

SMART REFRIGERATOR REPORT

May 1, 2012

This project has developd a prototype “Smart Refrigerator” system, which monitors grocery items purchased by the user in order to reduce food waste and facilitate efficient shopping habits.

Steven Strapp - ses6498@rit.edu

Ben Reeves - bpr5171@rit.edu

Dustin Stroup - dxs2857@rit.edu

Contents

1	Overview	3
1.1	Needs Statement	3
1.2	Objective Statement	3
1.3	Description	3
2	Requirements Specification	4
2.1	Customer Needs	4
2.2	Engineering Specifications	5
3	Concept Selection	6
3.1	Evaluation of Existing Systems	6
3.2	Concepts Considered and Chosen	6
4	Design	9
4.1	Base Station Code Package	11
4.2	Mobile User Interface	13
4.3	Web Interface	16
4.4	Database Definitions	17
4.5	Other Design Considerations	17
5	Considerations	18
6	Cost Estimates	19
7	Testing Strategy	19
7.1	Base Station User Interface Testing	20
7.2	Mobile User Interface and Network Interface Testing	21
7.3	Shopping List and Expiration Prediction Test	25
7.4	Integration with BeagleBoard	25
8	Risks	26
9	Milestones	28
9.1	Gantt Chart	29
10	Perspective	30
10.1	Base Station Code Package	30
10.2	Web Interface	30
10.3	Mobile Interface	30
10.4	BeagleBoard Integration	31
11	References	32

1 Overview

1.1 Needs Statement

The New York Times reports that an average American family of four will account for over 120 pounds of food waste per month and that 27% percent of all food available will be lost to waste [1]. In addition, other resources are lost due to inefficient shopping practices; forgetting common items or special trips made for recipe ingredients waste time and fuel. A system is required for shoppers both to ensure their purchases are used before expiration and to assist in planning of grocery shopping trips.

1.2 Objective Statement

The objective of this project was to design a prototype that would allow a user to track food items in order to reduce waste and improve shopping efficiency. The system reminds the user about items nearing their expiration date and tracks the frequency of purchased items. From this frequency calculation the system suggests typical shopping lists. A mobile phone application and web page provide interfaces to the current inventory of the unit and to shopping lists while on the go.

1.3 Description

A UPC scanner is used to identify items added or removed from the refrigerator's inventory, and a database of UPC codes is used to translate from the scanned code to an item description. Two product databases are maintained: one linking UPC codes to product descriptions and another to store items currently checked into the refrigerator and their expiration dates. A Beagleboard-xM functions as the central processing platform on the base station and is used to decode UPC information and to interact with the databases. The Beagleboard hosts a web interface accessible via a web browser or an Android mobile device. A display on the main unit allows users to review the current inventory, with expiration dates, and provides additional information when adding or removing items. The base station, web and mobile interfaces can also be used to display the current inventory and suggested shopping lists. The mobile application interacts with the same web interface as the web browser based interface but provides a graphical interface optimized for smaller displays. The system continually monitors the purchasing frequency for items and uses this information, combined with the expiration and purchase dates, to suggest shopping lists. In addition to shelf life, temperature is also a critical factor for food storage systems. To address this need, the system incorporates a temperature sensor and the temperature information is accessible through the base station application.

A high level system diagram which partitions major components is shown in Figure 1.

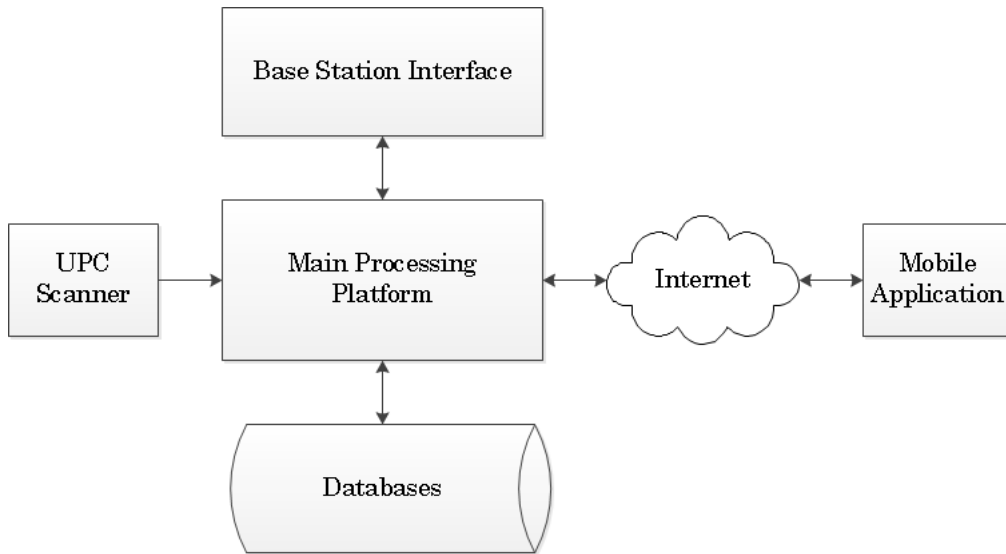


Figure 1: High Level System Diagram

2 Requirements Specification

2.1 Customer Needs

1. The system should provide an intuitive and easy to use graphical interface.
2. The system should require minimal user input.
3. The system should be able to scan product codes and identify corresponding items quickly.
4. The system should provide secure remote access.
5. The system should report items nearing expiration.
6. The system should provide access to the current inventory.
7. The system should provide a method to create and edit shopping lists.
8. The system should recommend shopping lists which accurately reflect buying habits.
9. The system should function as an add-on to an existing refrigerator or pantry.
10. The system should indicate if food products are stored at an unsafe temperature.

2.2 Engineering Specifications

Customer Need	Engineering Requirement	Justification
2,3	A. An off-the-shelf UPC scanner should be used to input items.	A UPC scanner can read product codes with a single click.
3	B. An internal UPC code database should be used to associate codes with items.	An internal database will remove delays associated with an Internet look-up.
1,4,6	C. The system should be Internet enabled and provide a web interface.	By providing a web interface any other Internet-connected device can access the system.
4	D. Remote access should be authenticated with user name and password.	User names and passwords are standard for access control.
2,5	E. An internal database will store default recommended expiration estimates for common categories of items.	Inferring expiration dates based on item category helps minimizes user input. It is well known how long some products take to expire.
1,5	F. The user interface will provide a method for updating default expiration estimates.	Default estimates will not account for the condition of product on arrival and may need to be updated.
1,5	G. Interface will provide a visual indication to the user which items are closest to expiring.	The goal of the system is to reduce waste due to expiration.
1,6	H. From the base station, web and mobile interfaces the user will be able to view an inventory list.	The user needs access to the current inventory in order to use items and shop effectively.
7,8	I. A database will be devoted to storing recommended shopping lists produced by the system.	User may wish to retain generic shopping lists for future use.
8	J. Recommended shopping lists will reflect purchasing history and expiration dates of current inventory.	Recommendation policy must suggest items relevant to the user in order to be useful.
7	K. Custom shopping lists, created from the base station, can be added to shopping list database.	Inefficient shopping practices can be prevented by storing shopping lists and the system can not anticipate all required items.
9	L. The system will be self-contained and no modifications will be required to existing appliances.	Similar systems are commercially available but require costly replacement of existing appliances.
10	M. The system should measure temperature within the refrigerator.	Temperature measurements will allow the user to quantitatively determine if food storage conditions are safe.

3 Concept Selection

3.1 Evaluation of Existing Systems

Many refrigerator systems are currently available that offer integrated displays and internet connectivity. LG, Electrolux, and Samsung all offer refrigerators with large LCD displays that provide access to calendar applications, recipes, weather forecasts, and music and photo sharing services. The principle shortcoming of these devices is the elevated price and the need to completely replace existing appliances. As a more affordable alternative, tablet mounts are available for refrigerators as well. However, these systems do not offer tracking of the refrigerator's contents and do not attempt to reduce waste or improve efficiency. In April of 2011, LG demonstrated a "Smart Fridge" with goals closer to the proposed system. The sensors and algorithms used were not disclosed but the product objective is similar: tracking user purchases and providing a mobile interface to the refrigerator's contents while shopping [2]. Our system will provide a much more inexpensive alternative and will be more flexible; the system proposed will not be strictly limited to refrigerators and can be used as an add-on to an existing system.

