Boston University
Electrical & Computing Engineering
EC464 Capstone Senior Design Project

User's Manual

Fridge Buddy
Submitted to

Professor Pisano
Boston University ECE Department
8 St. Mary's Street
Boston, MA 02215
apisano@bu.edu

by

Team 10
Fridge Buddy

Team Members

Trevor Chan trevchan@bu.edu
Jeffrey Chen jchen25@bu.edu
Andres Garcia andresfg@bu.edu
Cole Wentzel wentzel@bu.edu

Submitted: 14 Apr 2024

Table of Contents

Executive Summary

Fridge Buddy: Your Waste-Reducing Smart Refrigerator Companion

Food waste is a significant issue in the United States, with a substantial portion occurring as a result of grocery spoiling in households. Fridge Buddy aims to combat this problem by empowering users to manage food items efficiently and reduce food waste by transforming traditional refrigerators into smart appliances. Fridge Buddy is a mobile application and an easily mountable touchscreen that tracks item inventory, synchronizes data across multiple devices, notifies users when expiration dates are near, and suggests ways to help users best make use of their grocery items. Users will be able to easily view, add, remove, and update items using a touchscreen interface that easily mounts onto existing refrigerators, supported with an optional mobile application. The data displayed and modified between these two interfaces will be synchronized through a cloud backend, ensuring that user data is always up-to-date. Users will also be provided access to a recipe suggestion feature to help them more optimally use their existing food, prioritizing items that will soon expire. Fridge Buddy empowers users to better manage food at home, reduce waste, and save money on groceries using its low-cost, effective solution.

1 Introduction

Fridge Buddy is a system designed to offset food waste at a household level and help consumers buy and consume food more efficiently. A study by Bosch home appliances found that consumers discard an average of $53.81 worth of spoiled food a week because they forget what is in their refrigerator, buy excess food, or allow food to spoil. Currently, consumers rely on printed expiration dates for expiring food management, but this information is insufficient as demonstrated by the large amount of food waste produced in America. Fridge Buddy will be an affordable and convenient alternative to existing solutions that give users the capabilities of a smart refrigerator without the financial and appliance replacement burdens of existing smart refrigerators.

Currently, existing solutions such as smart fridges couple expensive hardware solutions with software, but the cost can be $5,000 or more. Our product is a primary software solution that relies on much less expensive hardware, costing less than $130 per tablet to manufacture. Additionally, as young adults, our team knows that renters may not be allowed to replace the fridge in their unit, so traditional smart fridges may be inaccessible to those who do not own homes.

Using Fridge Buddy, a user can store up to 100 items and access them from any device. Using our user authentication, users can check their inventory from their phone, making it simple to see what may be needed when out at the store, instead of overbuying. When the user returns from the store, instead of spending time inputting each result individually, they're able to scan the barcode or hold the item up to the camera to have it automatically recognized and logged into their inventory. FridgeBuddy predicts the day that scanned items will expire for users who don't have the time to add the information manually and allows for manual entry for users who want the most precision.

Additionally, users will be notified when their items are within a week of their expiration date. This allows consumers to prioritize using these items. Fridge Buddy assists with this by suggesting meals and recipes the user can make using their existing food items. Fridge Buddy is able to take a user's entire inventory or prioritize items that will expire soon, and suggest up to 5 recipes to assist users in more efficiently using their food items.

We also ensure the security of user data by using secure cloud services for user authentication, information handling, and data storage. Despite using a camera module, Fridge Buddy seeks to ensure user privacy by not storing images any longer than it takes to process results.

Fridge Buddy emerges as a promising solution to combat food waste and promote efficient food management at the household level. By offering an affordable alternative to costly smart refrigerators and leveraging user-friendly software with minimal hardware requirements, Fridge Buddy aims to improve access to efficient food inventory management.

Throughout the rest of this manual, users will be directed through Fridge Buddy's system design, typical user operations, technical background, engineering standards, and price breakdown.

