Pantrymate: A Mobile Application to Reduce Household Food Waste

Jake Levin

Advisor: Professor Alan Kaplan

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Arts
Department of Computer Science
Princeton University

June 2018

Abstract

Over 30% of food produced in the United States is wasted every year. This waste has far-reaching global effects, and reducing it is critical in addressing problems such as food insecurity, resource scarcity, and climate change. Within the United States' food supply chain, household food waste is responsible for 14% of total food loss. The goal of this project is to try to reduce food waste by targeting key sources of loss at a household level. Pantrymate is a mobile grocery shopping and food management application that tracks a user's food inventory. Using this inventory data, the application alerts users about potential instances of overbuying, food expiration, and unnecessary wasting of leftovers. Additionally, Pantrymate uses the data gathered on a user's food inventory to help improve household nutritional awareness. Pantrymate uses a user's shopping lists to build its inventory, and therefore takes a uniquely proactive approach in addressing the key sources of household food waste. The application design also focuses on reducing manual input requirements to lower the time threshold of use. Pantrymate has been implemented using React Native, a framework for building cross-browser mobile applications, so as to increase potential accessibility of this system to all individuals with an Android or iOS smartphone. Upon completion of this report, a working Android APK and iOS release bundle for the functioning application have been exported and tested on their respective hardware devices. In order to evaluate Pantrymate, the application's behavior was simulated given 100 users' grocery shopping data. Results show that Pantrymate proactively targets key factors in household food waste. Over an average of fourteen grocery shopping orders per user, the application alerted users to close to one hundred potential instances of over buying and inventory expiration.

Acknowledgements

I would like to thank my advisor, Professor Alan Kaplan, for all of his time and guidance. He challenged me to always think critically about my work, and encouraged me throughout the entirety of the thesis process to produce the best result that I could.

I would also like to thank my second reader, Professor Robert Fish, for his continued mentorship over the course of my undergraduate independent work experience.

Lastly, I would like to thank all of my family and friends, many of which overlap. Your love and support means the world to me, and you are the reason I work to improve it in the small ways that I can. There are too many of you to list by name, but I have no words to express how grateful I am to you all for playing such a significant role in my life.

This thesis is dedicated to my father, an incredibly (and perhaps too) generous cook, and to my mother, an incredibly brave cleaner of fridges.

Contents

Abstract . . . iii

Acknowledgements . . . iv

1 Introduction **1**

1.1 Problem Background . . . 1

1.2 Motivation and Goal . . . 3

1.3 Use Case . . . 4

2 Related Work **6**

3 Approach **10**

3.1 Proactively Targeting Key Factors in Waste . . . 11

3.2 Automation to Alleviate Time Constraints . . . 11

4 Implementation **13**

4.1 User Login and Authentication . . . 16

4.1.1 Front-end . . . 16

4.1.2 Back-end . . . 17

4.2 Shopping Lists . . . 18

4.2.1 Shopping List Creation: Front-end . . . 18

4.2.2 Shopping List Creation: Back-end . . . 20

4.2.3 Shopping List Management: Front-end . . . 22

4.2.4 Shopping List Management: Back-end 23
4.2.5 Active and Recent List Displays: Front-end 24
4.2.6 Active and Recent List Displays: Back-end 26
4.3 Nutrient Content Display 26
4.3.1 Nutrient Content Display: Front-end 27
4.3.2 Nutrient Content Display Information: Back-end 29
4.4 Inventory Management 29
4.4.1 Food Item Inventory Display: Front-end 30
4.4.2 Food Item Inventory Display: Back-end 33
4.4.3 Manual Item Addition: Front-end 34
4.4.4 Manual Item Addition: Back-end 35
4.4.5 Barcode Scanning: Front-end 35
4.4.6 Barcode Scanning: Back-end 37
4.4.7 Food Item Expiration 38

5 Evaluation **42**
5.1 Data and Metrics Used for Evaluation 42
5.2 Evaluation Results 44

6 Conclusion **47**
6.1 Limitations 48
6.2 Future Work 49

A Code **51**

B Sample Evaluation Data **52**

Chapter 1

Introduction

1.1 Problem Background

According to the United States Department of Agriculture (USDA), an estimated 30-40% of the United States food supply is wasted each year, totaling to approximately 133 billion pounds and $161 billion worth of food [1]. While there is an interesting debate surrounding the accuracy of these estimates, particularly with regard to the lack of a standardized definition of food waste [2], the amount of food wasted in the United States is an undeniably large problem with implications affecting issues including food insecurity, resource scarcity, and climate change. In 2016, the USDA reported that 12.3% of U.S. households were food insecure [3]. A food insecure household is one that either lacks access to an appropriate quantity of food, or cannot sustain an affordably balanced diet. In an effort to address this issue, the USDA Office of the Chief Economist initiated several programs to recover and recycle food across every level of the U.S. food supply chain (FSC) [4]. The FSC is the entirety of the process involved in distributing food from farmers to consumers.

An international study conducted in 2012 reported that the production of wasted food within the global FSC uses 24% of total freshwater resources, 23% of cropland area, and 23% of fertilizer used in food crop production, and pointed to the United States as one of the countries primarily responsible for these losses [5]. Particularly concerning is food production's reliance on phosphorous, a non-renewable resource present in almost all mineral fertilizers. Ninety percent of global demand for phosphorous is for food production, and phosphate reserves are predicted to be depleted within the next 50-100 years [6]. Food waste is also a primary contributor to greenhouse gas emissions. To put this in perspective, the World Resources Institute has noted that "if food loss and waste were its own country, it would be the world's third-largest emitter - surpassed only by China and the United States" [7]. A study conducted in 2014 calculated that the production of food lost in the United States in 2010 was responsible for 160 MMT CO2-eq of GHG emissions, an emissions level equivalent to that of roughly 33 million passenger vehicles annually [8]. Figure 1.1 shows the visual breakdown of these emissions, categorized by food type, provided by the aforementioned study.

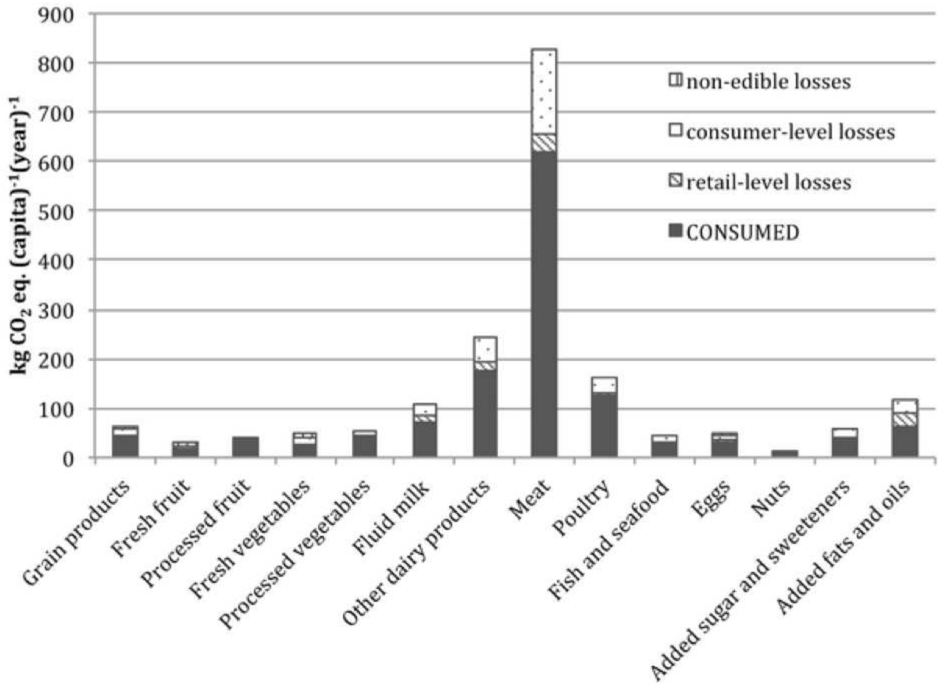


Figure 1.1: Greenhouse Gas Emission Estimates of U.S. Dietary Choices and Food Loss [8].

Another major food-related problem is the nutritional imbalance in the diets of many U.S. households. It is estimated that about 75% of the population has a diet that is low in fruits, vegetables, dairy, and oils [9]. While the focus of this project is not nutritional imbalance, the approach used here to reduce food waste uses data that is fundamentally tied to modeling household nutrition, and thus tries to concurrently raise users' awareness of their diets' nutrient content.

1.2 Motivation and Goal

The problem of food waste is significant not only in its size and scope, but also in factors of loss. Food waste occurs at every step of the FSC as food makes its way from producer to consumer. A diagram of the percentage of food wasted at each step of the FSC across different regions, as published by the Food and Agriculture Orginization of the United Nations, is seen below in Figure 1.2 [10]. This project focuses specifically on consumer waste at a household level. By limiting the scope of waste under consideration, more effective solutions can be implemented to address the critical sources of loss at this step within the FSC. Roughly 14% of U.S. food waste occurs at a household level [11]. This equates to over 200kgs and an estimated $600 of food per year per household.

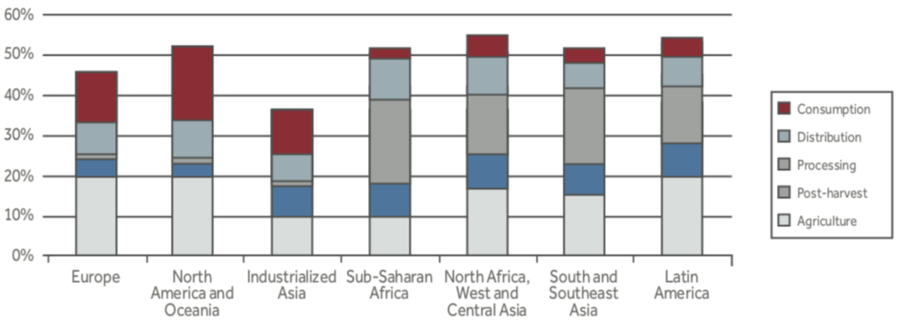


Figure 1.2: Percentages of total food wasted at different steps within the FSC by region [10].

The primary goal of Pantrymate is to help reduce household food waste by developing a mobile application to aid users in food inventory management and food acquisition planning. Secondarily, Pantrymate hopes to improve nutritional awareness by providing an automated interface for easily accessing data on the nutrient content of a user's food inventory. The application will help keep track of current pantry, fridge, and freezer inventories as well as the expiration dates of purchased food, and will use this data to aid in more efficient grocery shopping, while avoiding food wasted either by expiring or being thrown away as leftovers. Each item in a user's inventory and shopping lists is also linked to a one-click nutrient content display. Furthermore, Pantrymate has been developed as a mobile application that runs on both Android and iOS to maximize accessibility and make this application available to anyone with a smartphone. The scope of this project has been framed within the context of food waste and dietary imbalance within the United States. However, this application is intended to be applicable to all developed countries with sufficient infrastructure for households to maintain a stored food inventory.

1.3 Use Case

A possible use case is as follows: Jane is about to go grocery shopping. She opens up the application to quickly jot down her shopping list for the enchiladas she plans on making tonight. When she includes salsa on the list, the application notifies her she might have a jar left over from the Super Bowl party two weeks ago. It also notifies her that she has chicken that is about to expire, and leftover beans from the other night. Jane is set on making the enchiladas, but she decides to make half of them with chicken, and does not bother putting beans on her shopping list. At the store she uses her phone to scan the bar code on the tortilla wraps she had been planning on buying. They are automatically checked off her shopping list, and the app displays a nutrient breakdown of the tortilla product she is holding. Jane decides that perhaps she should go instead with the whole wheat tortillas, noting that they are much lower in sugar. Looking at her inventory, Jane also realizes that her diet has recently been low in produce, so she purchases some bell peppers. The next day Jane marks down that she has leftover peppers and rice, and the app notifies her that these items will keep for approximately four days if kept in her fridge. Overall, Jane has saved money she would have otherwise spent on new rice and beans, she has found a way to put to use her expiring chicken and salsa, and her diet has been improved by whole wheat tortillas and fresh produce.