Many patents exist on inventions related to the smart refrigerator system as a whole and its goal to reduce waste, but do not attempt to reduce user input. Patents 2004/0085225 A1 *Methods and Apparatus to Monitor the Inventory of a Food Storage Unit*, 2010/0148958 A1 *Expiration Warning Device of Refrigerator*, and 2011/0109453 A1 *Apparatus for Warning of an Expiration Date* all treat the goals of the overall system but rely on the user to enter expiration dates manually. More advanced systems, as in Patents 7,861,542 B2 *Refrigerator Including Food Product Management System* and 2011/016555 A1 *Refrigerator and Control Method Thereof*, use radio frequency identification (RFID) tags attached to foods to read expiration dates, with user input as a fallback. The prototype designed improves upon the simple user-intensive method of the first group but without the added scope of radio frequency identification used in the second group.

3.2 Concepts Considered and Chosen

Many of the system design choices were easily derived from the engineering requirements; a UPC scanner with a standard USB interface was a clear choice for input of product codes and a mobile application was an obvious interface choice for a system catering to an on-the-go shopper. However, the choices of implementation platform and main base station display presented more alternatives. Expiration date recognition was also a potential shortcoming of the system; ideally image processing would have been employed to read expiration dates. However, the difficulty and computational complexity of applying image processing would significantly extend the scope of the project and would place additional performance constraints on the processing platform used. An evaluation of different expiration date recognition systems is tabulated in Table 1. The different evaluation criteria, ease of use, feasibility and accuracy, are at odds, and each criterion was given equal weight during concept selection.

Ease of use was one of the most critical system requirements; a system relying completely on input from the user would not be acceptable to consumers. However, feasibility and limiting processing performance required were important secondary objectives. Accuracy was also critical to the goal of reducing waste due to expiration, but there is inherently some variability even in reported expiration dates. Image processing presented too much additional scope and too many additional requirements in exchange for marginal gains. A predictive system that learns

Table 1: Comparison of Expiration Date Systems

	Method			
	User Input of expiration dates	Image to Text Recognition	Predictive Strategy without itemMaster	Predictive Strategy with itemMaster
Ease of Use	- - -	+	+ + +	+ + +
Feasibility	+ + +	- - -	- - -	+ + +
Accuracy	+ +	+ +	+	+
Total	2+	0	4+	7+

from user input and anticipates that items will be purchased in different conditions, appeared to be sufficient. One additional risk posed by the predictive system was the problem of deciphering text descriptions in order to assign an appropriate prediction. That risk was mitigated by using the ItemMaster UPC database. Many websites, such as the Food and Drug Administration or community based resources like www.stilltasty.com, provide “rule of thumb” predictions for expiration dates. However, the system must associate a product description with a rule of thumb, which, after investigation, appeared to be a difficult classification problem. The ItemMaster UPC database provides not only an association between a UPC code and a text description but also provides a GS1 category. There are a modest number of GS1 categories applicable to this system, each of which was assigned a rule of thumb to initialize the prediction system.

The problem of predicting shopping habits was formulated as a problem of predicting the probability that the user will purchase a product again after N days from the last purchase. Products are be added to the shopping suggestions at the peaks in the probability density function, after which the process resets. To evaluate modeling strategies, receipts were retrieved for a three month interval from a single user. An initial attempt was to assume that the large number of factors influencing shopping habits could be approximated as normally distributed. However, for the data tested, this approximation was very poor; the data considered were either multi-modal or contained a single mode with outliers. In all cases considered, the distribution was shifted to the point where the most likely suggestion time was actually positioned in an interval not supported by any of the samples. A more advanced approach, a non-parametric distribution estimate, was considered next; this method outperformed the simple normal approximation, but appeared to interpolate more than necessary and appeared to be the most computationally complex method considered. However, SciPy, a Python scientific computing library provides non-parametric distribution evaluation functions which are optimized and simple to interface with. Database storage is another area where a non-parametric method excelled; all other methods would required some parameter, or possibly a variable number of parameters, be stored in the database or be recalculated every prediction.

A final approach clustered the data points, approximated each cluster with a normal distribution, and summed these distributions. With this strategy, each mode could be captured without the influence of outliers. The accuracy of the methods considered were evaluated both qualitatively, by looking at the resulting probability density functions, and also quantitatively, by considering performance on the example sets. Overall, clustering to produce a sum of Gaus-

sians appeared slightly more accurate but was the most difficult to store in a database. The probability metrics used are tabulated in Table 2. The goals of this subsystem, maximizing the probability of accurate recommendations, minimizing the probability of unsupported recommendations, ease of computation, and the ability to be stored in a database, were given equal weight during concept selection. Despite its slightly worse performance, the non-parametric distribution estimate was chosen since it required no storage in the database and was provided in the Python scientific computing library.

Table 2: Comparison of Distribution Estimate Performance Metrics

	Trial	Method		
		Normal Approximation	Non-Parametric Distribution	Clustering to produce sum of Gaussians
\sum Log Probability Observed Habits (Goal to Maximize)	1	-38.3394	-35.9682	-34.7721
	2	-20.5647	-17.0897	-15.6641
	3	-47.8101	-44.9658	-43.9845
	4	-29.1931	-19.6762	-24.4915
Evaluation		- - -	+	+ + +
\sum Log Probability Habits Not Observed (Goal to Minimize)	1	-36.7898	-38.4187	-50.6578
	2	-188.514	-225.002	-318.926
	3	-62.2909	-63.8609	-69.9759
	4	-29.6667	$-\infty$	-86.0767
Evaluation		- - -	+	+ +
Ease of Computation		+ + +	-	- -
Ability to Store in Database		+	+ + +	- - -
Total		2-	4+	0

The choice of a base station main display and processing platform were linked, but dictated mainly by the processing platform. For example, if a personal computer were used a standard LCD monitor may be appropriate, whereas if a tablet were chosen as the main processing engine the interface would be provided automatically. The most strongly considered option was to use a simple micro-controller or BeagleBoard to handle the processing load and to use a modest sized LCD display. Comparisons of different processing platform methods and different user interface choices for the base station are shown in Tables 3 and 4, respectively. The evaluation criteria for the processing platform and user interface were given equal weight; though since the processing platform and user interface concepts were related, the criteria of the processing platform were given higher priority than the user interface.

Evaluating both the interface choice and the processing platform choice together eliminated the personal computer; a personal computer could not be integrated without significantly increasing the form factor of the system. A personal computer also greatly simplifies the system design and strays away from an implementation tailored to this prototype. A tablet based interface was considered a very feasible alternative; however the cost and tailorability of the system were again concerns. A micro-controller based system was more appropriate for a small and specialized solution, with the principle concern being quality of the graphical interface produced compared

Smart Refrigerator Proposal

Table 3: Comparison of Main Processing Platforms

	Method			
	Personal Computer	Tablet (Combined UI and Processing)	Micro-controller	Beagleboard-xM
Processing Resources	+ + + +	+ +	+	+ + +
Cost	- - -	+	+ + +	+ + +
Size	- - -	+ +	+ + +	+ + +
Total	2-	5+	7+	9+

Table 4: Comparison of Main User Interface Displays

	Method		
	LCD PC Monitor	Tablet	LCD with BeagleBoard-xM
Integration with Unit	- - -	-	+ + +
Ease of Use	+ + +	+ + +	+ +
Size of Display	+ + +	+ + +	+ +
GUI Quality	+ + +	+ + +	+ + +
Size of Unit	- - -	+ + +	+ + +
Total	3+	12+	13+

with the other two methods. Considering both choices together, the BeagleBoard-xM with an LCD display to inspect items visually as they are checked in and to view inventory appeared preferable. A BeagleBoard, running a full Linux environment, maintained a small form factor while still providing a high quality interface and sufficient processing resources.

