RecipeCart

Senior Design II — Final Report

UNIVERSITY OF
CENTRAL FLORIDA

College of Engineering and Computer Science

Department of Electrical and Computer Engineering

Dr. Lei Wei & Dr. Chung Yong Chan

Department of Computer Science

Dr. Matthew Gerber & Mr. Richard Leinecker

EEL4914 – Group 35

Darren Ha — Computer Engineering

Quan Nguyen — Computer Engineering

Bryan Ha — Computer Science

Reviewers

Dr. Enxia Zhang — ECE

Dr. Tanvir Ahmed — CS

Mr. Arup Guha — CS

Table of Contents

1.0 Executive Summary

The goal of this project is to develop a unified architecture for the smart detection of ingredient items and generation of insights on how to combine the ingredient items to form user-aligned recipes. Generally, anti-food-waste recipe recommendation apps often lack a robust method of inputting ingredient items and their quantities, often requiring that users manually input all their ingredients. Furthermore, due to only selecting recipes rather than producing new ones, they often force users to make their own modifications to existing recipes to tailor to their own dietary needs or ingredient availability.

With the advancement of technology, these issues have been mostly alleviated. Many companies like Samsung and Amazon have developed smart fridges and smart shopping carts that prove to be convenient tools for inputting and managing said ingredient data. Researchers have developed artificial intelligence capable of generating novel recipes given text prompts. However, these methods often fall short: smart devices often come with extremely high cost of entry for users, often offsetting the marginally increased consumer convenience, and artificially generated recipes face the issue of alignment with users' preferences by failing to encourage the generated recipes' fitness in a human-driven market. The RecipeCart aims to bridge this gap by proposing a minimal and generalizable architecture that focuses on optimizing the cost, convenience, and experience for the users.

The RecipeCart encompasses a modern solution to consumerism habits, leveraging state-of-the-art cloud technologies, machine learning, and IoT concepts to provide users with alternatives to their food leftover dilemmas. Recipes are recommended to users based on the ingredients available to them, which directly reflects the user's physical inventory. Ingredients are dynamically identified through computer vision algorithms using a mini computer, and server-side logistics are handled through cloud-based services.

The RecipeCart is intended to be a complementary add-on service to the existing technology, accommodating to user dietary needs in a seamless manner by providing a pipeline from the raw, physical ingredients to digitally manageable recipes. The RecipeCart also attempts to address a shortcoming that none of these technologies seem to properly resolve: food object detection and classification.

In this report, we evaluate and identify the optimal components to build the physical and digital systems underlying the RecipeCart. We then detail the design and construction of these different components into one complete, end-to-end product. Finally, we cover the testing methodologies employed to validate both the individual components as well as their integration with the rest of the RecipeCart system.

2.0 Project Description

2.1 Project Context

Consumerism has always dominated and shaped the modern global economy. In response to increasing consumption and demands, customer-centric companies often find themselves relentlessly upscaling supplies to meet consumer demands. However, demands do not directly translate to needs, and several supply chains often end up producing surpluses. The common consumer's spending habits are a microcosm of this exact development. A consumer's speculation of their needs often leads to them buying more supplies than necessary, increasing the ratio of unused resources going to waste.

Correcting mass spending behaviors most effectively addresses this overconsumption problem but is not realistically achievable in the short term. Rather, the simpler approach is to accommodate the current spending habits. Our project, the RecipeCart, attempts to encourage the usage of leftover resources in the kitchen by generating a collection of personally tailored recipes given a physical assortment of ingredients.

2.2 Project Goals

The primary goal of the RecipeCart is to simply reduce the waste of leftover ingredients by creatively re-purposing them with novel recipe suggestions. The RecipeCart must function as an extension to existing IoT technologies, namely the smart fridge and the smart shopping cart.

To make the RecipeCart as user-friendly as possible, the RecipeCart must intelligently identify the ingredients being placed into its container and subsequently produce a list of user-relevant recipes. The ingredient classification is performed through a combination of a simple barcode detection system and a reverse, vector-based image search leveraging common modern machine learning frameworks. The recipes are extracted from an existing dataset and loaded into an internal database, where they can be custom-filtered and sorted based on user needs and displayed on a mobile platform for user interaction.

A more ambitious goal involves extending the recipe list to also include suggested grocery locations where the missing ingredients may be purchased. This development may entail bridging the RecipeCart with the Google Maps API and all relevant grocery store databases. Another advanced goal is to integrate a scaling system into the RecipeCart to quantify the ingredients and suggest recipes that more accurately reflect the state of the leftovers.

The foremost stretch goal is to integrate the RecipeCart with a smart fridge or smart shopping cart to demonstrate its realistic use cases as an add-on for existing grocery inventory management technologies. This task is deemed extremely advanced primarily due to the difficulty of obtaining development rights for a contemporary smart fridge or smart shopping cart as well as the necessary SDKs to allow for bi-directional communication between the RecipeCart and these proprietary technologies.

In designing a seamless input process for the end user, video streaming the food items is strongly considered. Using images as input implies having the user take pictures of the ingredients after they have been placed into the container. Video streaming the input allows for dynamic addition and removal of select food items from the container without the user having to directly modify their collection. However, running machine learning models in tandem with a video-based input poses an extreme challenge in both complexity and resource, making this feature a stretch goal.