Chapter 2

Related Work

While significant, food lost at a household level is a relatively small fraction of total food waste. As such, technological efforts to reduce food waste have targeted loss primarily at an industry level, focusing on farms, restaurants, catering companies, and big businesses. Only recently has household food waste been recognized as a problem to be addressed via technical solutions. To best address this problem, it is essential to understand the primary factors contributing to this food waste.

Looking specifically at household food waste, there are a number of sources of waste that can be addressed. In 2015 a team of researchers conducted fifty interviews with experts on food waste and consumer relations, and reported that "most experts especially stressed that the lack of planning and management of purchase, storage, preparation and reuse of food and meals is at the heart of the problem. It is rooted in constraints of time and little priority given to food waste reduction behaviors" [12]. Additionally, a recent study conducted in Austria and the U.K. examined the food practices of fourteen homes in an attempt to analyze how a technological solution could improve food management in the home [13]. This study similarly found that a lack of planning in consumer shopping routines, poor food storage, and inefficient use were the biggest contributing factors in household food waste. They concluded that technological solutions targeting shopping coordination, food storage management, and community food sharing would be most effective at reducing this waste [13].

Of these solution strategies, food storage management systems have received the most attention to date. FridgePal allows users to keep an inventory of their food inventories by manually inputting or scanning packaged products with bar codes [14]. It also manages expiration dates for this inventory, and allows users to search for recipes and create shopping lists in app. FridgePal, unlike Pantrymate, builds users' inventories manually by inputting or scanning every item. This is highly burdensome on the user and makes it difficult for users with serious time constraints to make use of the application. Furthermore, FridgePal does not provide nutritional data on a user's food inventory. Freshbox is another application that acts as a virtual refrigerator, keeping track of inventory and displaying it on a virtual shelf [15]. Freshbox also manages expiration dates and allows users to create shopping lists using ingredients in their inventory. Like FridgePal, Freshbox still requires users to manually create their virtual inventory, and does not deal at all with nutrition. EatChaFood is another prototype household food management application designed to avoid unnecessary expiration [16]. Of particular interest is that EatChaFood was implemented with a two-click item entry process, specifically designed to reduce manual input. Pantrymate hopes to cut down on the need for manual input even further by constructing the majority of user inventories through shopping lists and through the aid of barcode scanning.

Applications for grocery shopping coordination have also been explored, however the research conducted in this area has been primarily towards utilizing mobile technology to encourage shoppers to make healthier dietary decisions. Students at University of Colorado built a mobile-based augmented reality system using color-based AR tagging in an attempt to quickly point shoppers towards healthier products while steering them away from unhealthy ones [17]. Another application was built by students in Zurich to classify products both on shopping lists and on shelves in-store based on images taken from smartphones [18]. These applications do little to reduce food waste, but were found to be reasonably effective at proactively encouraging shoppers to make better informed decisions. Pantrymate looks to adopt this proactive approach to not only increase awareness of shoppers' diets, but also to prevent overbuying.

Community food sharing is another interesting approach that has been explored both in theory and in practice with applications such as AirShare [19] and Waste No Food [20], which currently operates quite successfully within San Jose. In 2016, a smartphone based supply chain was proposed in Aurangabad City to connect donors with excess food at risk of going to waste to hungry individuals in need throughout the city [21]. While a community food sharing application was initially considered for the topic of this thesis, it would be extremely difficult to implement such an application without raising serious concerns regarding proper food handling techniques within individual homes. Thus, Pantrymate will instead target shopping coordination and food storage management.

Further research has suggested that technology can also be used to reduce food waste through methods of social persuasion [22] [23]. In 2012, a social persuasive system called BinCam was designed to engage young adults collectively in reflections on their waste management lifestyle by automatically logging disposed items through digital images that were uploaded to an application on Facebook [24]. While an interesting approach, participants of BinCam reported that they did not notice a significant decrease in the amount of food they wasted. This is likely because BinCam takes an entirely reactive approach, doing little to actually stop the waste from occurring. Interestingly, users of BinCam noted that they appreciated how little the system interfered with their existing schedules. Pantrymate hopes to learn from these studies by requiring minimal manual input, while taking a predominantly proactive approach to aid in the planning and management of household food.

It is finally worth noting that there has been research done in defining the theoretical specifications of a SmartPantry management system [25]. Ideally, a SmartPantry would be a component of a larger smart house, and would stock all of the supplies a homeowner needs. It would manage foods, place orders to be delivered and distributed among its storage system, and would ensure products were only replaced when used or expired. It would adjust inventory based on user behavior and specification, and would also include a diet management module. While an ideal solution, a SmartPantry is still a predominantly theoretical system. Moreover, this sort of hardware solution has a high barrier of accessibility. A mobile solution such as Pantrymate can be quickly distributed to millions of users with ease. Furthermore Pantrymate is an open source project, free to use for anyone with a smartphone.

Chapter 3

Approach

While a number of shopping coordination and food inventory management applications exist with various functionalities, Pantrymate uniquely combines these two solution strategies in a proactive and relatively automated approach, designed specifically to reduce household food waste. Furthermore, Pantrymate not only combines an inventory and expiration date tracker with an improved grocery shopping experience, it also provides data to increase nutritional awareness. Figure 3.1 shows a diagram of the functional component architecture that allow for this approach as described in further detail below.

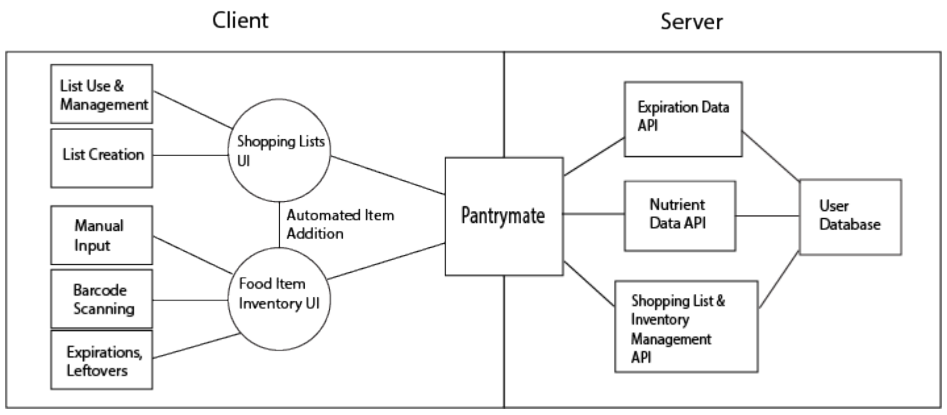


Figure 3.1: Pantrymate's functional component architecture.

3.1 Proactively Targeting Key Factors in Waste

As discussed in Section 2, research shows that the primary factors contributing to household food waste are a lack of proper planning and management in the purchasing, storage, and reuse of food [13]. While there are several other applications in the food storage and reuse management space, Pantrymate is alone in that it also aids in the planning and management of food purchasing. In doing so, it is uniquely positioned to proactively alert users to potential instances of overbuying, a key and under-addressed factor in food waste. When a user creates shopping lists within Pantrymate, the application automatically checks the items in a user's list against current inventory, and notifies if any of the listed food items are already in stock. A similar alert occurs at the actual food acquisition stage, when a user checks off listed items as they are located in the store. Moreover, a similarly proactive approach has been implemented across Pantrymate's design. The application automatically updates data on the expiration dates of food in a user's inventory, and sends alerts in advance of food items expiring. The inventory management interface also has a separate tab that highlights food marked as having been opened or cooked, allowing for a more dynamic expiration date management system (for example, raw chicken kept in the fridge expires almost twice as fast as a cooked piece of chicken), as well as encouraging users to use up leftover food before considering unopened options or buying more.

3.2 Automation to Alleviate Time Constraints

The other key aspect of Pantrymate's approach is a design focused on minimizing user input in order to reduce the time spent inside the application beyond what would already be spent on weekly shopping routines. Research has shown that most households are unwilling or unable to prioritize adjusting their food management behaviors when it involves substantial time constraints [12]. Reducing these constraints is an important step for effectively addressing household food waste. Instead of requiring users to manually input or scan the entirety of their existing food inventories, Pantrymate gathers the majority of its data from users' shopping lists and the items they scan and 'check off' while at the grocery store. Additional barcode scanning functionality automates as much as possible the individual item input and nutrient display process. Although this will lead to an application that is less effective at the time of its initial download and beginning of use, the minimally intrusive user experience will hopefully make it easier for users of all incomes and time constraints to pick up and use the application so they can start reducing their household food waste and better balance their diets. Lastly, while this does not fall under the heading of automation, it is important to note that Pantrymate has been implemented as a cross-browser mobile application in a similar effort to broaden the potential user base, and therefore the overall impact, of this application. Anyone who has access to a smartphone can begin to use Pantrymate to reduce household food waste.

Chapter 4

Implementation

Since Pantrymate is a fully functioning mobile application, the implementation process for developing this application includes user interface (UI) design and development, server structure and API development, database design and integration, and cloud hosting. The following section has broken this implementation process down by the major functional components of the application, subdividing these sections into their respective“front-end” and “back-end” implementation processes. This has been done for simplicity’s sake and is not a direct reflection of the implementation process time line.

The front-end development of Pantrymate includes all user interface design and implementation. Pantrymate’s user interface has been implemented using React Native [26], a JavaScript framework for creating highly performant cross-browser mobile applications. This allows the application to run on both Android and iOS, with JavaScript used as a neutral language to efficiently interact with each system’s native code. In order to jump-start the front-end development process, Ignite CLI and Infinite Red’s boilerplate [27] was used to initialize the application. Ignite CLI is an open-source, free to use React Native application generator. Of the boilerplate plug-ins provided by Infinite Red, Redux [28] was chosen as the default state manager. Without giving excessive detail, it is important to note that Redux allows for extremely efficient management of state data objects within Pantrymate’s

React view components. Therefore, a user's entire inventory can be efficiently maintained and updated directly in the application state as a user interacts with it. This allows for a responsive user experience that is not lost when inventories begin to scale. Note also that React Native Vector Icons [29] was used to provide all of the icons (icon buttons included) used throughout the application. This package was chosen because it is open-source and free to use, with over 3,000 customizable icons that can be rendered natively on both iOS and Android. After initializing the application, additional configuration was necessary to run the application on local simulators. The majority of the front-end development process was run on local simulators, with final testing occurring on hardware devices. Before continuing, it is important to quickly cover some React Native terminology that will be used throughout the remainder of this section. The following has been adapted from a previous independent work report that also used React for application development:

> React Native is a declarative component-based framework. This means that whenever a user interacts with a screen within Pantrymate, they are interacting with a series of components that each contain only the necessary data for that component to properly function. For example, each button is a component that contains the information and data needed to perform whatever function that button is supposed to trigger upon press. The components within Pantrymate are broken up into two categories: "smart" components called containers or screens, and "dumb" components. The "smart" containers are the containers for a given application screen that a user interacts with. They contain the state of all the data being used for a given screen and disperse this data to all of the dumb components within the screen as needed. This data includes data from the server, as well as from the client as the user interacts with the user interface. As noted above, Redux helps store and manage the state of all of this data so that it can be passed within and between components in a screen. As the state of the data within a screen changes, which might happen for example, when a user inputs

a new item on their shopping list, the state of the container screen is updated and React Native automatically re-renders any of the components within the container that use this data [30].