4 Design

Consideration of these concepts, as well as the high level system diagram presented in Figure 1, clarified the separation of tasks while implementing the project. One group of tasks contained the mobile interface and also development of an interface control specification to enumerate the commands provided over the web interface. A second task group consisted of configuring the internal databases on the BeagleBoard, the expiration date warning system, and the shopping list suggestion algorithm. The development of the base station user interface and web interface was also included in this task group. The final group of tasks consisted of interfacing the BeagleBoard with the UPC scanner, Ethernet interface, temperature sensor, and main user interface. The partitioning of project work is also evident in the full system diagram shown in Figure 2.

The majority of software developed runs on the BeagleBoard, which functions as the main processing platform for the Smart Refrigerator system. The BeagleBoard runs a custom build of the Angstrom operating system, a lightweight embedded Linux. A full Linux environment is very conducive for software development and greatly simplifies connecting with peripherals; the keypad or keyboard, as well as the UPC scanner, were able to simply “plug-and-play.” The temperature sensor required more effort, particularly to ensure the input and output voltage levels met the specifications of the sensor and did not damage the BeagleBoard. The Digilent

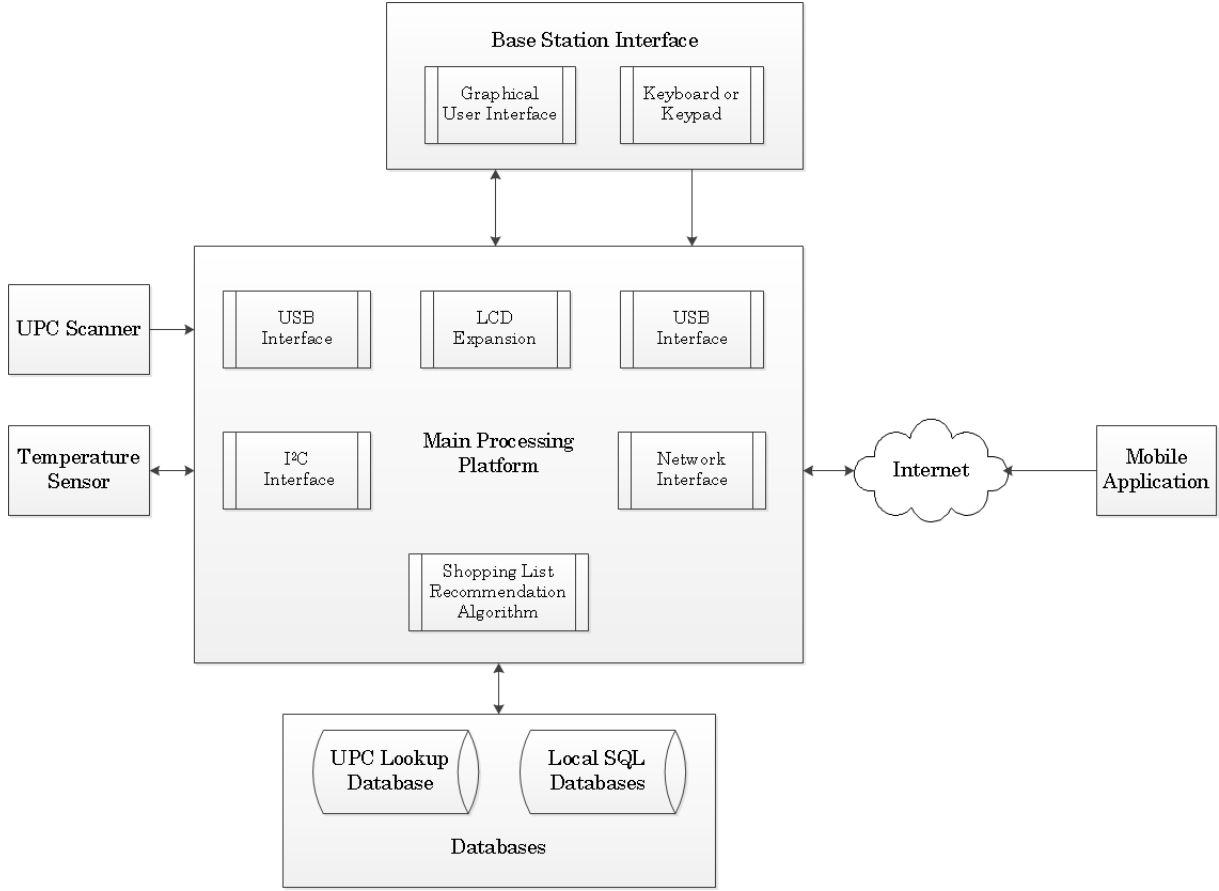


Figure 2: Full System Diagram

Pmod TMP2 temperature sensor that was used requires a 3.3V or 5V power supply and will output voltages as high as the supply voltage. The BeagleBoard's I²C interface supplies and receives voltages up to 1.8V only. To resolve this discrepancy, a BOB08745 Logic Level Converter was incorporated to interface between the BeagleBoard and the temperature sensor.

A more detailed diagram of the BeagleBoard subsystem is shown in Figure 3. The figure shows all external connections to the BeagleBoard as well as an internal separation of sub-components. The base station code package was inspired by the Model, View, Controller paradigm. A distinct module of code creates the user interface displayed on the BeagleBoard's touchpad and relays user interface events to a separate controller module. The controller sub-component coordinates the various input events generated by the system. Input from the UPC scanner and keyboard pass through this module and then are transferred to the view. The controller is a necessary middleman in this process, since scanned UPC codes and user inputs must be shared with the model as well. The controller is also responsible for handling input from the temperature sensor; this interaction occurs through a dedicated I²C driver. The controller also interacts with database structures and handles events from the network interface. The model itself is distributed among the remaining modules; the content needed by the model is stored within the databases, and the principal modeling tasks occur within the expiration date and

shopping list prediction sub-module.

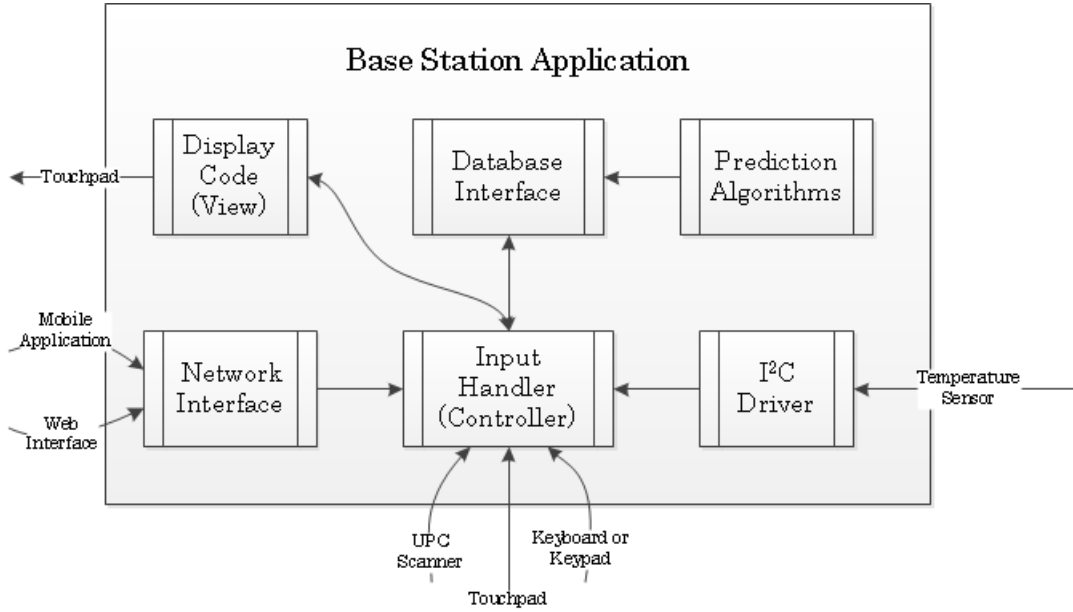
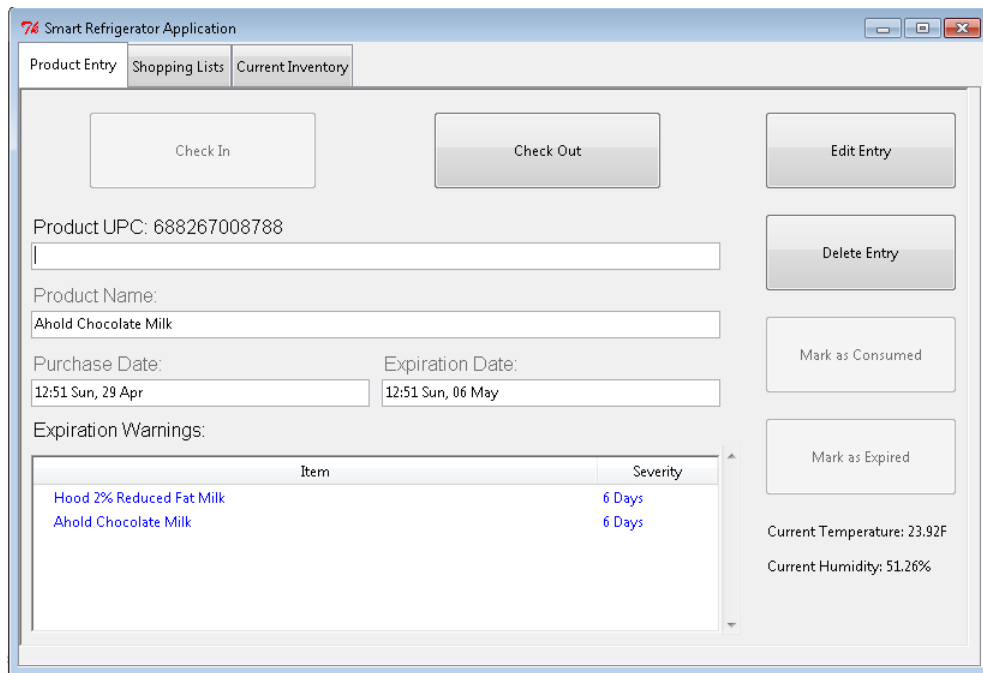