2 System Overview and Installation

2.1 Overview Block Diagram

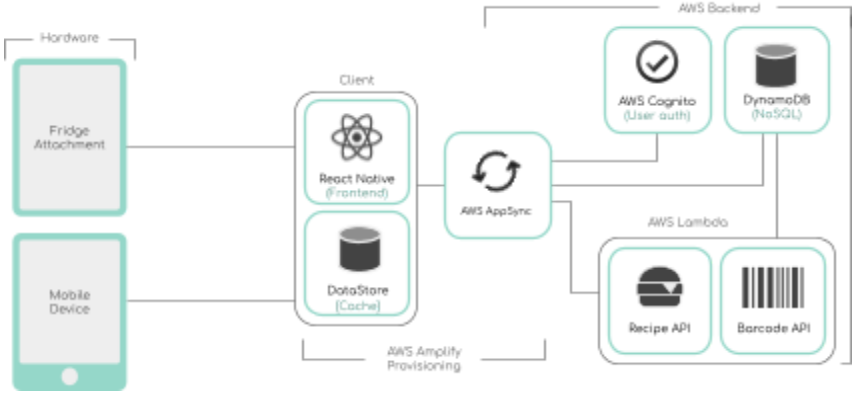

Figure 1: Fridge Buddy System Block Diagram

The overall system design of the Fridge Buddy consists of its hardware setup (mobile application and mountable touchscreen), its client (React Native application with AWS AppSync provisioning), and backend (AWS services and external APIs). These components will be further discussed in section 4 of this document.

2.2 User Interface

The user interface Fridge Buddy allows for grocery management by allowing users to add, view, update, and remove items. Additionally, it allows users to generate recipes from current items and monitor expiration dates. The interface operates on both the included touchscreen unit as well as on the Fridge Buddy mobile app.

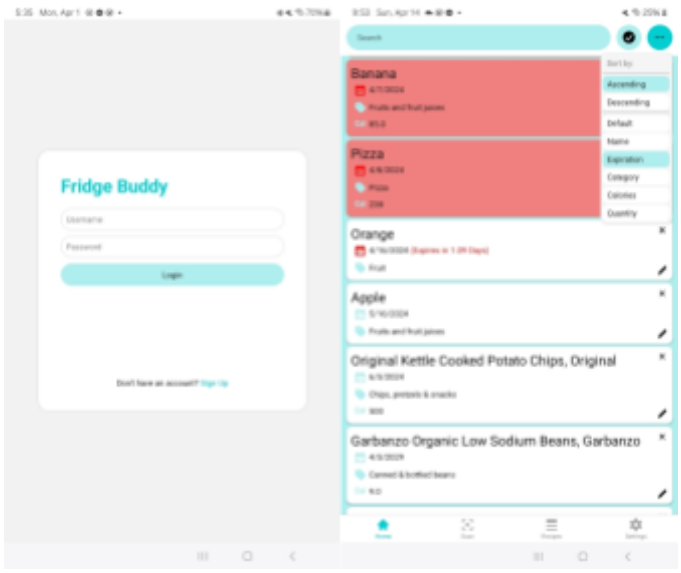

Figure 2: Fridge Buddy Login Page and Home Page

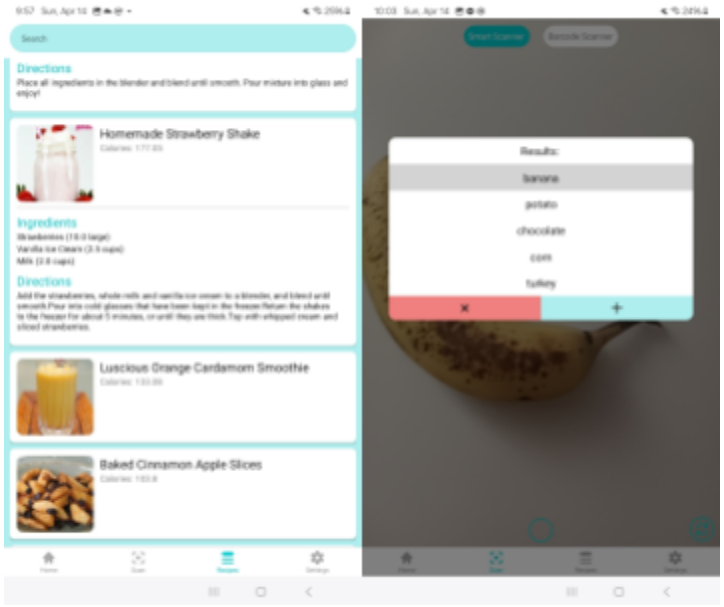

Figure 3: Fridge Buddy Recipe Page and Item Scanning Page

2.3 Physical Description

The Fridge Buddy includes a touchscreen unit to be mounted conveniently onto refrigerator doors. Hardware accessories included are:

- Housing clips x2
- Magnetic inserts x4
- Touchscreen panel
- Power cable

2.4 Installation and Setup

Hardware setup:

1. For each housing clip, insert 2 magnet assemblies
2. Attach the housing clips onto the top and bottom of the touchscreen unit, oriented such that the camera module and power port are exposed
3. Attach the completed assembly onto a refrigerator door
4. Connect the power cable to the power cord
5. Allow the device to power on and follow the on-screen instructions to set up or log in to a Fridge Buddy account

Mobile application:

1. Install the Fridge Buddy SDK on your mobile device
2. Allow the device to power on and follow the on-screen instructions to set up or log in to a Fridge Buddy account

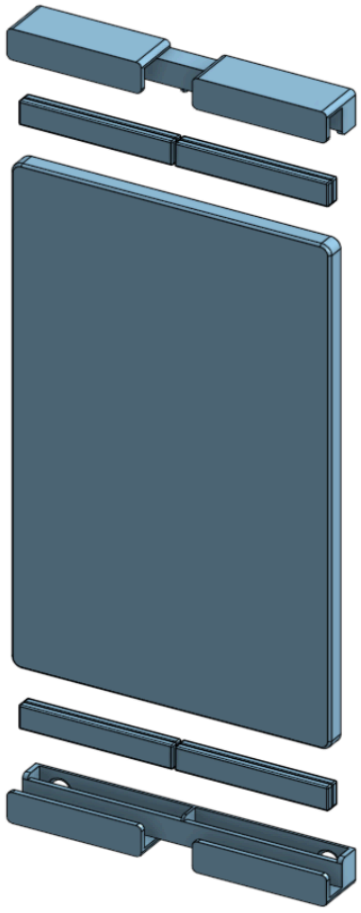

Figure 4: Fridge Buddy Hardware Setup

3 Operation of the Project

3.1 Operating Mode 1: Normal Operation

Managing Inventory

The user can view their current item inventory on the **Home** page of the Fridge Buddy tablet and mobile app. A user's items will sync across devices as long as they are connected to the Internet and logged in with their Fridge Buddy account.

The user can manually add an item to their inventory on the **Home** page by entering the following relevant information in the *Add Item* field:

1. Item name (required)
2. Expiration date (optional)
3. Category (optional)
4. Quantity (optional)
5. Calories (optional)

After entering the item information, the user can press the + button to save the item to their inventory.

To modify or remove an item from their inventory, the user can press the *Edit* button to enter the editing mode on the **Home** page. The user can then modify the item information of items in their inventory, or delete items by pressing the × button next to the appropriate item.

The user can search for items in their inventory by pressing the *Search* button in the **Home** page and typing in the text field that pops up. Items that match the user's text entry will be returned based on item names and categories. The user can also sort their existing items by pressing the *"..."* button in the top right based on filters such as calorie count, name, expiration date, etc.

Scanning Items

The user can scan items to add them to their item inventory using the **Scan** page of the Fridge Buddy tablet and mobile app. The **Scan** page offers two modes of scanning: **Barcode Scan** and **Smart Scan**.

The user can add an item to their item inventory by scanning the item barcode by going to the **Scan** page and selecting **Barcode Scan**. This is most useful for packaged food items. The following steps are taken:

1. On the Fridge Buddy tablet, the tablet's front camera view is displayed. On the Fridge Buddy mobile app, the device's back camera view is displayed (with a button that allows the user to switch which camera view to use).
    a. If using the Fridge Buddy tablet, hold the item up so that the barcode is visible within the camera view.
    b. If using the mobile app, point the camera at the item so that the barcode is visible within the camera view.
2. The barcode will be scanned automatically when recognized.
3. The user is alerted with a prompt confirming that the barcode has been scanned.
4. The item is automatically added to the user's inventory.