Pantrymate's server has been implemented using Express.js [31], a framework for writing servers in Node.js [32]. Node.js is an event-driven, asynchronous JavaScript runtime. This choice was made with performance in mind, as an asynchronous API allows for scalable efficiency when dealing with large data sets of expiration dates and user inventory information. Google's Firebase [33] has been used to provide a NoSQL database in the hopes of making it easy for any member of the open-source community to further develop this application. Pantrymate is being hosted on a scalable AWS server, provided by Heroku [34]. Figure 4.1 shows this technical stack, with some additional information regarding the specific UI/API components discussed in further detail in the sections below.

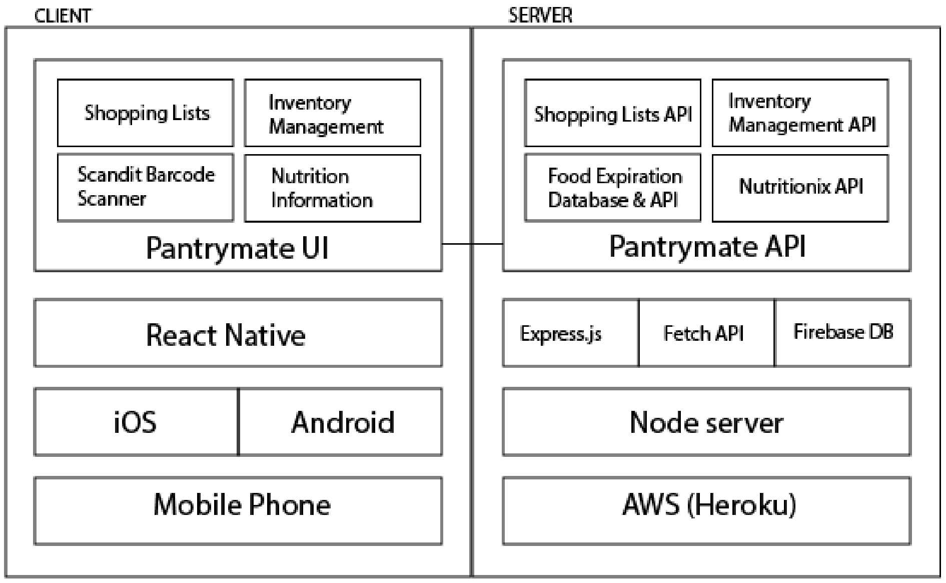


Figure 4.1: Pantrymate's technical stack.

4.1 User Login and Authentication

In order to support user-specific inventory and shopping list management, it was essential to create a unique profile for each user. However, as Pantrymate's approach focuses on reducing the time and convenience thresholds needed to use the application, it was important to avoid requiring users to manually create this profile. To reduce this input requirement, an authentication system was implemented that uses a user's preexisting Facebook or Google profile. Once authenticated, users are automatically logged in upon application launch.

4.1.1 Front-end

From a user interface standpoint, the login and authentication component of this application is fairly straightforward. The user is presented with a prompt to log in and two buttons as options, one for login with Google and one for login with Facebook. These buttons send a request to the server to redirect the user to Google's or Facebook's respective authentication pages. If the user is successfully authenticated, the server responds to the client with a JSON object containing the user's name, as well as a universally unique identifier (UUID) generated specifically for that user's profile. This UUID is then stored using AsyncStorage, an asynchronous storage system that allows for persistent key-value data to be stored locally on the user's device. Storing the UUID in this manner allows for automated user-login, as each subsequent time the user opens Pantrymate on their device, the React component that renders this login screen checks if such a UUID exists in storage, and if found, automatically redirects the user to the main application screen. Figure 4.2 shows the implemented login screen, and Figure 4.3 shows an example authentication interface the user is redirected to when they press on one of the login buttons. The main logo seen on this screen was designed and implemented using Adobe Illustrator.

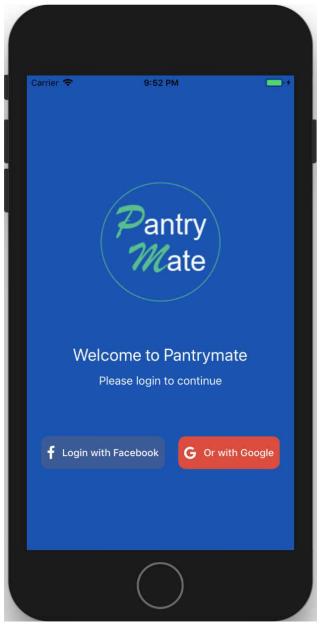


Figure 4.2: Pantrymate login screen.

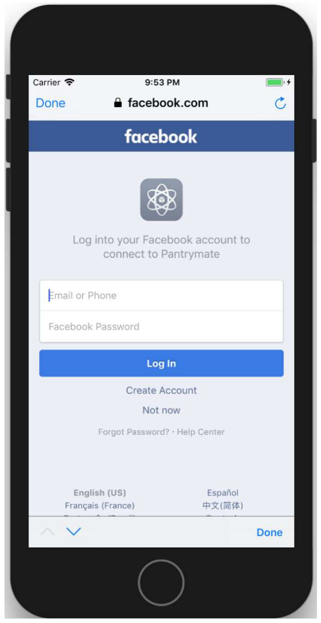


Figure 4.3: Pantrymate auth interface.

4.1.2 Back-end

When the client makes an authentication request to Pantymate's server, the server uses Passport.js [35] to support the requested authentication method (Facebook or Google). Passport.js is a lightweight authentication middleware for Node.js. For authentication with Facebook and Google to be enabled, it was necessary to create developer accounts for both services and have Pantrymate registered as an approved application. Once this registration process was completed, the root domain for Pantrymate's server was approved as a valid OAuth redirect route. Finally, Passport.js was configured to support single sign-in using OAuth for authentication. OAuth [36] is a secure authentication protocol used by both Facebook and Google for authenticating users by providing them with unique user tokens that routinely expire. Once a user has been successfully authenticated following this protocol, Passport.js redirects the login request back to Pantrymate's server with the securely obtained OAuth authentication token. This token is then used to seed an RFC-compliant UUID generator, node-uuid [37]. The generated UUID is then sent back to the client to be stored in Async-Storage for automated login. The UUID is also used to generate a new user reference in

Pantrymate's database. If a user already exists with the UUID, no new reference is created so as to avoid overwriting pre-existing user data. Because the UUID is unique and generated pseudo-randomly with a unique user-specific authentication token used as seed, it is probabilistically safe to assume that no user UUID collisions will occur. This is an important guarantee, as all of a user's Pantrymate data is stored in the database relative to their UUID.

4.2 Shopping Lists

The key to Pantrymate's approach is that it combines solutions for both shopping coordination and inventory management into one application, allowing it to target instances of food waste that occur from over-buying and inefficient use of existing food inventories. Moreover, by engaging with users at the shopping list stage of food acquisition, Pantrymate is able to construct household food inventories without requiring burdensome manual user input. Most shoppers create shopping lists, and most shoppers do so on their mobile phones, so it is natural to do so in Pantrymate where that information will also be used to construct an easy-to-manage inventory. Given that the shopping list feature is so crucial to Pantrymate's approach, it was important to design and implement the shopping list creation and management components of the application such that the user experience is as friendly as that of the default mobile note-taking apps. The following section has been broken down by the sub-feature implementations necessary to develop the overall shopping list creation and management feature. This includes the creation of new lists, the interface for using an active list at the store to check off items as they are being purchased to send to the user's inventory, and the interface for accessing data on past lists.

4.2.1 Shopping List Creation: Front-end

As noted, a strong emphasis on UI design was placed on Pantrymate's shopping list creation feature. Therefore, the first step of this implementation process was to study what makes applications such as Apple Notes and Reminders so user-friendly. For both applications, it was noted that the user can touch anywhere on a blank area within the application's note-pad, and this will automatically trigger a cursor focus on a new text-input component within the list. In order to replicate this behavior, first a state object was constructed within the new list screen's main container to store: 1) the existing data for the list, including the list title and any existing items on the list, and 2) the state of the new item a user might be actively inputting. This data is passed to a modular shopping list component. The shopping list component was implemented such that it could be used for both creating new lists, and also viewing existing lists. Inside this component, all existing list items are rendered. Below the existing list items, a dynamically re-sizing component was implemented within the note-pad to fill any additional list space. This component contains a text-input and a touch handler. The touch handler was implemented such that when a user touch event is registered by the device within the space of the component, the text-input automatically focuses with an active cursor.

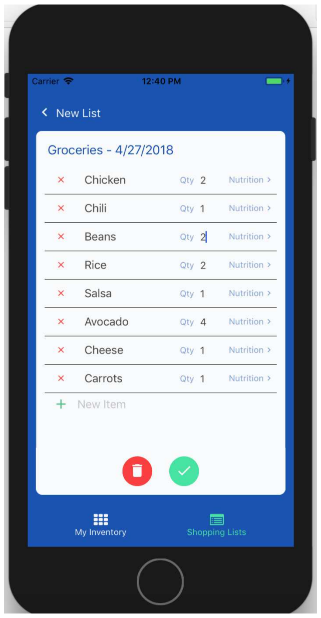
To implement the new item text-input, a custom event handler was created that, upon receiving user-input, updates the state of the screen so that the new item state is equal to the text-input value. When a user submits the value (either by hitting the enter key or by simply touching out of the input – this was another subtle feature noticed while studying Apple's Reminders), the value of the new item state is used to create a new list item, which is appended to the state of the existing items within the list data state. When the list state updates, React Native automatically re-renders the outer shopping list component so that the new item is also rendered. Adding a new item triggers a function that checks the selected item against the existing items in a users inventory. Using the Levenshtein distance algorithm for string similarity comparison [38], the function returns whether the selected item label is particularly similar to any of the items in the user's inventory. If so, an alert dialogue is triggered to notify the user that they might be about to purchase food they already have in stock. This alert allows the user to either continue adding the item, or remove it from the list to avoid overbuying.

The items within the shopping list are rendered using a modular list item component. Each instance of this component renders a single row within the greater list, corresponding to the state of a single item within the list. This component contains a text-input with value set to the state of the item's label. When a user edits this text-input, a custom event handler updates the state of the item label within the outer new list screen container, causing the text-input to re-render and display the updated item label. For new shopping lists, a button in the shape of a red 'X' is also rendered to the left of the item label. This button has an event handler that, when pressed, removes the item from the state of the list items stored in the outer container. This then causes the shopping list to automatically re-render without the removed item. To the right of the label is a quantity text-input, which has been implemented with similar event handling as described above. At the end of the component row is a button that reads 'Nutrition'. Pressing on this button navigates the user to a nutrition screen that displays nutrition facts relative to the item's label. The nutrition screen implementation process will be discussed in further detail in Section 4.3.

Lastly, 'submit' and 'delete' buttons were implemented at the bottom of the list. When triggered by a user press event, these buttons send a request to Pantrymate's server to add the finished list to the user's active lists, or to delete the record of the list, respectively. Before the deletion request is sent, a dialogue has been implemented to confirm the user wants to delete the list before it has been saved. Once either request has been sent and the client receives a successful response status from the server, the user is redirected to the main list management screen. Figures 4.4 and 4.5 show the new list creation user interface.