Figure 3: BeagleBoard Subsystems

4.1 Base Station Code Package

The base station application was developed in Python using the TkInter user interface framework. Java was also given consideration as the primary language, since this could potentially increase consistency with the mobile application. However, Python is notoriously quick to develop with and was able to prototype the user interface rapidly. Regardless of the implementation language, effort was made to maintain consistency between the two user interfaces. Screen captures of the base station graphical user interface are shown in Figures 4, 5, 6. When designing the interface layouts, the constraints of a touch screen interface were considered; all buttons and tabs are intentionally large and easy to click. The product entry tab, shown in Figure 4, is the default tab, and provides feedback to the user while scanning items. The check in and check out buttons function as radio buttons to indicate whether the next scanned item will be interpreted as a new purchase or an item being removed from the current inventory. Since this tab is also the default, a list of items closest to expiring is shown at the bottom to remind the user. The shopping list tab provides a straight-forward view of past shopping lists, organized by descending creation dates. Clicking the suggested list button will produce a new recommended shopping list. The current inventory tab simply lists items currently checked into the refrigerator and provides a reset function to clear the current inventory.

Smart Refrigerator Proposal

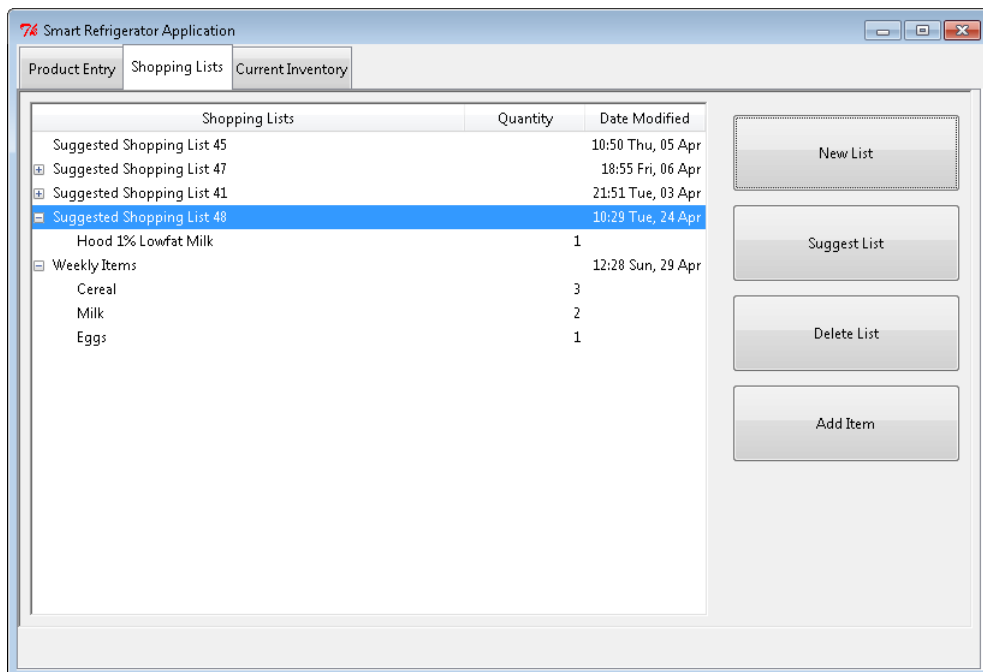


The Product Entry tab layout includes buttons for 'Check In', 'Check Out', 'Edit Entry', 'Delete Entry', 'Mark as Consumed', and 'Mark as Expired'. It features input fields for 'Product UPC' (688267008788), 'Product Name' (Ahold Chocolate Milk), 'Purchase Date' (12:51 Sun, 29 Apr), and 'Expiration Date' (12:51 Sun, 06 May). An 'Expiration Warnings' table lists items and their remaining days.

Item	Severity
Hood 2% Reduced Fat Milk	6 Days
Ahold Chocolate Milk	6 Days

Current Temperature: 23.92F
Current Humidity: 51.26%

Figure 4: Product Entry Tab Layout



The Shopping Lists tab layout displays a table of shopping lists with columns for 'Shopping Lists', 'Quantity', and 'Date Modified'. It includes buttons for 'New List', 'Suggest List', 'Delete List', and 'Add Item'.

Shopping Lists	Quantity	Date Modified
Suggested Shopping List 45		10:50 Thu, 05 Apr
⊕ Suggested Shopping List 47		18:55 Fri, 06 Apr
⊕ Suggested Shopping List 41		21:51 Tue, 03 Apr
⊕ Suggested Shopping List 48		10:29 Tue, 24 Apr
Hood 1% Lowfat Milk	1	
⊖ Weekly Items		12:28 Sun, 29 Apr
Cereal	3	
Milk	2	
Eggs	1	

Figure 5: Shopping List Tab Layout

Items	Item Quantity	Purchase Date	Expiration Date
<input checked="" type="checkbox"/> Hood Fat Free Milk	1		
<input checked="" type="checkbox"/> Hood 1% Lowfat Milk	3		
Hood 1% Lowfat Milk		10:52 Thu, 05 Apr	10:52 Thu, 12 Apr
Hood 1% Lowfat Milk		18:55 Fri, 06 Apr	18:55 Fri, 13 Apr
Hood 1% Lowfat Milk		21:53 Sat, 07 Apr	21:53 Sat, 14 Apr
<input checked="" type="checkbox"/> Kellogg's FiberPlus Antioxidants Berry Yogurt Crunch Cereal	1		
<input checked="" type="checkbox"/> 2% Chocolate Milk 3pk	1		
<input checked="" type="checkbox"/> Kellogg's Special K 90 Calories Honey Nut Cereal Bars	1		

Clear Inventory

Figure 6: Current Inventory Tab Layout

4.2 Mobile User Interface

The mobile interface is divided into two similar tabs, one to view the current inventory and another to view shopping lists. Screen captures for the two mobile interface tabs are shown in Figures 7, and 8. Clicking an item in the inventory tab displays a list of all purchase dates for that item, as shown in Figure 9. Similarly, on the shopping list tab clicking an item displays a list of all items on that shopping list, as shown in Figure 10. The mobile application is updated on a “need to know” basis only. If additional items have been added to the inventory or new shopping lists have been created, the mobile application is not notified until the user has opened the application and navigated to the appropriate tab. It was anticipated that the largest data exchanges occurred when loading the shopping lists, since they require information from multiple items and additional top-level information about the list. The mobile application further improved data efficiency by not loading shopping list items until a list has been selected. Navigation to the shopping list tab generates a query for the top-level information about new or updated shopping lists, but does not delve into the lists themselves to retrieve item information. Only once a list has been selected complete updated information about its items is retrieved. This implementation creates additional latency while using the application; faster strategies included using a single large update of all data upon launching the application or a background fetch of information while running the application. However, given the high costs of mobile data plans and the lack of widespread WiFi at most grocery stores, limiting the amount of data passed across the network interface is an important consideration for usability.