The user can also add an item to their item inventory using AI-based food item recognition by going to the **Scan** page and selecting **Smart Scan**. This is most useful for produce, prepared foods, and other unpackaged food items. The following steps are taken:

1. On the Fridge Buddy tablet, the tablet's front camera view is displayed. On the Fridge Buddy mobile app, the device's back camera view is displayed.
    a. If using the Fridge Buddy tablet, hold the item up so that it is clearly visible within the camera view.
    b. If using the mobile app, point the camera at the item so that it is clearly visible within the camera view.
2. Press the *Capture* button when the item is clearly visible within the camera view.
3. The user will be prompted to select from five predicted food items that were identified. One or more food items can be selected.

4. Press the *Add* button to add the selected items to the user's inventory.

Fetching Recipes

The user can search for recipes that utilize items in their item inventory by going to the **Home** page of the Fridge Buddy tablet or mobile app and taking the following steps:

1. Press the *Select* button.
2. The user can select the items that they would like to include in the recipe.
3. Press the *Generate Recipes* button to search for recipes based on the user's selections.
4. A list of recipes is displayed on the **Recipes** page.
5. The user can select a recipe to view its full list of recipe names, ingredients, step-by-step instructions, and calorie count.

To save a recipe, press the *Favorite* button for the desired recipe (so that the recipe will continue to display even after the next search). Favorited recipes and those from the last recipes search can be viewed on the **Recipes** page.

Alternatively, the user can also search for recipes by using the *Search* bar on the **Recipes** page and entering the name of their desired recipe. Hitting *Enter* will search for recipes based on the inputted name and display them on the **Recipes** page.

3.2 Operating Mode 2: Abnormal Operation

Managing Inventory

If the user does not have a working Internet connection, the Fridge Buddy tablet and mobile app will encounter the following errors with syncing the item inventory:

1. Items may be missing from the user's item inventory on the **Home** page.
2. The user will be unable to add items to their inventory through manual entry on the **Home** page or by scanning the item on the **Scan** page.
3. The user will be unable to modify item information in the **Home** page.
4. The user will be unable to remove items from their item inventory in the **Home** page.

Scanning Items

1. When scanning using the **Barcode Scan** mode on the **Scan** page, it is possible that the barcode does not match any entry in the FoodCentral database. If this occurs, the user should manually add the item to the **Home** page.
2. When scanning using the **Smart Scan** mode on the **Scan** page, it is possible for none of the food item predictions to be correct. The following can be done if this occurs:
    a. Ensure that the item is unpackaged.
    b. Ensure that the entire item is within the camera view.
    c. Ensure that the image is as clear as possible. A more brightly lit image with no obstructions to the food item will yield better results.
    d. If using the mobile app, placing the item on a plain surface when capturing an image of it will yield better results.

e. Manually add the item within the **Home** page if predictions continue to be incorrect.

Item Information

An item's information such as expiration date, category, and calories can be incorrect when automatically retrieved from scanning an item. The user can correct any inaccurate item information by pressing the *Edit* button on the **Home** page.

4 Technical Background

4.1 Hardware Approach

Fridge Buddy's main hardware component is the mountable touchscreen interface designed to attach to existing refrigerator doors. A Samsung A7 Lite tablet was selected because of its size and relatively cheap cost. Using a pre-built tablet rather than a custom hardware solution also allowed the team to focus more on software development. The enclosure for the tablet was designed using CAD such that it easily slips onto the tablet with two strong magnets on both brackets so that it can be set up on a magnetic refrigerator surface. The touchscreen interface also comes with an adhesive option for users who have non-magnetic refrigerators.

To power the device, a flat charging cable was selected to remove the inconvenience of charging reusable batteries. The main goal of the product is to be readily available and convenient to use, and as such it is designed to require only its initial setup.

4.2 Frontend Approach

The frontend of the mobile application and touchscreen interface is developed using the React Native framework and TypeScript. React Native was chosen for Fridge Buddy because of its cross-platform capabilities for iOS and Android, code reusability between platforms, native performance resulting in a smooth user experience, and large developer community. Thanks to the cross-platform functionality of React Native, we are also able to use the same application for both the mobile application and touchscreen interface.