4.2.2 Shopping List Creation: Back-end

Pantrymate's shopping list creation functionality occurs predominantly on the front-end; the back-end is responsible solely for saving or deleting a newly created list. When a user presses the submit button on a new list, a request is sent to Pantrymate's server that contains

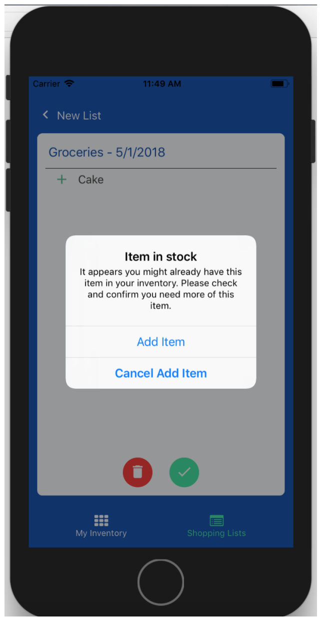


Groceries - 4/27/2018

× Chicken	Qty	2	Nutrition ›
× Chili	Qty	1	Nutrition ›
× Beans	Qty	2	Nutrition ›
× Rice	Qty	2	Nutrition ›
× Salsa	Qty	1	Nutrition ›
× Avocado	Qty	4	Nutrition ›
× Cheese	Qty	1	Nutrition ›
× Carrots	Qty	1	Nutrition ›

+ New Item

Figure 4.4: New list creation UI.



Item in stock

It appears you might already have this item in your inventory. Please check and confirm you need more of this item.

Figure 4.5: Item in-stock alert triggered.

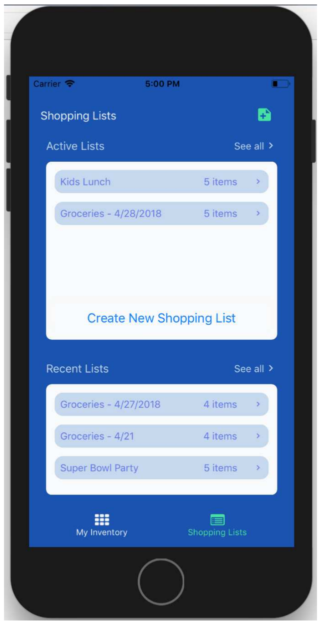
all of the list information stored in the list data state. This includes all of the items within the list, the list title, the status of the list (in the case of a new list this is always ‘active’, as opposed to ‘recent’ used for archived lists), and the date it was created. Additionally, the user UUID responsible for the list creation request is sent so that the list can be stored under the correct user in the database. When the server receives a save list request, it sets the list status flag, adds a list UUID key-value pair to the list data object for later referencing, and stores the list as a JSON object within the database under the appropriate user profile. For a delete request, it looks up the appropriate list by user and list UUID, and removes any record of it from the database. The server then responds with a response success, and in the response body sends the updated list information for the user. This is used to immediately update the state of the user’s lists within the list management screen. In doing so, minimal delay time occurs between the user redirect to the main list management screen and the rendering of the appropriate user list information. Because Pantrymate’s shopping lists are stored as JavaScript objects, they can efficiently handle frequent user updates by directly referencing the appropriate keys.

4.2.3 Shopping List Management: Front-end

Pantrymate's shopping list management occurs through a main list management screen. The key component of the approach behind the shopping list feature is to be able to proactively alert user's if they are going to purchase something they might already have in stock. Therefore, when designing the list management screen, it was important to highlight a user's lists that are ready to be used while at the store. To do so, lists that have yet to be used for inventory submission are categorized as 'active'. Given that past lists can provide helpful nutritional information and can also be conveniently re-used if certain items are frequently repurchased, an interface was also implemented to keep track of 'recent' lists (lists that have already been submitted to the user's inventory). The first step when implementing the shopping list management screen was to create two subsections within the screen to display the user's active and recent lists. These subsections were implemented as wells, within which are rendered 'previews' for the first few lists of the user's active/recent lists. Of course it is possible that a user will have created more lists than fit within half of a mobile device's screen, therefore a button above each well labeled 'See all lists' was implemented. When pressed, it navigates the user to a screen that displays previews of all of the respective active/recent lists. The display all recent/active list screens were implemented using the same list preview component described above. Figure 4.7 shows the 'See All Recent Lists' screen.

The list previews were implemented as modular list selector components. Each list selector is passed the data pertaining to one list. The list selector displays the list title and the number of items on the list. The entire component is wrapped within a touch handler. When pressed, the preview navigates the user to a shopping list display screen. The list UUID is passed between components during this navigation, and is used in order to determine the corresponding list data to be displayed. The active/recent list display screens are described in greater detail below.

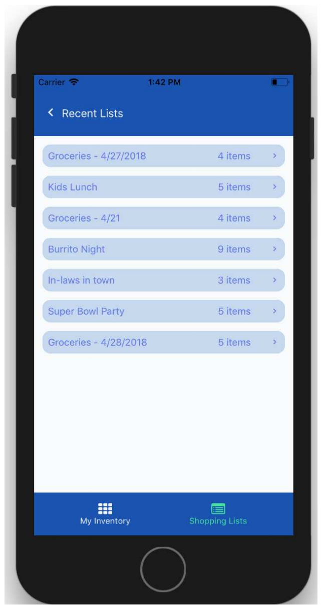
Additionally, two main methods of creating a new list are available from within the main list management screen. The first of these is a 'Create New List' button, situated at the bottom of the active lists well. The second is a new list icon button located at the top right corner of the header of the screen. Both of these have been implemented so that when pressed, the user is navigated to the new list creation screen as described above in Section 4.2.1. Figure 4.6 shows the list management screen.



Kids Lunch 5 items
Groceries - 4/28/2018 5 items

Groceries - 4/27/2018 4 items
Groceries - 4/21 4 items
Super Bowl Party 5 items

Figure 4.6: Main list management UI



Groceries - 4/27/2018 4 items
Kids Lunch 5 items
Groceries - 4/21 4 items
Burrito Night 9 items
In-laws in town 3 items
Super Bowl Party 5 items
Groceries - 4/28/2018 5 items

Figure 4.7: See All Recent Lists screen.

4.2.4 Shopping List Management: Back-end

Similar to that of shopping list creation, the implementation of Pantrymate's shopping list management feature was predominantly a front-end development process. For shopping list management, the primary role of the server is to retrieve the user's shopping list data from the database and send this to the client. As the shopping list management screen mounts, a request is sent to the server containing the user's UUID to retrieve all of the user's lists. The server looks up the user's list data using their UUID, and responds to the client with a JSON string in the response body containing an array of list objects. These are stored on the client state using Redux, and are then accessible throughout the various list screens.

4.2.5 Active and Recent List Displays: Front-end

When a user presses on one of the active/recent shopping list previews, the user is routed to the selected list display screen. The active list interface was designed to be used when a user is in the process of grocery shopping, therefore it was important for active lists to include an interface for checking off items that have been added to a user's shopping cart. Active lists should also be updateable as new items are picked up in the store. Furthermore, a key element of Pantrymate's approach is that active lists can be used to automatically construct user food inventories, and, more importantly, they are able to alert users to potential instances of overbuying by comparing the items on the list against existing user food inventories. Finally, active lists are a good potential point in the household food acquisition and management process to raise users' nutritional awareness, therefore the active list interface was designed to include a food item nutrient content display.

The first step in implementing the active list display was to adjust the previously developed shopping list component designed originally for new list creation, to allow for modular re-use of the basic list interface design. The interface for adding new items, adjusting existing items (both labels and quantities), changing the list title, and viewing item nutrient information remains the same as described above in Section 4.2.1. The interface for list deletion also remains the same. However, when the shopping list component is passed a flag marking it as active, the item deletion buttons are replaced with toggle-able radio buttons. These buttons allow the user to check off items while shopping. When pressed, a function is triggered that compares the item against current inventory items, similar to the one described above in Section 4.2.1. Upon user confirmation, the radio buttons are re-rendered and become visually checked. The press handler also updates the state of the relevant list item such that it is marked as having been selected. This is ultimately used to determine which items are added to the user's inventory upon list submission.

Lastly, the list submission button was implemented. When pressed, an alert is triggered that confirms with the user that all selected list items are about to be purchased, and prompts the user to add any unlisted items. Upon confirmation, the items are sent to the server for an inventory update, and the user is routed back to the main list management screen. Figures 4.8 and 4.9 show the active list interface.

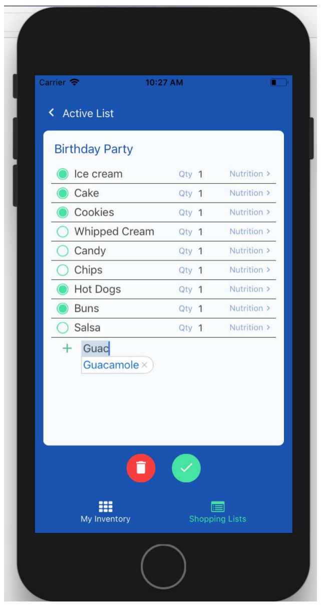
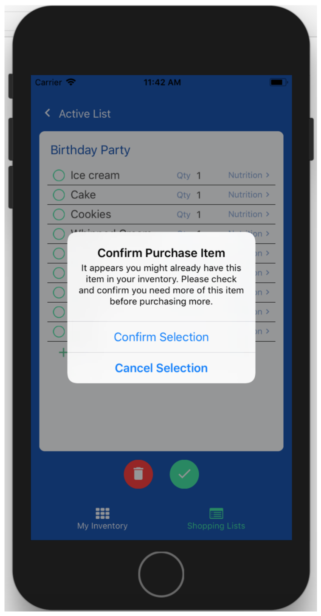


Figure 4.8: Active list UI.



Confirm Purchase Item

It appears you might already have this item in your inventory. Please check and confirm you need more of this item before purchasing more.

Confirm Selection

Cancel Selection

Figure 4.9: Active list overbuying alert.

When an active list has been submitted by the user, it is updated on the server and marked as ‘recent’. The recent list interface also uses the modular shopping list component to display a snapshot of the list data as it was upon submission. This includes all checked items. Recent lists cannot be updated (i.e. unchecked items cannot be checked, items cannot be changed, etc...). However, nutrient information for each item is still available, and recent lists can be deleted from the database archive. Lastly, a list reactivation interface was implemented in place of list submission. When the user presses on the list reactivation button, a confirmation dialogue is triggered. Upon confirmation, the list is updated on the server as ‘active’, the user is routed to the main list management screen, and the list now appears under the active lists portion of the interface. Figures 4.10 and 4.11 show the recent list interface.

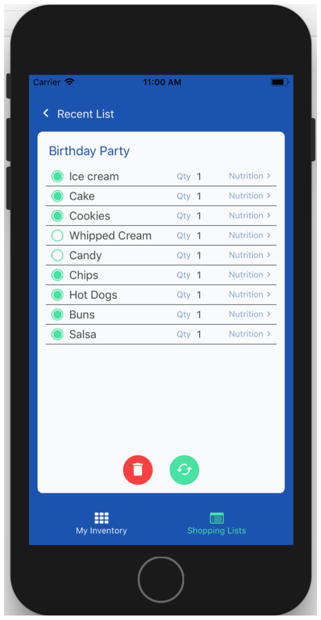
4.2.6 Active and Recent List Displays: Back-end

The first step in implementing the server for active and recent list displays was to develop a method for returning the appropriate list data given the desired list to be displayed. This was done via list UUIDs. When the user selects a list by pressing on a list preview (either within the list management screen or the see all active/recent lists screen), the selected list's UUID is passed through the navigation router to the active/recent list container component. Before this component mounts, a request is sent to the server containing the selected list UUID and the stored user UUID. Upon receiving this request, Pantrymate's server looks up the desired list using these UUIDs, and sends a response to the client containing the desired list data as a JSON string in the response body.

Pantrymate's server is also responsible for the active list item submission and reactivation of recent lists. For list reactivation, the list status is set as 'active' within the list data object, and then the appropriate list data reference is updated in the database. For item submission, the client sends a request to the server containing the user UUID and the list data to be submitted. First, the server filters out all items from the list that are not selected. Then, the server appends these items to the current inventory item array in the user's inventory database. Note that before this inventory update occurs, the list items are passed to the expiration data API described in further detail below in Section 4.4.7. Lastly, the server also handles list deletion. For list deletion, the client simply sends the server the appropriate user and list UUID, and the server removes the corresponding reference from the database.