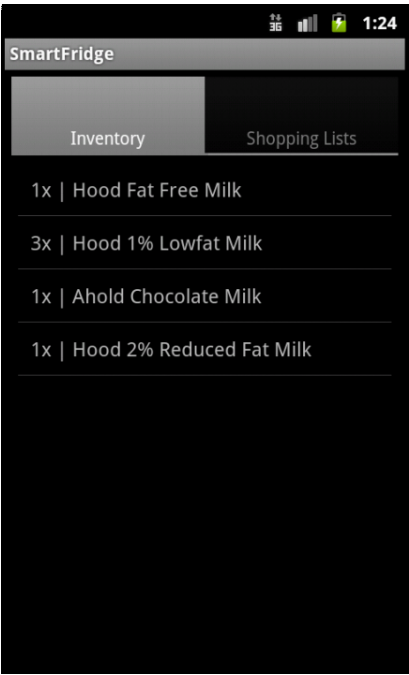


Figure 7: Mobile Interface Current Inventory Tab

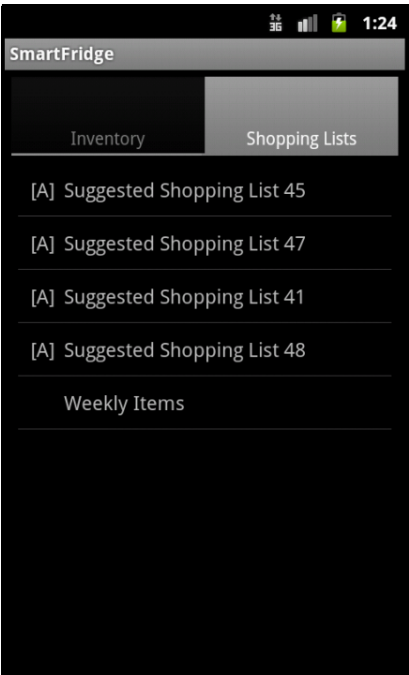


Figure 8: Mobile Interface Shopping List Tab



Figure 9: Mobile Interface Showing Item Details

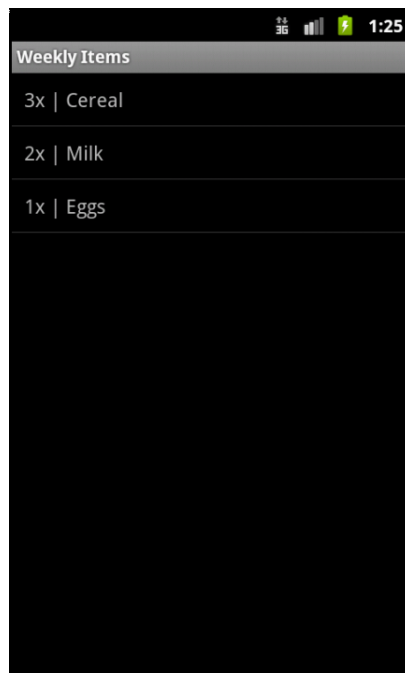


Figure 10: Mobile Interface Showing Shopping List Details

The mobile application was also designed to be tolerant of interrupted connections, since telecommunication networks often have spotty coverage. To mitigate the impact of dropped connections, updates were transferred from the BeagleBoard to the mobile application in a sequence of short communications; a large number of small messages is more robust than a small number of large messages. This structure facilitated the “need to know” distribution of information as well.

To actually communicate with the web interface a combination of HTTP requests, PHP, and JSON was used. The mobile application indicated the type of query to the web interface using an HTTP request. A snippet of PHP code on the server side actually performed the query into the BeagleBoard’s MySQL databases, and the results were formatted as a JSON string and returned. The Android application parsed the JSON string and formatted the results into objects.

4.3 Web Interface

The web interface itself was hosted by a Cherokee web server running on the BeagleBoard. A simple HTML front end was also hosted in addition to the interface code for the mobile application. This interface consisted of two tables, one to show the current inventory and another listing the available shopping lists. The elements in the shopping lists table are links, and clicking any shopping list will lead to another page showing a table with the items in that shopping list. Screen captures are shown for the web interface front end in Figures 11 and 12.

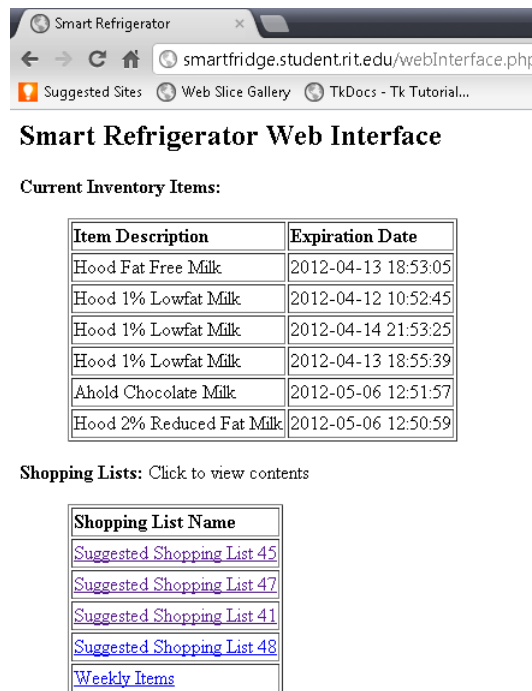


Figure 11: Web Interface Front Page

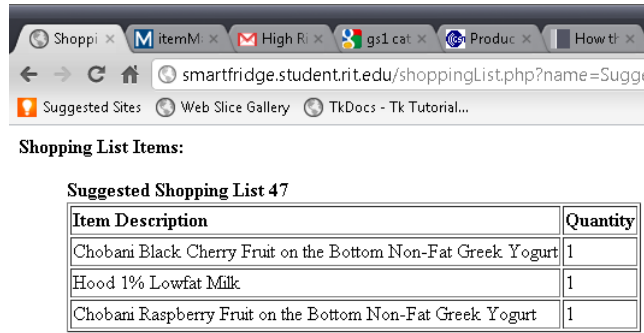


Figure 12: Web Interface Shopping List Page

4.4 Database Definitions

The various databases were realized as MySQL databases on the BeagleBoard. MySQL databases are certainly reliable, and reduced testing only to ensuring correct interaction with the databases. Figure 13 illustrates the partitioning of data into the multiple tables stored on the BeagleBoard. The most intuitive database is the Item Inventory Table, which stores the current contents of the refrigerator and their corresponding purchase and expiration dates. However, only the last purchase date is not be sufficient for the shopping list prediction algorithm, so a separate item history table stores all purchase dates indexed by UPC. The purchase dates are not stored as a list but just as separate rows in the table; the number of rows returned by a query indicates the number of purchase dates. A shopping list table is included to store the top-level information about a shopping list: the ID, name, and creation date. The shopping lists also include a flag to indicate whether each shopping list was manually created or generated by the shopping list creation algorithm; the user may find this information useful when evaluating shopping habits. To coordinate between the table of items and the shopping lists themselves, an intermediate linking table was included to store the actual items included on a shopping list. This table is indexed by the shopping list identification number, and each row represents an item belonging to a shopping list. The number of rows returned by a query against a specific list ID indicates the number of items on that shopping list.

4.5 Other Design Considerations

Extensibility – The Smart Refrigerator system developed is only a first step toward tackling food waste; integration of the concepts in this prototype into a larger context could provide enormous utility. Possibly the expiration date and purchasing prediction systems could be improved and applied in commercial domains. Both small restaurants and national chains could benefit from more accurate shelf life predictions. The shopping list suggestion algorithm may also be improved by aggregating data from multiple users.