The Expo-Camera React Native library is used to incorporate the camera of the touchscreen interface and mobile devices for Fridge Buddy. This camera feature is used for adding items using barcode scanning and object recognition. The Expo-Camera library was also chosen for Fridge Buddy because it includes Universal Product Code (UPC) retrieval from barcodes recognized in the camera view.

4.3 Cloud Backend Approach

Fridge Buddy incorporates Amazon Web Services (AWS) as its cloud backend for data storage and synchronization, computation of 3rd-Party API calls (e.g. FoodCentral and Spoonacular), and user authentication. AWS was selected as the cloud service provider because of its widely-used services like DynamoDB and Lambda that fulfill Fridge Buddy's storage and

computational needs, as well as its service Amplify which can connect React Native applications to AWS services (which is important for connecting the frontend application with the cloud backend).

DynamoDB is AWS's distributed NoSQL database service, meaning that there is no enforced table schema for the stored data. The data is instead only accessed through a single key (which can be split into a partition and a sort key). Fridge Buddy utilizes DynamoDB for storing and accessing user information (username, password, email, and name) and item information (name, Universal Product Code, expiration date, quantity, calories, category, and image URL), with its NoSQL schema allowing for all information belonging to a given user being easily accessible and queryable using the user's ID.

Lambda is AWS's computing platform that allows users to execute specified code as containerized workloads. Lambda was selected because of its lightweight nature, only using computational resources when a function is called. Fridge Buddy is using the service for querying 3rd party APIs that are called by and connected to the React Native frontend application using AWS Amplify. The Lambda functions Fridge Buddy uses include querying the FoodCentral API for item information and inserting the item into DynamoDB for users, as well as querying the Spoonacular API with user items to get suggested recipes and recipe nutritional information.

Amplify is AWS's toolkit that enables users to connect frontend applications to backend AWS services. Fridge Buddy uses Amplify's React Native library and features to connect the previously mentioned AWS services to the React Native frontend application. Amplify was also selected for Fridge Buddy because of its "Auth" features that allow frontend applications to easily set up the AWS Cognito service for user authentication.

AppSync is the middleware that Amplify uses to connect frontend applications to backend services. Fridge Buddy utilizes AppSync's GraphQL API to define how the React Native application can interact with the application's DynamoDB database and Lambda function calls.

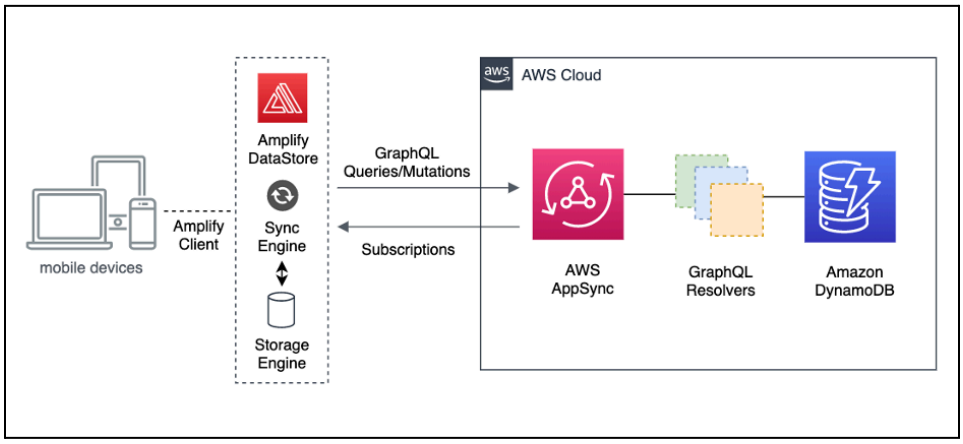

Figure 5: How Amplify and AppSync Connect to AWS Cloud

4.3 API Approach

Fridge Buddy incorporates multiple external APIs and libraries for functionalities such as querying for item information and object recognition. The external APIs used by the application are:

- **FoodCentral**: United States Department of Agriculture (USDA)'s official API and database for storing food product information such as Universal Product Code (UPC) and nutrition information. This API is called using an AWS Lambda function when users add items using UPC barcodes or object recognition. When users add an item in those ways, Fridge Buddy sends the barcode or item name retrieved from object recognition to FoodCentral to query item information including item name, category, and calorie count. This API was chosen because of its large size, breadth of information, and its credibility from being created by the USDA.
- **Spoonacular**: An online API with a large breadth of capabilities including getting a list of recipes using given product names and retrieving recipe nutritional information. Spoonacular was selected because of its generous API free tier for prototyping, as well as its capabilities for gathering recipe information. Although Spoonacular includes barcode lookup functions, its limited database made FoodCentral the better option.
- **Clarifai**: An online API that uses a pre-trained model to classify food items in an inputted image. This API can recognize multiple food items within a single image and returns a list of all possible food names and their corresponding accuracy percent. Fridge Buddy uses this API with its smart scanner to use object recognition for inputting user items. This API was also selected because its multi-item output allows users to input multiple items with a single image.

4.4 Expiration Date Prediction

Expiration date prediction occurs in the background when the user adds an item to their inventory through the Barcode Scanner or Smart Scanner. Since an expiration date is not an available piece of information that can be fetched from the FoodCentral API, the expiration date prediction function serves to automatically fill in this information for the user with a near-accurate value.

- **Data Extraction and Formatting:** Relevant data for expiration date prediction was extracted and formatted from the FoodKeeper dataset, which is publicly available on the USDA FSIS website. This was achieved through a Python script named formatData.py. The script extracted data from the "Product" sheet of the foodkeeper.json file and formatted it into a structured JSON file (foodkeeperFormatted.json), enhancing efficiency for subsequent calculations.
- **Predicted Time Until Expiration Calculation:** A script named expTimeCalc.py was developed to calculate predicted times until expiration for over 600 food product types. The script followed specified conditions and preferences for determining the basis for expiration time calculation, considering various storage types and available data.

- **Expiration Date Prediction Script:** The expPredict.py script was created to utilize the previously generated expiration predictions dataset (expirationPredictions.json) for predicting expiration dates. Upon input of a food item name, the script employs a best match, score-based search to identify the corresponding food product type and its associated expiration time. Using the Python datetime module, the current date is fetched, and the expiration date is calculated by adding the expiration time to the current date.
- **Integration into Fridge Buddy:** Modifications were made to the expPredict.py script to output the predicted expiration date as a Unix timestamp. The script was converted into a function (predict_expiration) that takes an input string and returns the expiration date timestamp. A Lambda handler function was created to call predict_expiration with the input string from the event and return the expiration date timestamp, facilitating integration into the Fridge Buddy app. To optimize prediction accuracy, it was decided to use generic product names for expiration date prediction, derived from branded item names.

5 Relevant Engineering Standards

5.1 React Native, JavaScript, and JSX

The application for the Fridge Buddy user interface utilizes React Native, a framework based on JavaScript and JSX. JavaScript serves as the programming language and provides the logic and functionality for the application. React Native offers a layer of abstraction allowing developers to write JavaScript code that translates into native platform components (for iOS and Android). JSX is a syntax extension for JavaScript that allows for HTML-like structures within a React or React Native codebase. These structures represent user interface components of the application.

5.2 TypeScript

This project leverages TypeScript, a superset of JavaScript that adds optional static typing. Static typing allows for defining the data types of variables and functions, leading to several benefits over vanilla JavaScript:

1. **Enhanced Code Maintainability:**
   Types contribute to improved code comprehension and reasoning, thereby mitigating the likelihood of errors during development.
2. **Early Error Detection:** The TypeScript compiler possesses the capability to identify potential type mismatches prior to runtime, effectively catching errors earlier in the development lifecycle.
3. **Elevated Developer Experience:** Type information can provide autocompletion and code navigation functionalities within development environments, leading to increased developer productivity.

5.3 GraphQL

The application leverages GraphQL, a query language for APIs that allows the client to request specific data from a resource server. GraphQL provides a single endpoint for fetching data, enabling clients to specify the data required in a single query. This approach contrasts with traditional REST APIs, which often require multiple endpoint calls to retrieve necessary data. GraphQL queries are built using a schema-driven approach. The server defines a schema that outlines the data types, mutations, and queries that establish how clients can interact with data sources configured on the GraphQL API. The client then constructs queries specifying the desired data by referencing these types and fields. The GraphQL schema used for Fridge Buddy was designed based on AWS's documented modeling style, using authentication rules to ensure the cybersecurity principle of least privilege (maintaining the security of a system by granting users the minimum required amount of privileges) to give users limited access to information.