4.3 Nutrient Content Display

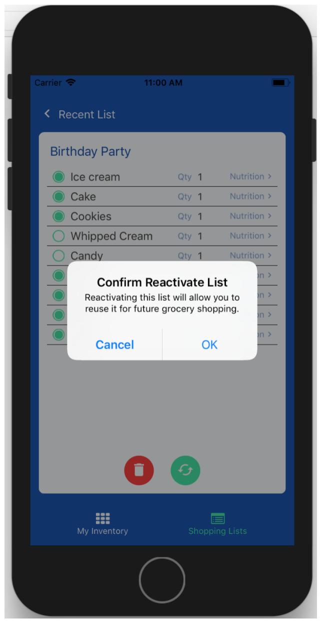
A nutrient content display is used by both the shopping list and inventory management portion of the application, and therefore will be briefly discussed before continuing. The UI for Pantrymate's nutrient content display component is an original design and implementation. The server uses Nutritionix's [39] food item database and API to provide the nutrition



Birthday Party

●	Ice cream	Qty 1	Nutrition ›
●	Cake	Qty 1	Nutrition ›
●	Cookies	Qty 1	Nutrition ›
○	Whipped Cream	Qty 1	Nutrition ›
○	Candy	Qty 1	Nutrition ›
●	Chips	Qty 1	Nutrition ›
●	Hot Dogs	Qty 1	Nutrition ›
●	Buns	Qty 1	Nutrition ›
●	Salsa	Qty 1	Nutrition ›

Figure 4.10: Recent list UI.



Confirm Reactivate List

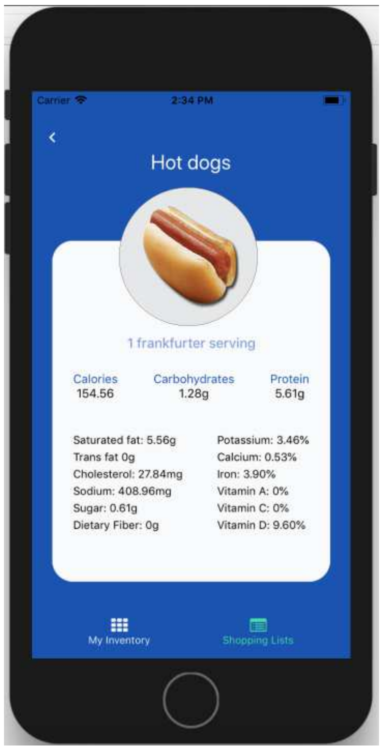
Reactivating this list will allow you to reuse it for future grocery shopping.

Figure 4.11: Reactivate recent list dialogue.

data. Nutritonix was chosen for this service as it is the largest grocery database available, with 92% coverage for U.S. food products. A free-to-use 'Hacker' plan and API key were acquired in order to access Nutritionix's large volume of nutrition data. It should be noted that this API key is rate-limited, and thus Pantrymate's current implementation is limited in its ability to serve a large number of users.

4.3.1 Nutrient Content Display: Front-end

When a user presses on the 'Nutrition' button on any food item in a list/inventory management tab, the user is routed to a nutrient content screen for the corresponding item. Before the container component mounts, a request is sent to the server for the necessary nutrition data. Once the client receives this data, the nutrient content display is rendered. When designing this display, research was done into the primary nutrients that are emphasized in popular nutrition management applications. Across the board, calories, carbohydrates, and proteins were considered the most important nutrients to display. Therefore, when implementing the nutrient information screen, calories, carbohydrates, and proteins were highlighted in the center of the component. Above these nutrients, a thumbnail of the food item and the serving type and size (if available), have also been prominently displayed. For the remainder of the nutrition facts card, additional research was conducted to determine which other nutrients, vitamins, and minerals are considered most important to be aware of in a balanced diet. Ultimately, it was decided to include nutrient information for the main twelve of these: saturated fat, trans fat, cholesterol, sodium, sugar, dietary fiber, potassium, calcium, iron, Vitamin A, Vitamin C, and Vitamin D. Lastly, it is worth noting that further research was necessary in order to determine the appropriate units in which to display these nutrients (for example, iron is typically measured in milligrams, sugar in grams, Vitamin A in percent of recommended daily value). Figure 4.12 shows the nutrient information screen.



1 frankfurter serving

Calories	Carbohydrates	Protein
154.56	1.28g	5.61g

Saturated fat: 5.56g	Potassium: 3.46%
Trans fat 0g	Calcium: 0.53%
Cholesterol: 27.84mg	Iron: 3.90%
Sodium: 408.96mg	Vitamin A: 0%
Sugar: 0.61g	Vitamin C: 0%
Dietary Fiber: 0g	Vitamin D: 9.60%

Figure 4.12: Pantrymate's nutrient information screen.

4.3.2 Nutrient Content Display Information: Back-end

As noted above, Pantrymate's nutrient information data is provided by Nutritionix. Nutritionix supports both common and branded food nutrient look-ups, with over 26,000 common foods (ex. 'chicken', 'pasta') and over 650,000 branded products [39]. When building shopping lists, users typically to write down common food names, so it was crucial to use by default Nutritionix's common food database. Before the nutrient content display container mounts, a request is sent to Pantrymate's server with the desired item's common name. The server then uses this label to search Nutritionix's database for the best match. Once the best match has been determined, the Nutritionix food ID is obtained from the database, and this is then used to send an API request for the desired nutrition information. If the Nutritionix server responds successfully with a nutrition information object, this is then parsed by Pantrymate's server in order to be efficiently used by the client.

Implementing this data parsing function required asynchronous object construction, so as to ensure that only and all of the desired nutrient data is returned by the server response. This was done using JavaScript's `Promise` API and the `await` operator. Once the nutrient data object has been constructed, it is sent as a JSON string in the response body to the client. Additionally, item look-up by UPC code was implemented in order to provide nutrient content information for items added via bar code scanning. Given the UPC code, the Nutritionix API supports a direct nutrient search. This was implemented by first obtaining the scanned UPC code for the item (this process is described below in Section 4.4.6), sending this to the Nutritionix API, and then parsing and returning the retrieved data using the same asynchronous parsing function as for common foods.

4.4 Inventory Management

The second major piece of Pantrymate's functionality is providing users with easy-to-use food item inventory management. The primary purpose of the inventory management interface is to alert users when food items are about to expire, and to help them keep track of what they have in stock in an attempt to avoid instances of overbuying. In addition to a basic interface for displaying a user's food item inventory, the implementation process for Pantrymate's inventory management also included developing an interface for manual item addition, item addition via bar code scanning, and an API for automated food item expiration data.

4.4.1 Food Item Inventory Display: Front-end

When designing the inventory display component, the primary concern was to ensure that expiring foods are highlighted in order to alert the user before they go to waste. As leftovers are another frequent source of household food waste, the ideal inventory display would also be able to highlight leftover food. With this in mind, the most efficient interface option was determined to be a tabular interface, where the main tabs are 'Expiring', 'Leftovers', and 'All' (everything else). As most cooked foods expire within four days, it was decided that the threshold for an expiring food item is one that is set to expire in three days or fewer. This would ideally provide users with ample time to realize that they should consume their expiring food. The actual process for determining food item expiration is discussed in detail below in Section 4.4.7.

In order to handle leftover food management, it was necessary to implement an interface for users to mark certain food items as having been cooked. After testing several potential interface designs, the final implementation uses a 'swipe out' menu. A swipe out menu allows the user to swipe left along any of the food item components within the inventory display, and the component will slide over to reveal a button menu. This interface is one familiar to users who use Apple's Mail or Messages app (among many others). For Pantrymate's food item component swipe out menus, a 'Delete' button and, if available, a 'Cook' or 'Open' button are contained within the menu. When a user presses on the 'Delete' button, the specified item is removed from the inventory state stored in the inventory display container.

This causes an immediate automatic re-rendering of component, and an additional request is sent to the server to update the user's inventory data in the database. Pressing on either the 'Cook' or 'Open' buttons within the menu will mark the specified food item as belonging to the 'Leftovers' tab. Similar to item deletion, this update to the food item data occurs both in the client container state to allow for an immediate re-render, as well as in the database through a request to Pantrymate's server. Note that the reason 'Cook' and 'Open' are both menu options is because it was most efficient to consider packaged food that has been opened as leftovers, because frequently the time until expiration of packaged food drastically decreases once opened, and it is important to be able to alert users to this potential expiration acceleration before the food goes to waste. The reason why these buttons are not always available is because when an item is cooked or opened, it is necessary to have data for the new expiration date of the item. This will be discussed in greater detail in Section 4.4.7. However, finding a reliable source of food item expiration data proved exceedingly difficult, and ultimately it was necessary to implement Pantrymate's expiration API and database from scratch. As a result, the expiration data available is exceedingly limited, especially when trying to provide data for specific foods both in their raw/unopened state, and in their cooked/opened state. Therefore, only foods that have available data for both their 'default' and 'leftover' state have the option to be marked as leftovers.

Regarding the implementation of the actual food items as they are displayed within the inventory management screen, a similar approach was taken to the shopping list implementation, where a modular inventory item component was developed and used to render all of the items throughout the interface. First, when the inventory management screen container is mounted, the user's inventory data is retrieved from the server and stored in the container state. This data includes the state of all of the food items in the inventory which are stored as an array of food item objects. Each food item object contains the item's label, its expiration data, a flag to determine if it was added via UPC, the date it was purchased (this is set automatically if an item is added through the shopping list submission process, and manually by the user if added through the manual item addition interface described below), and the date it was cooked/opened (if applicable). Similar to the items in a shopping list, the inventory item component renders an individual food item as a single row of data.

First the label of the food item is displayed, followed by the item's expiration data. When designing the expiration data display, an interesting problem arose when deciding how to handle variable time units, as well as the different methods of inventory storage. To further explain this variability, consider the difference in how long a chicken breast might last when kept in one's refrigerator as opposed to one's freezer. Consider also that this chicken will last for no more than 120 days either way, unlike a can of beans, which might last several years if left unopened. The expiration data provided by the server is structured such that each food item's expiration data contains a separate data object for the item's 'default', 'cooked', and 'opened' expiration, within which are data objects for each of the primary methods of storage: the pantry, refrigerator, and freezer. First, the client determines which of the expiration objects to consider for rendering by checking whether a flag has been set in the item's data object indicating whether the item has been cooked or opened. For all of the three possible categories where expiration data is available, the inventory item component will display this data alongside an icon representing the appropriate category for the data (a snowflake for freezer, a fridge for refrigerator, a shelf for pantry). If no expiration data is available, the component will simply render 'Data unavailable'. To handle variable time units, it was ultimately decided that any time period under one year will be displayed in terms of days, and any expiration date over one year in terms of years, rounded to one decimal place. All expiration data is calculated on the server in units of days, therefore an additional function was implemented to convert the UI labels as desired. Lastly, each inventory item component contains a nutrition button identical to that used by the shopping list item components, the implementation of which is described above in Section 4.3. Figures 4.13 and 4.14 show the inventory management display in a couple of the possible states outlined above.

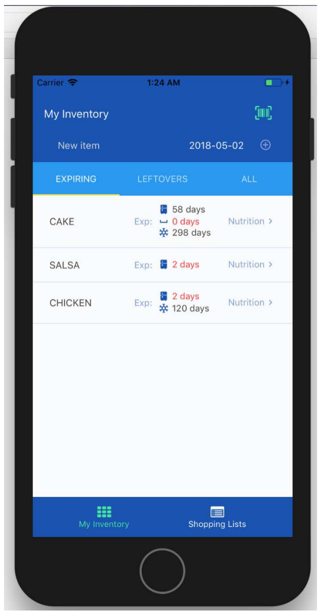


Figure 4.13: Inventory management UI.