Manufacturability – The Smart Refrigerator is a mainly software system and will therefore be easy to reproduce; the same code package could easily be mass-produced. However, the BeagleBoard is a very general processing platform. If this prototype system were manufactured commercially, a more tailored and application specific platform could be desirable. The system does not have a true need for a complete Linux environment, even though it is ideal for rapid development of a prototype.

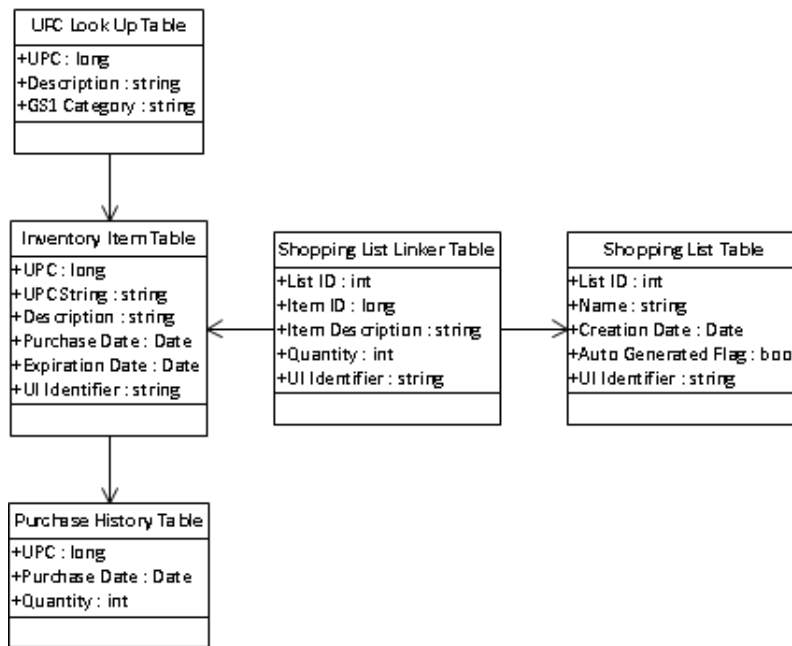


Figure 13: Separation of Information in Databases

Reliability - The components developed and tested externally, such as the Angstrom operating system, were assumed to be very robust in comparison with the modules developed specifically for this prototype. The user interfaces and prediction algorithms were both tested extensively, however in both cases it is difficult to eliminate the possibility of error. It was difficult to simulate all interrupted connection scenarios in testing and mobile interface likely did not encounter all possibilities and defects may have gone unnoticed. The shopping list recommendation algorithm was not exposed to all shopping habits in testing, and therefore may perform poorly in some untested scenarios. Fortunately, the system will not incur any damage from a failure and will not immediately endanger the user in event of a failure.

Background - Experience with Linux based operating systems throughout the computer science sequence has been helpful. The computer science sequence and software engineering also have provided a valuable introduction to user interface development. The project does not contain significant hardware design, though skills learned in Interface and Digital Electronics may be useful in interfacing with the temperature and humidity sensor.

Multidisciplinary Aspects - This prototype system requires coordination between hardware and software and requires a mixture of Computer Engineering and Software Engineering skills. However, multidisciplinary projects often carry a mechanical connotation and this system does not require integration of any mechanical components.

5 Considerations

The Smart Refrigerator system developed is a step toward promoting sustainability and good stewardship of natural resources. Both the New York Times articles mentioned in the statement

of needs, and other reports [1, 3], indicate that approximately 27% of all food available for consumption is lost to waste. A study published by the UN Food and Agriculture Organization declares the global percentage is even higher, totaling 1.3 billion tons or 33% overall [4]. The system designed will increase awareness about expiring items, with the goal of reducing these figures. Hugh Collins from AOL News speculates that food waste is dismissed subconsciously; many foods are cheap, and the average consumer does not think about the aggregated cost of these small wastes [3]. By providing reminders to the user, the Smart Refrigerator can remedy this source of waste by keeping the user aware of all their purchases. Expiration of food products themselves is not the only source of waste involved with grocery shopping. Making unnecessarily frequent trips to a store for forgotten or unexpected items also wastes resources. The shopping list recommendations provided by the Smart Refrigerator will hopefully mitigate waste here as well. A final consideration of the system is health and safety, by providing reminders about expiring products the risk of eating expired products will hopefully be decreased.

6 Cost Estimates

We submitted a proposal to the ARM student design contest requesting a BeagleBoard-xM and power adapter. Our team received a BeagleBoard-xM at no cost, however the ULCD7 Lite display and power adapter were not provided and were purchased. We also already own many of the principle system components; the dorm room refrigerator, Android smart phone, and keyboard did not need to be purchased.

Table 5: Cost Table

Part	Retail Cost	Our Cost
BeagleBoard-xM	\$149	\$0
BeagleBoard-xM Power Adapter	\$26.33	\$26.33
Dorm Room Refrigerator	\$100	\$0
Android Smart Phone	\$100	\$0
ULCD7 Lite Display	\$151.50	\$151.50
UPC Barcode Scanner	\$40.67	\$40.67
USB Keyboard/Keypad	\$10	\$0
Digilent Pmod TMP2 Temperature Sensor	\$42.55	\$0
Total Cost	\$620.05	\$271.05

7 Testing Strategy

Testing of the Smart Refrigerator was divided into unit testing of the various subsystems and then top-level integration testing once the sub-systems had been connected. Some components used within the system, such as the Angstrom operating system and SQL database implementation, which had undergone extensive testing prior to use in our system, were tested only to ensure proper configuration. The principle subsystems tested were the base station user interface, mobile user interface and network interface, expiration date and shopping list prediction algorithms, and integration with the BeagleBoard.

7.1 Base Station User Interface Testing

The main testing focus was placed on the user applications, both the software running on the base station and the web and Android interfaces. Unit testing was performed during development of each component, as well as integration testing of the final application. This subsection will focus on top-level testing of the base station user interface as a module, listing tests particularly directed at the engineering specifications and user requirements. Tests directly motivated by the requirements specification and engineering specifications are listed below, and the test procedure completed is tabulated in Table 6.

- The user interface is required to be easy to use and intuitive; in order to verify this someone not involved in the project was enlisted to contribute to top-level testing of this sub-system. This also was also tested quantitatively; tests were performed to ensure the most used items were presented on the default tab, and the most frequently used controls were the most accessible.
- The user interface provides access to the current inventory, which was stored using a MySQL database. The principal test effort at this step were verifying integration of the display with the database, not verifying the storage of items themselves.
- The user interface provides both read and write access to shopping lists, also stored using a MySQL database. Testing of this feature again focused on the ability of the interface to query and modify database entries, not on the database implementation itself.
- The user interface was required to provide a method to update expiration estimates. Testing of this subsystem did not verify that the update is reasonable or correct but simply verified that this user interface action triggered an update from the expiration prediction subsystem.
- To achieve the principal goal of the system, the user interface needed provide a notification of items about to expire. Testing of this subsystem did not verify that the expiration estimate was reasonable or correct, but simply that if triggered by the expiration prediction subsystem the user interface displayed an indication.

Table 6: Base Station User Interface Test Cases

Test Writer: Steven Strapp									
Test Case Name:		Base Station Interface Top-Level Unit Tests			Test ID #:		Base-01		
Description:		Verify that the base station user interface meets the requirement and engineering specifications. Some, such as usability will be evaluated qualitatively and are difficult to outline in this way.			Type:		White Box		
Tester Information									
Name of Tester:		Steven Strapp				Date:		04/29/12	
Hardware Ver:		1.0				Time:		10:50	
Setup:		User interface subsystem should be entirely integrated with prediction subsystems and SQL databases. System should begin without shopping lists or inventory. System date should be made mutable to facilitate quick simulation of expiration.							
Test	Action	Expected Result		Pass	Fail	N/A	Comments		
1	Enter test product code	Switch to inventory tab, entered product should be shown. Inventory should be otherwise empty.		X					
2	Wait for test product to nearly expire	Interface should display a notification indicating expiring item.		X			Item added to list of items near expiration on main tab.		
3	Use interface to indicate product has not yet expired	Verify that prediction sub-system is triggered to update its expiration estimate for this product.		X					
4	Create fake shopping list	Verify that list becomes accessible through base station and Android interface		X					
5	Modify items on fake shopping list	Verify that changes are retained and visible through base station or Android interface		X					

7.2 Mobile User Interface and Network Interface Testing

The web and mobile interfaces had their own set of tests, focused on basic functionality and interoperability on various platforms. The web interface was tested on the most popular browsers (Google Chrome, Firefox, and Internet Explorer), as well as some of the most popular mobile platforms (Android, WebOS, and iOS). The Android interface was tested on various recent versions of the operating system; major versions between 2.3 and 4.0 were tested.

