisfied through system testing. The UI will be tested differently and will require testers to play around with the mobile application and provide feedbacks on the user-friendliness of the app.

During these testing processes, it is inevitable for a bug to appear. These bugs might be simple so it may be fixed easily during the testing phase. However, for complex bugs, they must be handled with appropriate care using some bug-tracking mechanism. The complex bugs will be tracked using an Excel spreadsheet. The spreasheet contains the steps to reproduce the bug, its expected behavior and observed behavior, and a list of names to whom the bug fix is assigned to. Each bug in the list has a column to indicate if it has been fixed or not, then all developers know whether it has been fixed and if its implementation is ready to be used.

Adequacy Criterion

- Make sure that at least one test exists for each method written
- Make sure that at least one test exists for each requirement and for each use case
    - Make sure that at least one test exists for the camera to recognize texts of the grocery receipt regardless of the available lighting and the quality of the receipt
    - Make sure that at least one test exists for the camera to fully recognize a food database of a particular store
    - Make sure that at least one test exists for the camera to recognize food items that are abbreviated on a receipt

- Make sure that the mobile application (specifically UI) has been tested by a sufficient amount of testers

Test Cases and Results

Test	Requirement Purpose	Action/ Input	Expected Result	Actual Result	P/F
Test Cases for Character Recognition and Filtering Algorithm Depending on Different Stores	Identify all food items from names and its possible abbreviation	Click camera floating button, tap anywhere on live view	Scan Result screen should display all food items on the receipt including those that are abbreviated	Food items are added if they are written in actual names, but not added if they are abbreviated	F
Test Cases for Character Recognition and Database Interaction	Grab all textblock objects on tap and filter, adding to food stock	Click camera floating button, tap anywhere on live view, tap add button	All food items from receipt should first appear in Scan Results page when tapping camera live view, then when the user taps add button, all food items should be added to food stock	Basic food names that are already in the database are recognized and added. Unfamiliar words not in the database are not recognized and added.	F
Test cases for interaction between different activities and use cases (Integration testing)	Test the flow of an entire action such as adding to food stock using the camera or receiving notifications and altering the current food stock based on the foods that are about to expire	Specify the action to be carried out	Observe and record the flow and interaction between the pages, for example in order to test the action of adding to the food stock, test whether the scanReceipts page is followed by the IdentifiedFoods page on successful scanning of reciepts and is further followed by the form to add unknown foods detected from	The scanReceipts page is followed by the IdentifiedFoods page after succesfully scaning the receipts. However, there are more action flows to be tested.	-

			the receipt		
Unit tests for filling forms	Test successful submission of forms and entering of data into the database	Fill a form for manually adding an item to the foodStock and submit	Successful submission of form with no errors should enter the form contents to the database and any errors in the form should result in an error message	The submission of the form has yet to be implemented.	-
Recoverability	Test how the system responds to a crash	Throw an error in order to shut down the app in the midst of an action or use case	Successfully recover the lost information from the database, and continue on the action	To be implemented.	-
Scalability	Test how the system responds to increase in data to be entered or scaling of the database	Test the case for a receipt with the maximum number of items	Successfully populate the database with every item and display all items correctly on the foodStock page	To be implemented.	-