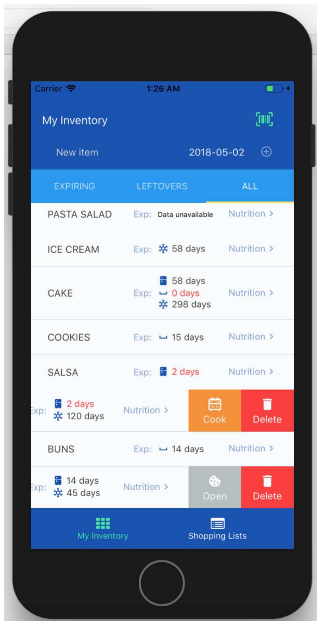


Figure 4.14: Swipe out menu options.

4.4.2 Food Item Inventory Display: Back-end

For the food item inventory display, the primary role of the server is to provide the user's inventory data. When the inventory management container mounts, the client sends a request to the server containing the user's UUID. The server then retrieves a reference to the specified user's inventory data from the database. This is then sent in the response body as a JSON string, where all of the user's inventory items are stored within an array. An array was chosen for easy sorting and filtering, and also because it allows for iterable rendering within the interface. There is no need to ever directly access individual inventory items by an ID or key, therefore there was no efficiency benefit in storing the inventory items within a JavaScript object.

Additionally, it was necessary to implement a function on the server to handle database storage of any state changes that are made by the user within the inventory management interface. These changes occur when a user deletes, cooks, or opens an item. Because these state changes already occur within the inventory management container on the client to allow for immediate re-rendering, a simple update function was sufficient to handle inventory updates within the database. Whenever the user changes the state of their inventory data, a request is sent to the server with the user's UUID and the new state of the user's inventory. Pantrymate's server then replaces the previous reference to the user's inventory in the database with the updated one.

4.4.3 Manual Item Addition: Front-end

Pantrymate's approach focuses on automating the process of constructing a user's inventory by linking it to the shopping list phase of food acquisition and management. However, the trade-off of this approach is that Pantrymate has no data on the state of a user's food item inventory prior to the application's initial use. In order to address this issue, a manual input for adding items to a user's inventory was included in Pantrymate's inventory management design. Furthermore, a manual input allows users an easy method by which to add individual items they may have forgotten to include or check-off when adding items via shopping lists.

Implementing an interface for manual item addition required both a text input for the item label, as well as a method for users to select what date the item was purchased. The latter is necessary in order to calculate the expiration data for the item. The text input was a straightforward implementation using a text handler to update the new item state in the container, similar to the method used for adding items in the shopping list interface seen in Section 4.2.1. For the date of purchase interface, an open-source React Native Date Picker component [40] was integrated into the UI. This date picker component was chosen in order to provide a familiar date selection interface for both iOS and Android users. By default, the item's purchase date is set to the current date using JavaScript's `Date` API. Figures 4.15 and 4.16 show the manual item addition interface.

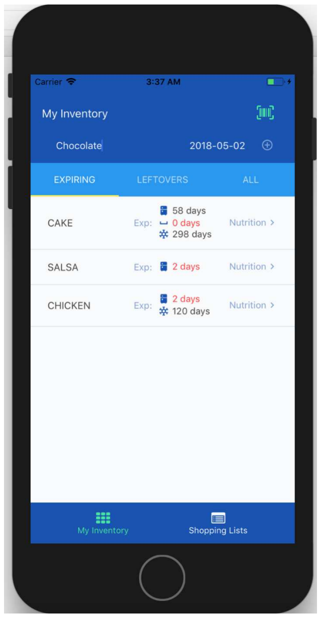


Figure 4.15: Manual item addition input UI.

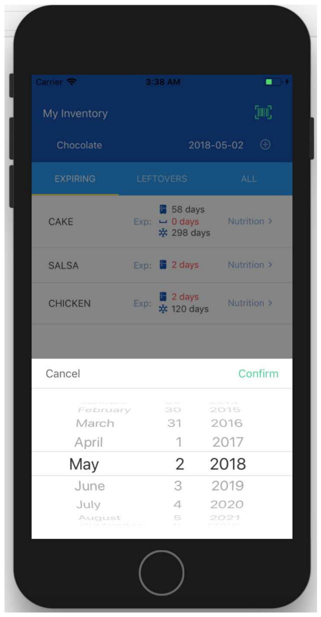


Figure 4.16: Manual item addition date picker.

4.4.4 Manual Item Addition: Back-end

When an item is manually added by the user to their inventory, a request is sent to the server containing the user's UUID and the new item data. The server first performs the necessary operations to retrieve all of the item's expiration data, then appends the new item with its expiration data to the inventory items array in the user's inventory data. Finally, the server updates the inventory reference in the database, and assuming no errors have occurred, sends a status success response to the client.

4.4.5 Barcode Scanning: Front-end

In order to best implement Pantrymate's automation-focused approach, the application design included barcode scanning functionality so that users could simply scan products in their home or at the store, and these items would then be automatically added to their inventory. After researching the available barcode scanning solutions for mobile phones, Scandit [41] was selected as the best possible option. Scandit provides an industry standard barcode scanner for mobile phones with unrivaled performance, and offers a solution specifically for React Native integration. After contacting a member of Scandit's team, access to a community version of their solution was acquired for the purpose of this thesis. Note that before releasing Pantrymate for public consumer use, a new Scandit plan would need to be purchased. The API key has also been excluded from the open-source repository for security purposes. Therefore, any future contributors will need to obtain access to Scandit as deemed necessary. This holds true as well for the Nutritionix API, and the Google/Facebook OAuth authentication keys.

Once Scandit's React Native solution was acquired, the bulk of the barcode scanning implementation process involved dealing with integrating the barcode scanner into the existing application. Integrating the Android SDK was fairly straightforward, but various errors kept occurring while trying to integrate the iOS SDK with Pantrymate's code base. Unfortunately, Scandit does not provide a particularly comprehensive documentation, and it was a long and difficult process debugging the integration. After successfully integrating both the Android and iOS SDK, a barcode scanning screen was implemented. To navigate to Pantrymate's barcode scanning screen the user simply presses on the barcode button located in the top right corner of the main inventory management screen. The barcode scanning interface consists of Scandit's main barcode scanner component, below which are two buttons to turn the scanner on and off.

When the scanner is turned on, it automatically registers any barcode that is focused on within the view frame of the mobile phone's camera. To access the devices camera, it was necessary to implement a dialogue to confirm user camera permissions. A custom event handler was then implemented to catch any registered barcode scans and extract the scanned product's UPC code if available. The UPC code is then sent to the server to check if the product is included in Nutritionix's UPC product database. An alert dialogue is then triggered notifying the user if the product has been successfully identified. If so, the user is provided with an option to automatically add the item to their inventory. If the user selects to add the item to their inventory, the barcode scanner component is turned off and the user is routed back to the main inventory management screen where the scanned item will be displayed. Figures 4.17 and 4.18 show Pantrymate's barcode scanning interface. Note these figures are screen shots taken directly from an iPhone. This is because it is impossible to simulate a mobile phone's camera, so all development of the barcode scanning interface required testing the application directly on a hardware device.

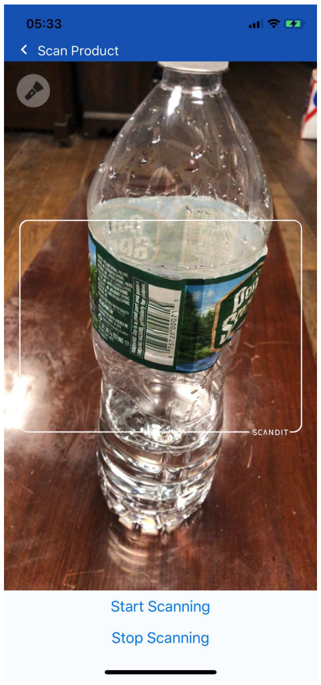
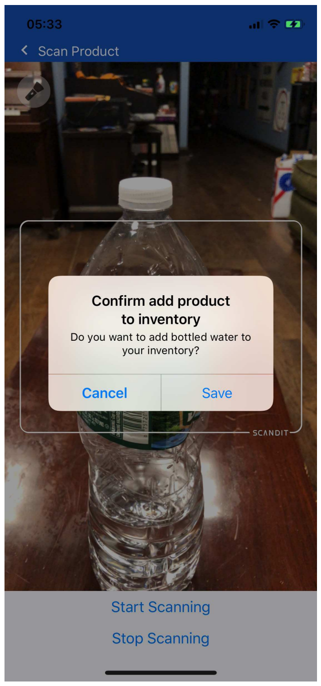


Figure 4.17: Barcode scanner UI.



Confirm add product to inventory

Do you want to add bottled water to your inventory?

Figure 4.18: Successful product scan alert.

4.4.6 Barcode Scanning: Back-end

Once the client successfully obtains a product UPC code via Scandit's barcode scanner component, the UPC code is then sent in a request body to Pantrymate's server to see if the product can be located in Nutritionix's database. To do so, the server sends a look up request to the Nutritinoix API with the given UPC code as the query parameter. If successfully located, the product data object is sent as a JSON string in the response body to the client. This item data object includes the product label, which is crucial for display, as the barcode scanning interface no longer relies on the client to provide inventory item labels. If the item is added to the user's inventory, then the item data object is sent back to the server along with the user's UUID, where the item is updated using Pantrymate's expiration API to include the relevant expiration data (if available). Once this process successfully completes, the item is then appended to the user's inventory items array, and the appropriate database reference is updated.

4.4.7 Food Item Expiration

The final component of Pantrymate's inventory management functionality is the food item expiration database and API. When originally designing Pantrymate's functional architecture, it was assumed that a comprehensive database of food item expiration data would be available for use. While such a database does exist to a certain degree [42] [43], the owners of this data have not made it available to the public, nor have they provided any sort of API through which developers can acquire even limited access. Given that providing users with automated alerts regarding the expiration status of items in their inventory is a crucial component of Pantrymate's approach, it became necessary to implement an original food item expiration database and API. Unfortunately, implementing this database and API such that it would achieve ideal data coverage was far beyond the scope of this thesis, especially considering Nutritionix provides data on hundreds of thousands of different food items. However, the foundations of this feature have been implemented in order to demonstrate Pantrymate's potential functionality.

The first step in developing any database is gathering data. Unfortunately, as previously mentioned there is no definitive source of food expiration data available for easy gathering. The two main organizations that provide reliable food expiration data are Eat By Date [42] and Still Tasty [43], however their data is only accessible via clicking through their website. As a result, the first strategy for gathering the desired data was to try to implement a web scraper to extract as much data as possible from these sites. Unfortunately, all of Eat By Date's data is not only non-searchable within the site, but it is actually displayed for users within tables and image files. Consequently, retrieving this data would have likely required implementing a relatively complex OCR screen scraper. The attempt at scraping Still Tasty's data was a bit more successful, and eventually data on about 200 food items was retrieved. However, this data ended up being frustratingly unhelpful, as it did not account for variability in expiration when storing in a freezer as opposed to the pantry or refrigerator.

After further research, a large, unlicensed spreadsheet of food item expiration data was eventually found [44]. While this data has not been verified as entirely accurate, many comparisons alongside Eat By Date's data indicated that this spreadsheet is relatively reliable. Importantly, the spreadsheet also included food expiration data for all three of the aforementioned storage locations. As this spreadsheet appeared to be the best data source available, it was then converted from CSV to JSON using an online CSV to JSON conversion tool [45]. Once in JSON format, the raw data was cleaned to make it usable for Pantrymate's expiration data API.