Table 7: Mobile App Tests

Test Writer: Ben Reeves						
Test Case Name:		Downloading large database updates over an intermittent network connection			Test ID #:	Mob-01
Description:		Ensure that the database is correctly downloaded even if the device's network connection is interrupted. This could be due to loss of service, a disabled network adapter, or the device powering down.			Type:	White Box
Tester Information						
Name of Tester:		Ben Reeves			Date:	4/19/12
Hardware Ver:		1.0			Time: 10:30	
Setup:		System should have a fresh install of the application and no previous copies of the database downloaded.				
Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Initiate download update of the database	System should connect to the server and begin downloading.	X			
2	Sever device's network connection	System should pause the download upon sensing the interrupted connection.	X			
3	Reconnect device to the network	System should resume download of the database			X	Download is restarted, not resumed.
4	Allow update to complete	System should download the remaining portion of the database	X			

Table 8: UI Usability Test

Test Writer: Ben Reeves								
Test Case Name:		UI Usability Test			Test ID #:		UI-01	
Description:		Ensure that the both the web and mobile versions of the User Interface are accessible and intuitive.			Type:		White Box	
Tester Information								
Name of Tester:		Ben Reeves			Date: 4/19/12			
Hardware Ver:		1.0			Time:		11:45	
Setup:		System should be representative of one which is in active use; that is, its database should contain both shopping lists and grocery items associated with them.						
Step	Action	Expected Result			Pass	Fail	N/A	Comments
1	System is given to a user unfamiliar with its operation and submitted to stress testing	User should experience little difficulty navigating the application and experience no bugs, freezes, or crashes.			X			

Table 9: UI Interoperability Test

Test Writer: Ben Reeves						
Test Case Name:		UI Interoperability Test			Test ID #:	UI-02
Description:		Ensure that the both the web and mobile versions of the User Interface are fully compatible with popular browsers.			Type:	White Box
Tester Information						
Name of Tester:		Ben Reeves & Steven Strapp			Date:	4/20/12
Hardware Ver:					Time:	10:40
Setup:		System should be representative of one which is in active use; that is, its database should contain both shopping lists and grocery items associated with them.				
Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Interface is accessed via Mozilla Firefox and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			
2	Interface is accessed via Google Chrome and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			
3	Interface is accessed via Microsoft Internet Explorer and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			
4	Interface is accessed via Android 2.3 and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			
5	Interface is accessed via Android 4.0 and subjected to stress testing	Interface is displayed properly, no artifacts or misplaced elements apparent.	X			Web interface has small break in table with default browser, not critical.

7.3 Shopping List and Expiration Prediction Test

Testing of the expiration prediction and shopping list prediction subsystems would have been difficult if the system's date could not be adjusted artificially; testing should occur over a few minutes rather than a series of days. For expiration date testing the system's date was made easy to change artificially, so products appear to expire very quickly. The intelligence of the system could then be tested by providing feedback that products expired more or less quickly than expected, and evaluating the updated predictions. By simply accelerating the rate with which the system changes date, this subsystem could be tested without adding specific test products with low shelf lives and without changing the algorithms to update more frequently. A set of tests is listed for this subsystem below.

- Enter a product code and verify that the expiration date system is initialized with recommended "rule of thumb" value.
- Provide feedback indicating that a product expired before estimate and validate that the estimate is decreased. Also validate the opposite case: providing feedback that a product had not yet expired on estimated date should increase the estimate.
- Enter a product code and advance the system time until the product is nearly expired. Verify that the prediction subsystem has indicated to controller that the product is nearing expiration.
- Rescan a product code after regular intervals, indicative of uni-modal shopping habits. Verify that the system recommends the product should be purchased again on this mode. Date may be artificially advanced to facilitate quick testing.
- Continue from the previous case and add outlier shopping habits. Validate that system continues to recommend purchasing the product again on the mode.
- Enter various products with different purchasing habits. Verify that the system recommends purchasing the products with the highest probabilities.
- Rescan a product code after regular intervals, then add significant variation. Observe that system attempts to track the variation in habits.
- Rescan a product code after varying intervals, indicative of bi-modal shopping habits. Verify that the system recommends purchasing the product on both modes.

7.4 Integration with BeagleBoard

Preliminary testing focused on the BeagleBoard itself and its ability to interact with the desired peripherals. The system required an LCD screen, a USB barcode scanner, a network connection, a keypad, and temperature sensor. Basic functionality of these components was tested thoroughly during development, as well as during final system testing.

The SQL database used to store all data for the system was tested once the core of the user application had been coded. Using the base station interface the databases were populated with test data in order to ensure that the database was configured as desired, and to verify that the user application was properly communicating with the database alongside the web interface.

It was difficult to outline exactly what testing will be required for the processing platform,

since it is unclear what compatibility issues would arise that would not be presented by a conventional platform, where ideally the system would be entirely “plug and play.” However, listed below is a baseline sequence of tests performed.

- Verify that the BeagleBoard, with power adapter, can power all peripheral devices reliably so that no sporadic failures occur; this was performed as an endurance test.
- Verify that MAC address of Ethernet interface can be statically assigned and the BeagleBoard can be pinged reliably; this was performed as an endurance test, cycling power and disconnecting the board multiple times.
- Verify that the BeagleBoard can reliably interface with the USB scanner and USB keypad; these tests were performed by writing to a text editor instead of the program interface to isolate failures.
- Verify that the BeagleBoard consistently receives accurate temperature measurements from the sensor, via the I²C interface. The measurements were verified with an external sensor.
- Verify that the touchscreen display accurately records users clicks and controls the pointer; tested outside of the user interface to isolate failures.
- Verify that touchscreen accurately displays the graphical user interface without artifacts or distortion consistently, and ensure all controls on the display are accessible.

8 Risks

The risks unresolved in the prototype system are best grouped by subsystem. Some potential risks were simply due to unknown availability of parts, others require further exploration, and still others are more fundamental risks.

Availability of LCD Display – The ULCD7 Lite was the preferred choice for the display to accompany the BeagleBoard. The 7-inch resistive touchscreen is designed to work with the BeagleBoard and has drivers built into the Angstrom Linux distribution. However, the touchscreen did not seem to be stocked by online suppliers and our request through the Arm Developer Day proposal process was not granted. Since the BeagleBoard provides a DVI-D and S-video output, this risk could have been mitigated using a traditional computer monitor if the preferred touch screen was not available. The availability of these alternate interfaces also facilitated development while waiting for the desired part.

Mobile Application Data Exchange – Exchanging inventory and shopping list data with the mobile application presented two risk areas: the amount of data plan usage required and interrupted network connections. These risks were partially mitigated by the design, which exchanges data in small increments on a “need to know” basis only; however the degree to which these risks are truly problematic is still not known. For development, the smart phone was connected with WiFi instead of a costly data connection; however, the data requirements were still taken into account during development to ensure this risk does not become a latent problem. Interrupted connections were considered in the design, but the effectiveness of the design was difficult to measure since it was difficult to repeatedly interrupt the data connection.

Practicality of Prediction Algorithms – The shopping list prediction algorithm outlined appeared optimal for the test data considered. However, the analysis performed did not consider a large sample of shoppers, and the conclusions drawn may not be appropriate for a larger population. Also, the prediction algorithm outlined does not perform well with only a few samples; it is unclear how unacceptable this warm-up period will appear to end users. The strategy presented is also quite heavy-handed and may be superfluous for this prototype system. The system will not serve as an effective shopping aid without an accurate prediction system, but a prototype may have been acceptable with a much simpler system requiring significantly less effort.