5.4 Time/Date Formats

Fridge Buddy stores date information (e.g. item expiration dates) in the Unix timestamp format. This time format represents the number of seconds that have transpired since the Unix epoch (January 1st, 1970, 00:00:00 UTC) and offers the following benefits:

1. **Universality:** Unix timestamps are acknowledged on a broad scale and are independent of the platform being used, facilitating data exchange and storage.
2. **Efficiency:** Storing timestamps as integers necessitates less storage space compared to human-readable date formats.
3. **Scalability:** Unix timestamps are readily manipulated and compared mathematically, simplifying calculations involving dates.

While Unix timestamps are employed internally, the user interface converts them into human-readable formats (e.g., "2024-03-24") for display purposes.

6 Cost Breakdown

Table 1: Project Cost Breakdown of Fridge Buddy

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	1	Touchscreen unit + enclosure	$110	
2	1	Power cable	$5	
3	1	Spoonacular API Calls		~$79 /mo (standard pricing) ~$10 /mo (student discount)
4	1	AWS Calls		~$50 /mo
5	1	Clarifai Calls		~ $30 /mo
			Beta Version-Total Cost	$115/unit + $159/month

Our cost breakdown is a bit tricky because our major costs are not concrete, but instead based on usage. While the Food Central database does not incur any cost based on volume, Spoonacular and Clarifai require monthly subscriptions and AWS charges based purely on usage. This means our costs are variable based on the amount of calls each user makes. Calls to Clarifai happen when the user attempts to recognize a food item using the camera. The Spoonacular API is called when a user searches for recipes. AWS is used when the user logs in, accesses their inventory, or updates their inventory.

Therefore, we have a per-unit cost to manufacture the hardware, but we also have variable costs based on user volume. For this project we are able to get an academic discount to use Spoonacular but assume this is not the case for a commercial product. We are estimating the base tiers for Spoonacular and Clarifai for the initial release and around ~$50/month in AWS costs.

7 Appendices

7.1 Appendix A - Specifications

Team 10
Team Name: Fridge Buddy
Name: Fridge Buddy

Table 2: Engineering Specifications of Fridge Buddy

Requirement	**Value, range, tolerance, units**
Attachment setup time	< 5 minutes
Touchscreen stability	Unit stays mounted with ≤ 15 lbs of force exerted on refrigerator door
Cross-platform interface	Compatible with iOS and Android systems
Size	< 13.8in (35cm) x 9.8cm(25cm) x 3.1cm (8cm)
Weight	< 5lb (2.27kg)
Data synchronization	< 1 minute
Food recognition	90% accuracy
Text recognition	90% accuracy
Expiration date prediction	80% accuracy based on item
Automatic inventory input	< 10s after scanning
Item storage	≥ 200 items
Item removal	≤ 1 button press per item
On-screen list updating	< 3s
Recipe generation	≥ 2 recipes, < 10s
Total component cost	≤ $250
Power cord cross-section	< 0.25in^2 (1.6cm^2)
Power cord stability	Cord stays plugged in with ≤ 15 lbs of force exerted on refrigerator door

7.2 Appendix B - Team Information

Trevor Chan worked on the AWS cloud infrastructure design and implementation. He is a graduating Computer Engineer who will continue his education in Boston University's ECE Master's Program. His long-term goals include pursuing a career in backend infrastructure software engineering.

Jeffrey Chen worked on the React Native frontend, which involved building the user interface and integrating the project's background services, as well as designing the device enclosure. He will be graduating with a degree in Computer Engineering and working as a software developer at ISO New England.

Andres Garcia worked on the barcode scanning, food item recognition, and expiration date prediction features of the Fridge Buddy application. He is graduating with a degree in Computer Engineering and currently working as a TA for EC441 Computer Networking, and as a cybersecurity analyst intern at Technical Consulting & Research, Inc. He wishes to pursue a career in cybersecurity after graduation.

Cole Wentzel worked on the recipe prediction and nutrition information retrieval features for the Fridge Buddy application. He will be graduating with a degree in Computer Engineering and wants to pursue a career in embedded systems or low-level software engineering.