The raw JSON data was exceedingly erratic, containing a vast number of duplicate entries with conflicting data, significant discrepancies in the format of how items were labeled and how their expiration data was structured, a number of inconsistent data points on items that were cooked, opened, or packaged in various ways, as well as a large volume of simply extraneous data (such as information on diapers, which do not in fact expire regardless of whether you store them in your freezer or your pantry). As a result, it was impossible to write a script that could reliably clean and parse the data. An automated script not being viable, the only method remaining was a manual clean. The raw 20,000 line data file was eventually condensed to a clean 2,300 line JavaScript file, containing a few hundred food items with their expiration data. This file also includes a number of data points on cooked/opened items. Each food item data object has been structured as follows:

```
{
 'item': 'chicken',
  'pantry': '',
  'refrigerator': [2.0, '2 days'],
  'freezer': [120.0, '120 days']
}
```

Note that the expiration data for each storage location is structured as an array. The first index in the array is the number of days the item has from purchase until expiration, and the second index is the corresponding label for this data in its appropriate units. This was calculated on the server to provide an easy method for the client to display the expiration data within the inventory management screen. If the storage location has no data (like the pantry in the above example), this likely indicates that the item should not be stored in that location. In the example above, raw chicken should never be left to sit in a pantry. Once the basic food expiration database had been cleaned and constructed, the remainder of the implementation process focused on developing an API to match food items in a user's inventory to their corresponding expiration data (if available), to asynchronously construct new food item objects containing the desired expiration data, and finally to provide an automated method of updating the expiration data for food items such that the time until expiration decreases with each passing day.

Whenever an item is added to a user's inventory, the server automatically sends the item through the food expiration API. The implementation for matching inventory items to their expiration data uses the Levenshtein distance algorithm to determine the closest possible match within a two character discrepancy threshold, with additional modifications to handle cases in which an item label is contained within another item label. These modifications were added to allow for more leniency when dealing with users who might abbreviate their food items, or frequently misspell/pluralize them. After searching through the available expiration items, the data for the closest match (assuming it meets the threshold requirements) is used to construct a new inventory item. Both the database search and item data object construction process are entirely asynchronous, and therefore the implementation for this portion of the API required several asynchronous callback functions. These functions use

JavaScript's `Promise` API and the `await` operator to ensure that they do not return before the full search and construction processes are completed. Assuming an expiration data match is found, a simultaneous object key look-up is used to determine whether there is available cooked/opened expiration data for the matched expiration item (as part of the data cleaning process all cooked/opened expiration data has complementary default expiration data under the same item label). The final item data object, now including all relevant expiration data, is ultimately what is provided by the server to the client to be rendered by the inventory management display as described above in Section 4.4.1.

The final piece of Pantrymate's implementation process was developing an API for daily automated expiration data updates. The key to implementing this feature was constructing the original inventory item data objects such that they include when every item was purchased and when every item was cooked/opened (if either are available options). With this data, assuming the user buys predominantly fresh ingredients, relatively accurate estimations can be made for each items expiration date. On a macro level, the API looks at the total number of days before an item expires in each possible storage location, and subtracts from this the difference between today's date and the item's purchased/cooked/opened date. Implementing this function ended up being trickier than expected, as operating on JavaScript's `Date` API objects requires first parsing the date strings, then converting the parsed date string into UTC millisecond counts, normalizing for potential time zone differentials, and finally converting this millisecond count back into the desired number of days. An additional function was also implemented to update the expiration data labels such that the client could easily render the data in the desired display format. Note that these functions were also implemented to run asynchronously over all items in the user's inventory, across all possible expiration data sets (purchased/cooked/opened) and each of the possible storage locations (freezer/fridge/pantry). While this adds a level of complexity to the implementation process, it ultimately allows for these updates to occur frequently with great efficiency, and thus Pantrymate's expiration data API is both performant and scalable.

Chapter 5

Evaluation

Directly evaluating Pantrymate's effectiveness at reducing household food waste would require an extensive fieldwork study to track a test group of households' waste for a period of time before and after they start using this application. Such a study not being possible within the limited time-frame of this thesis, a quantitative evaluation process instead quantified the application's effectiveness at alerting users to potential instances of waste. While it is impossible to measure how these alerts effect physical user behavior, this evaluation method provides a benchmark for measuring how significant the targeted factors are in their contribution to household food waste, and demonstrates Pantrymate's contribution in addressing them.

5.1 Data and Metrics Used for Evaluation

For Pantramte's evaluation, an open sourced dataset of over three million online grocery-shopping orders was acquired through Instacart [46]. This data was then used to simulate Pantrymate's behavior for 100 randomly selected users. Instacart's dataset covers user grocery shopping orders throughout the entire year of 2017, and thus provides data over a sufficient time frame to simulate and evaluate Pantrymate's use.

The dataset, as it is made available, has separate CSV files for user orders and the actual products purchased for each order. In order to simulate an actual user's behavior, it was necessary to use these separate datasets to construct a single dataset with full user grocery shopping profiles containing all of a user's orders, the time between each order, and the food items acquired within each order. As noted, the datasets in question were on the scale of millions of entries, therefore Stata (a data management tool provided by the University) was used to handle the necessary data manipulation. First, 100 user IDs were selected. All orders placed with these user IDs were then isolated from the larger dataset. A merge by order ID was then used to link each of the relevant product IDs to the orders in which they were acquired and to the user placing the order. A final merge on product ID provided data on the actual products being ordered. The final constructed dataset was then converted from a Stata file into a standard CSV file for analysis. The full CSV file contains 12,964 rows of data, and is therefore far too large to include in this report. However, an example of a grocery shopping profile for a single user is included in Appendix B. For reference here, the data corresponding to a single order for one user has been included below in Table 1:

Order ID	Order Number	Days Since Prior Order	Product ID	Reordered
2398795	2	15	10258	0
2398795	2	15	196	1
2398795	2	15	13032	0
2398795	2	15	12427	1
2398795	2	15	26088	1
2398795	2	15	13176	0

Table 1: Compiled Instacart user data for one order.

Each row represents one product ordered by the user. The order number refers to the order's place sequentially within the user's order history. Days since prior order indicates the number of days elapsed between an order and the previous one placed by the user. The reordered column is set to 1 if the product has been previously ordered, and 0 if not.

Using the 100 user grocery shopping profiles described above, the following metrics were used for Pantrymate's evaluation:

- Percentage of total products ordered are reorders.
- Average number of subsequent alerts per user to potential instances of overbuying.
- Percentage of unique products ordered covered by Pantrymate's expiration database.
- Average number of subsequent alerts per user to potential instances of expiration with current expiration coverage.
- Average number of subsequent alerts per user to potential instances of expiration with improved expiration coverages.

These metrics evaluate both the significance of the key food waste factors targeted by Pantrymate, and the application's effectiveness at alerting users to them. Additional metrics for the application's projected behavior with an improved expiration database have been included to demonstrate Pantrymate's potential for increased effectiveness given future work.

5.2 Evaluation Results

Analyzing the evaluation data, it was calculated that among the 100 user grocery shopping profiles, a total of 1,418 orders, and 12,963 total food items were purchased. Of these food items, 7,053 were reorders. The average time between orders was 12.5 days. Of the 5,910 unique products ordered, Pantrymate's food item expiration database covered 221 of them. These data points are summarized in Table 2.

Evaluation Data Analysis	
User Sample Count	100 users
Total Orders Placed	1418 orders
Avg. Orders Placed/User	14 orders
Avg. Products Ordered/User	129 items
Products Reordered	7053 items
Unique Products Ordered	5910 items
Unique items covered by Pantramte's expiration database	221 items
Avg. Time Between Orders	12.5 days

Table 2: Data analysis results for 100 Instacart users' grocery shopping profiles.

These data points have been used to provide a quantitative evaluation of the metrics proposed above to determine the significance of the targeted waste factors and the effectiveness of Pantrymate's user alerts. The evaluated metrics can be seen below in Table 3.

Pantrymate Evaluation Metrics Results	
Percentage total products reordered	54.40
Avg. overbuying alerts/user	72 alerts
Percentage unique products covered by expiration database	3.70
Avg. expiration alerts/user with current expiration coverage	5 alerts
Avg. expiration alerts/user with 25 percent product expiration coverage	32 alerts
Avg. expiration alerts/user with 50 percent product expiration coverage	64 alerts
Avg. expiration alerts/user with 75 percen product expiration coverage	96 alerts
Avg. expiration alerts/user with 90 percent product expiration coverage	116 alerts

Table 3: Pantrymate Evaluation Metrics Results.

In summary, 54.4% of the food items purchased were reordered. Using this data to simulate Pantrymate's behavior shows that the application would have alerted the average user to 72 potential instances of over buying, thus indicating that Pantrymate's proactive approach has significant potential to alert users to a key factor in household food waste. Only 3.7% of the products ordered by users were covered by Pantrymate's food expiration database. With this limited expiration date coverage, this data indicates that the average user would have been alerted to five cases of expiring food in their inventory. However, if Pantrymate's expiration data API were to be updated to cover more food items, it is possible that the application could alert users to one hundred expiring food items over the course of a year.

Lastly, it is worth briefly noting that Pantrymate has been implemented with an industry-standard development stack, with choices made across the board to maximize every aspect of the application's performance. React is currently considered to be at the top of user interface performance, and is used by companies such as Facebook (which developed it), AirBnb, Instagram, and Skype, among many others. Node.js is also pushing the standards of application performance, providing one of the most highly supported non-blocking I/O models for scalable system development available today. Even Pantrymate's authentication is industry-standard, using OAuth provided by two of the largest existing web services. As such, while it is impossible to directly compare Pantrymate's implementation stack alongside those of other related applications, a qualitative evaluation metric can be made qualifying Pantrymate as comparable, if not superior, in application performance to the remainder of existing applications addressing the problem of household food waste.

Chapter 6

Conclusion

Upon completion of this report, an open-source repository containing all of Pantrymate's code is available publicly on GitHub. The link to this repository can be found in Appendix A. This repository includes all of the code implemented for the application's client, server, and database, as discussed in Section 4. Additionally, both an Android release APK and an iOS release bundle have been exported and tested on hardware devices. This means that Pantrymate is theoretically ready to be submitted for official testing and deployment on the App Store and Google Play. The current application implementation accomplishes the primary goal of this project: notifying users of potential instances of household food waste while simultaneously raising nutrient awareness. It does so by combining the functionality of a shopping coordination application and an inventory management application, enabling an especially proactive and automated approach. This approach not only allows Pantrymate to uniquely target instances of potential food waste before they occur, but it also substantially reduces the time burden placed on users as they use the application. Furthermore, an unintended result of this project has been the implementation of the foundations for the first open-source food item expiration database and API. That said, Pantrymate's current implementation has a number of limitations, and notable future work should be done before releasing an ideal application for public use.

6.1 Limitations

While Pantrymate's implementation took into careful consideration optimization of application performance, this is not to say that it is without current limitations. Perhaps the most significant of these is the limited data provided by Pantrymate's expiration API. Relative to the hundreds of thousands of food items found in Nutritionix's nutrition data API, the few hundred included in Pantrymate's expiration database provide highly limited coverage. Furthermore, the quality of this data, while having passed some manual verification, is not entirely reliable. Of course, depending on factors such as temperature and humidity of a given user's inventory storage infrastructure, the expiration data provided by the API might not be entirely accurate to begin with. This is an unavoidable limitation, but could perhaps be mitigated by including specifications for the assumed storage conditions.

Another limitation of the current implementation is that Pantrymate relies on relatively error-free user input data. Spell-check and auto-correct functionality has been included for all input forms, but it is still highly possible for user errors to occur when inputting food items in their shopping lists and inventory. Because both Nutritionix and the Pantrymate expiration API rely on food item labels for database look-up, user errors when entering food items can result in limited functionality for the nutrition information and expiration management features of the application. User errors are particularly difficult to work around, but perhaps the best method for fixing this limitation would be to have user's select the items they want to add from an available food item list that dynamically populates with potential food item options as the user types. This would ensure that only food items that are recognized by the application databases are added by the user. However, implementing such a feature would restrict the user to a highly limited number of food items that can be managed within the application, at least until further food item coverage is developed for the expiration database.