Efficiency of Database Access – Efficient practices for database access were not known or considered when partitioning information into separate tables. The design was modified during development to mitigate this risk. For example, using the same database to store all possible items and the current inventory appeared to be a very inefficient choice. This strategy required looking over all elements and extracting only those with non-zero quantities. If the system were extended to large commercial applications, or if the BeagleBoard were replaced with a more cost-efficient and less powerful alternative, this organization may have become a significant risk.

Table 10: Summary of Risks

Risk	Comments
Availability of LCD Display	The ULCD7 Lite display took several weeks to procure. However, this delay did not significantly effect the project time line and this risk was averted.
Mobile Application Data Exchange	There was an initial concern that the amount of data exchanged with the mobile application would be problematic. This risk did not impact testing for our simple prototype system but may remain a latent issue if the system were expanded to a larger scale.
Practicality of Prediction Algorithms	There was initial concern that the shopping list prediction algorithms would take a significant amount of development time at the expense of areas that would be more obvious to an end-user. However, using SciPy this risk was absolved.
Efficiency of Database Access	This risk appeared more problematic with the original database schema, and the schema was updated during design to resolve this risk.

9 Milestones

Milestone	Scheduled Date	Assigned	Modified Date	Comments
BeagleBoard procured	February 10, 2012	SS	NA	Complete
Angstrom operating system running on board	February 24, 2012	DS	NA	Complete
Peripherals properly interfacing with board	March 02, 2012	DS	April 27, 2012	Complete. Touch screen Angstrom image works for both touch screen and temperature sensor.
Basic mobile UI, suitable for debugging	March 09, 2012	BR	NA	Complete
Basic base station UI, suitable for debugging	March 09, 2012	SS	NA	Complete
Database I/O configured	March 16, 2012	DS	March 23, 2012	Complete. MySQL databases configured, can be accessed with Python application.
Testing and integration of temperature and humidity sensor	March 16, 2012	SS/DS	April 27, 2012	Unable to interact with one wire sensor using BeagleBoard GPIO and level shifter. Replacement I ² C temperature sensor from Dr. Mondragon works correctly.
Database and web server hosted by BeagleBoard	March 16, 2012	DS	NA	Complete
BeagleBoard touchscreen display procured	March 16, 2012	DS	NA	Complete
Mobile application integrated with web server	March 30, 2012	BR	April 20, 2012	Complete
User profiling and statistical analysis	March 30, 2012	SS	April 6, 2012	Complete

Smart Refrigerator Proposal

Milestone	Scheduled Date	Assigned	Modified Date	Comments
Shopping lists, item modification, basic settings	March 30, 2012	SS		Complete. Added context menus and hidden “administrator” panel.
Updated base station UI	April 6, 2012	SS		Complete, awaiting review by team or outside testers.
Updated mobile application	April 6, 2012	BR	April 20, 2012	Complete
Integration testing and system verification	April 13, 2012	BR	April 27, 2012	Complete
Web interface development	April 20, 2012	SS		Complete. Does not look flashy but is effective and functional.
System testing and demo preparation	April 20, 2012	SS	April 27, 2012	Complete
Project Website	May 3, 2012	SS		Complete
Project Demo	May 3, 2012	Team		
Imagine RIT Demo	May 5, 2012	Team		

9.1 Gantt Chart



Figure 14: Project Gantt Chart

10 Perspective

The design of the final project adhered to the original proposal, however some of the implementations themselves changed. Some aspects of the system, such as the web user interface and database organization, were not thoroughly enough defined in the proposal and were not clear until the actual implementation. Difficulties and deviations from the original proposal are organized into subsections below.

10.1 Base Station Code Package

The most significant change in the base station code package was the programming language used; the original proposal declared C++ along with the Qt user interface framework would be used, whereas Python and the Tk framework were eventually used. Python is much quicker to develop and Tk allows for very quick creation of user interfaces. Python was also an advantageous choice for the prediction subsystems. The scientific computing library SciPy provides clustering and kernel density estimator code, both of which were considered as options for the shopping list recommendation system. The prediction system itself was revised after considering implementation with MySQL databases. The clustering strategy initially chosen during conception selection became undesirable in that it required clustering information to be stored in the database, and the number of clusters were not known a priori. Alternatively, the clustering could have been performed again at every prediction step, however this was computation undesirable. Once database storage was considered in the concept selection a non-parametric estimate appeared optimal instead.

The user interface design followed the mock-ups very closely. The only significant change was to remove the expiration warning pop-ups and replace them with a list of high risk items on the main panel. Once the GUI was created and experimented with it became clear that by only showing a pop-up it was too easy to forget the warning after the pop-up had been cleared. Displaying a list of items near expiration on the main window prevents users from forgetting items that had been identified as nearing expiration.

10.2 Web Interface

The specifications for the web interface were not clearly defined in the original proposal, and expectation for this interface varied between team members. The expectations and implementation for this interface were not resolved until initial development. The web interface was implemented as a simple HTML page with embedded PHP snippets to query the databases hosted by the BeagleBoard.

10.3 Mobile Interface

The initial plan outlined in the proposal was for the mobile application to interact with the BeagleBoard through the same web interface. However, part way through development it was suggested that it might be simpler to interact directly with the remote databases on the BeagleBoard. Attempts were made to connect to the databases using an object-relational mapping library without any success, and it was later discovered that Android does not support remote database access. Instead, the design reverted back to the original plan. Another small PHP snippet on the BeagleBoard's web server was included to respond to HTML post requests, access the database, and return result to the Android application. The Android application would

issue a post request indicating which tabel should be accessed, the server side PHP would perform this query and return results as a JSON string. The Android application parsed the JSON string and converted the returned values into Java objects. Aside from this indecision, the Android development proceeded as expected.

10.4 BeagleBoard Integration

A number of integration issues arose while using the BeagleBoard. The first issue encountered was with the Python base station code package. The shopping list prediction system used the SciPy library, however prebuilt binaries for this library are not available for ARM architectures. Compiling this package from source was unexpectedly difficult and required resolution of many dependencies. Next, the temperature sensor initially purchased, an RHT03, could not be interfaced with the BeagleBoard. The RHT03 uses a custom one wire interface that enforced very accurate timing requirements; the BeagleBoard, running a non real-time operating system, was not able to meet these requirements. Attempts were made to use more accurate timers or interrupts from the GPIO pins but neither of these methods accurately retrieved information from the sensor. Instead, an I²C temperature sensor was obtained which worked trivially. Finally, the ULCD7 Lite touchscreen was initially only partially functional with the Angstrom image used; the display worked but would not respond to touch. Investigation revealed that additional modules needed to be compiled in with the Angstrom kernel. Instead of pursuing this option, the demo Angstrom image provided for the touchscreen was used and the components developed on the earlier image was reinstalled on the touchscreen image. This strategy required repeating installations and configurations but also appeared to be the option that was simplest and most likely to succeed.

11 References

- [1] Martin, Andrew. "One Country's Table Scraps, Another Country's Meal." New York Times. N.p., 18 May 2008. Web. 26 Jan 2012. <http://www.nytimes.com/2008/05/18/weekinreview/18martin.html?pagewanted=all>.
- [2] Ridden, Paul. "LG launches first Smart-Grid appliance: the Smart Fridge." gizmag. N.p., 27 Apr 2011. Web. 29 Jan 2012. <<http://www.gizmag.com/lg-smart-fridge/18502/>>.
- [3] Collins, Hugh. "Study: US Food Waste Is a Huge Energy Drain." AolNews. N.p., 02 Oct 2010. Web. 29 Jan 2012. <<http://www.aolnews.com/2010/10/02/study-american-food-waste-is-a-huge-energy-drain/>>.
- [4] Ramaker, Rob. "Food waste is hard to combat." Resource. N.p., 26 Jan 2012. Web. 29 Jan 2012. <http://resource.wur.nl/en/wetenschap/detail/food_waste_is_hard_to_combat/>.