A final limitation is the rate limits imposed by the current implementation's Scandit and Nutritionix API plans. This can easily be fixed by upgrading to more expensive plans with full API coverage.

6.2 Future Work

One of the most exciting aspects of having worked on Pantrymate is the incredible potential for future work on this project. The application's current implementation is essentially an MVP, providing the core functionality necessary to demonstrate an effective implementation of the key aspects of Pantrymate's unique approach to reducing household food waste. This leaves a great deal of room for improvement of existing functionality and addition of new features.

As previously mentioned, one of the most significant opportunities for future work is in improving Pantrymate's expiration data API. This includes improving the quality of the existing data, as well as adding data for the many overlooked food items. Additionally, data could be added such that all of the food items in the database have expiration data for their various states, be they raw, cooked, packaged, opened, etc. While a huge undertaking, improving upon this database could provide not only a better application, but a helpful and otherwise unavailable resource for the greater open-source community. If the coverage for food item expiration data were greatly increased, this could also avoid user input-error as described above in Section 6.1.

Other potential improvements to existing Pantrmate features could include further developing the inventory management interface to better categorize items in a user's inventory. Instead of simply having 'Expiring', 'Leftovers', and 'All' as the menu tabs, additional categorizations such as 'Meat', 'Vegetables', 'Dairy', 'Grains', etc. could be implemented. Food items could then be automatically categorized using an improved food item database, or perhaps by implementing an NLP algorithm to determine food type classifications. Furthermore, a feature for dictation of inventory updates would help further improve Pantrymate's attempt at automation. For example, using the phone microphone and Google's Speech API, it might be possible to implement a feature that would allow a user to dictate directly what items they purchased, ate, or have as leftovers. This could similarly be applied to further automate shopping list creation.

Another possibility for future work would be to implement a recipe recommendation feature based on the user's food inventory that would help encourage the use or re-use of foods that might soon be expiring. This feature was part of the original application design, and an API key was actually obtained for Yummly [47], one of the leading recipe recommendation APIs. However this feature was ultimately cut from Pantrymate's initial implementation as it was deemed extraneous to the core application functionality.

These are just a few of the potential directions one could take when considering future work on this project. It is exciting to consider what Pantrymate might evolve into and the contribution it will have in reducing household food waste.

Appendix A

Code

Pantrymate's code is available at the following open source repository:

https://github.com/jkvlevin/pantrymate

Appendix B

Sample Evaluation Data

Order ID	Order Number	Days Since Prior Order	Product ID	Reordered
2539329	1		12427	0
2539329	1		26088	0
2539329	1		196	0
2539329	1		26405	0
2539329	1		14084	0
2398795	2	15	10258	0
2398795	2	15	196	1
2398795	2	15	13032	0
2398795	2	15	12427	1
2398795	2	15	26088	1
2398795	2	15	13176	0
473747	3	21	10258	1
473747	3	21	12427	1
473747	3	21	25133	0
473747	3	21	196	1
473747	3	21	30450	0

Order ID	Order Number	Days Since Prior Order	Product ID	Reordered
2254736	4	29	26405	1
2254736	4	29	196	1
2254736	4	29	25133	1
2254736	4	29	12427	1
2254736	4	29	10258	1
431534	5	28	41787	0
431534	5	28	10326	0
431534	5	28	12427	1
431534	5	28	25133	1
431534	5	28	196	1
431534	5	28	10258	1
431534	5	28	17122	0
431534	5	28	13176	1
3367565	6	19	10258	1
3367565	6	19	196	1
3367565	6	19	12427	1
3367565	6	19	25133	1
550135	7	20	25133	1
550135	7	20	12427	1
550135	7	20	13032	1
550135	7	20	196	1
550135	7	20	10258	1
3108588	8	14	196	1
3108588	8	14	49235	0
3108588	8	14	10258	1
3108588	8	14	25133	1
3108588	8	14	46149	0
3108588	8	14	12427	1

Order ID	Order Number	Days Since Prior Order	Product ID	Reordered
2295261	9	0	46149	1
2295261	9	0	25133	1
2295261	9	0	10258	1
2295261	9	0	196	1
2295261	9	0	12427	1
2295261	9	0	49235	1
2550362	10	30	46149	1
2550362	10	30	38928	0
2550362	10	30	10258	1
2550362	10	30	196	1
2550362	10	30	35951	0
2550362	10	30	13032	1
2550362	10	30	12427	1
2550362	10	30	25133	1
2550362	10	30	39657	0

Table 1: Sample dataset for single user grocery shopping profile as used for Pantrymate's evaluation.

Bibliography

[1] USDA, “Frequently asked questions.” [Online]. Available: https://www.usda.gov/oce/foodwaste/faqs.htm

[2] J. Fassler, “We’ve all heard the staggering statistics about food waste, a new study says they’re wrong,” 2017. [Online]. Available: https://newfoodeconomy.com/weve-heard-staggering-statistics-food-waste-new-study-says-theyre-wrong/

[3] USDA, “Food security overview.” [Online]. Available: https://www.ers.usda.gov/topics/food-nutrition-assistance/food-security-in-the-us/

[4] ——, “U.s. food waste challenge, usda’s activities.” [Online]. Available: https://www.usda.gov/oce/foodwaste/usda_commitments.html

[5] M. Kummu *et al.*, “Lost food, wasted resources: Global food supply chain losses and their impacts on freshwater, cropland, and fertiliser use,” Sep 2012.

[6] D. Cordell *et al.*, “The story of phosphorus: Global food security and food for thought,” Feb 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095937800800099X

[7] C. Hanson *et al.*, “What’s food loss and waste got to do with climate change? a lot, actually.” [Online]. Available: http://www.wri.org/blog/2015/12/whats-food-loss-and-waste-got-do-climate-change-lot-actually

[8] M. C. Heller and G. A. Keoleian, “Greenhouse gas emission estimates of u.s. dietary choices and food loss,” Sep 2014. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/jiec.12174/full

[9] ODPHP, “Shifts needed to align with healthy eating patterns,” 2015. [Online]. Available: https://health.gov/dietaryguidelines/2015/guidelines/chapter-2/current-eating-patterns-in-the-united-states/

[10] FAO, “Save food.” [Online]. Available: http://www.fao.org/save-food/news-and-multimedia/news/news-details/en/c/1026569/

[11] J. Parfitt, M. Barthel, and S. Macnaughton, “Food waste within food supply chains: quantification and potential for change to 2050,” Sep 2010. [Online]. Available: http://rstb.royalsocietypublishing.org/content/365/1554/3065.short

[12] J. Aschemann-Witzel, I. d. Hooge, P. Amani, T. Bech-Larsen, and M. Oostindjer, “Consumer-related food waste: Causes and potential for action,” May 2015. [Online]. Available: http://www.mdpi.com/2071-1050/7/6/6457/htm

[13] E. Ganglbauer, G. Fitzpatrick, and R. Comber, *Negotiating Food Waste: Using a Practice Lens to Inform Design.* [Online]. Available: https://pdfs.semanticscholar.org/0172/a63c981323e341514b385b99dfd89029c2bb.pdf

[14] C. Inc., "Fridge pal."

[15] C. C. Ltd, "Freshbox."

[16] EatChaFood, "Challenging technology design to slice food waste production." [Online]. Available: https://dl.acm.org/citation.cfm?id=2497311

[17] J. Ahn, J. Williamson, M. Gartrell, R. Han, Q. Lv, and S. Mishra, "Supporting healthy grocery shopping via mobile augmented reality." [Online]. Available: https://dl.acm.org/citation.cfm?id=2808207&CFID=991273917&CFTOKEN=64777883

[18] M. George, D. Mircic, G. Soros, C. Floerkemeier, and F. Mattern, *Fine-Grained Product Class Recognition for Assisted Shopping*, Oct 2015. [Online]. Available: https://arxiv.org/pdf/1510.04074.pdf

[19] E. Burton *et al.*, "A food sharing concept." [Online]. Available: https://dl.acm.org/citation.cfm?id=3154859

[20] K. Sridhar, "Waste no food." [Online]. Available: http://wastenofood.org/

[21] H. Raut, S. Rajput, D. Nalawade, and K. Kale, "Smartphone based waste food supply chain for aurangabad city using gis location based and google web services," *International Journal of Research in Engineering and Technology*, vol. 05, no. 04, p. 301308, 2016. [Online]. Available: http://esatjournals.net/ijret/2016v05/i04/IJRET20160504058.pdf

[22] R. Comber and A. Thieme, "Designing beyond habit: opening space for improved recycling and food waste behaviors through processes of persuasion, social influence and aversive affect." [Online]. Available: https://dl.acm.org/citation.cfm?id=2559091

[23] F. Altarriba *et al.*, "Reducing food waste through playful social interactions." [Online]. Available: https://dl.acm.org/citation.cfm?id=3079125

[24] A. Thieme, R. Comber, J. Miebach, J. Weeden, N. Kraemer, S. Lawson, and P. Olivier, "We've bin watching you: designing for reflection and social persuasion to promote sustainable lifestyles," May 2012. [Online]. Available: https://dl.acm.org/citation.cfm?id=2208394

[25] C. F. Hsu *et al.*, "Smart pantries for homes." [Online]. Available: http://ieeexplore.ieee.org/abstract/document/4274571/

[26] Facebook, "React native a framework for building native apps using react." [Online]. Available: https://facebook.github.io/react-native/

[27] Infinitered, "infinitered/ignite." [Online]. Available: https://github.com/infinitered/ignite/

[28] Reactjs, "Redux." [Online]. Available: https://redux.js.org/

[29] Oblador, "oblador/react-native-vector-icons." [Online]. Available: https://github.com/oblador/react-native-vector-icons

[30] J. Levin, *Bookbag: A Web Application for Collaborative Authoring and Distribution of Course Textbooks.*

[31] N. Foundation, “Express - node.js web application framework.” [Online]. Available: https://expressjs.com/

[32] ——, “Node.js.” [Online]. Available: https://nodejs.org/en/

[33] Google, “Firebase — app success made simple.” [Online]. Available: https://firebase.google.com/

[34] Heroku, “Heroku.” [Online]. Available: https://www.heroku.com/

[35] Passport, “Passport.js.” [Online]. Available: http://www.passportjs.org/

[36] OAuth, “Oauth community site.” [Online]. Available: https://oauth.net/

[37] Kelektiv, “kelektiv/node-uuid.” [Online]. Available: https://github.com/kelektiv/node-uuid

[38] “The levenshtein-algorithm.” [Online]. Available: http://www.levenshtein.net/

[39] Nutritionix, “Nutritionix.” [Online]. Available: https://www.nutritionix.com

[40] Xgfe, “xgfe/react-native-datepicker.” [Online]. Available: https://github.com/xgfe/react-native-datepicker

[41] Scandit, “Scandit.” [Online]. Available: https://docs.scandit.com/stable/react_native/index.html

[42] EatByDate, “How long does food last? guide to shelf life expiration.” [Online]. Available: http://www.eatbydate.com/

[43] StillTasty, “Keep it or toss it?” [Online]. Available: http://www.stilltasty.com/

[44] “Fall reminders.” [Online]. Available: https://formerlynmurbanhomesteader.weebly.com/uploads/2/2/5/0/22509786/fall_reminders__2015_.pdf

[45] “Convert csv to json.” [Online]. Available: http://www.convertcsv.com/csv-to-json.htm

[46] Instacart, “Instacart.” [Online]. Available: https://www.instacart.com/datasets/grocery-shopping-2017

[47] Yummly, “Personalized recipe recommendations and search.” [Online]. Available: https://www.yummly.com/