The project is incomplete without mentioning AI-generated recipes. Although generating creative and composite recipes through AI has already been done multiple times, adding this feature is considered a stretch goal because the resulting recipes do not necessarily guarantee a similar level of delicacy. Not only do the AI-inspired recipes need to be vetted for adequacy, the recipe outputs must be formatted and cleaned before being added to the database. Incorporating this feature requires a significantly more advanced generative model and a process to validate the recipes.

2.3 Project Objectives

An ensemble of pre-trained machine learning models balancing between object classification and barcode parsing must be implemented to accurately identify the input ingredient. The object classifying model must classify visually distinct unpackaged groceries based on vector encoding similarities, while the second model must specifically detect for barcodes on packaged products and query an external UPC database for the relevant product metadata. Both models must be concurrently available to provide a flexible ingredient detection process for the users.

To capture pictures of the food item from a central point of view, the camera must be mounted at the front of the container—pointing inwards at an appropriate tilt—and be connected to a single board computer (SBC) for centralized power and control. The SBC itself must be equipped with the appropriate network adapters for easy communication with the backend server. The appropriate power sources must be attached to the hardware modules to simulate the RecipeCart as an add-on feature to existing technologies such as the Amazon DashCart and Samsung’s Smart Fridge.

A bathroom scale may be embedded into the cart container to retrieve the quantity of a detected ingredient. The scale may be wired to the rest of the cart body and must send its results to the SBC via analog voltage signals for data cleaning and processing.

A server must be developed to host the backend business logic to pre-process the input images, provide a repository for machine learning analytics, and query the external barcode database. The server must also host its own database to maintain the basic user metadata, recipes, ingredients, and other necessary information to tune the recipe recommendation system.

The mobile user interface must feature a customizable catalog of all previously queried recipes, fronted by a basic authentication system and allow users to dynamically update their recipe preferences and initiate requests to the RecipeCart to retrieve its current

contents. The app must evidently include the function to generate recipe recommendations based on the retrieved ingredients.

A recommendation system is added to provide more personalized recipe recommendations. The model will base its inferences on a variety of inputs such as recipe popularity, ratings, currently held ingredients, avoidances, and dietary preferences.

2.4 Related Works

The concept of the RecipeCart is not completely original and unique. There have been many prior inventions that attempt to facilitate the transition of food items from the grocery store to the home kitchen such as Amazon's Dash Cart and Samsung Food. This section highlights some related technologies that implement some form of food inventory management and tracking system, recipe database and personalization, and smart grocery shopping features.

2.4.1 Cust2Mate

Cust2Mate's Automated Smart Cart is a smart shopping cart with automatic item identification, weighing and checkout. These functions also attempt to address measures against theft or any error when taking items in and out of the cart (Lenovo, 2022). This feature greatly improves the users' shopping experience by omitting the waiting checkout time that regular buyers experience. Cust2Mate's carts are shelf-ready and operational, so they can be deployed quickly. The company currently administers pilots with several food chains around the world with the intention of significantly increasing their operations even further (Slater & Kreizman, 2022). The smart carts, as a launched product, are yielding significant returns on investments and gross profit.

Figure 2.1. Cust2Mate Smart Cart features

2.4.2 Samsung Smart Fridge and Family Hub

Samsung's advanced smart fridge integrates with other Samsung devices through the Family Hub and has AI assistance with Alexa. The fridge contains an inventory tracker, streaming services, and convenient, personalized apps such as memoing and shopping lists (Samsung, 2023). The Family Hub acts as more than a conventional smart fridge: it is a centralized medium for smart houses, integrating with other Samsung smart-home devices, as well as a computer itself with advanced user interface. The user interface on the most advanced Samsung smart fridge includes options to quickly view the contents stored inside the fridge, instantly access saved recipes and videos, make meal plans based on current food inventory, and create and share grocery shopping lists.

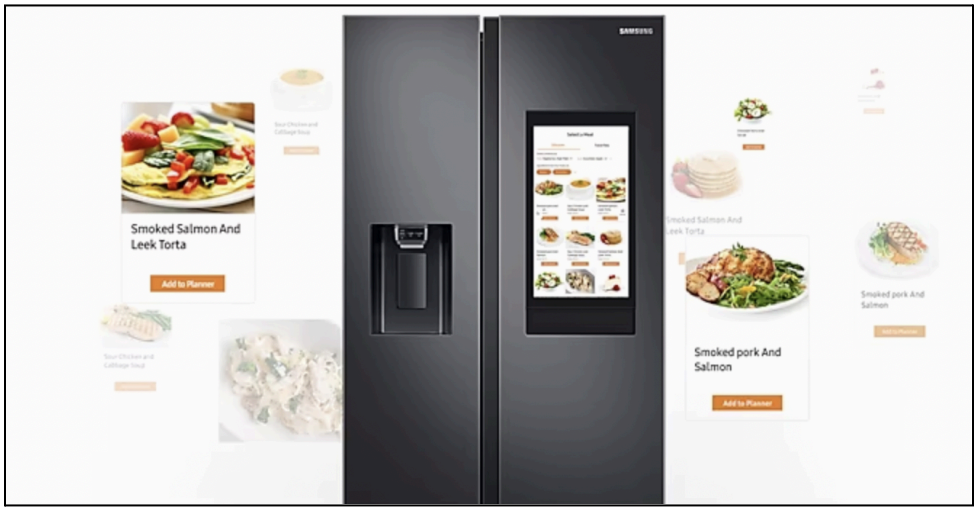


Figure 2.2. Samsung Family Hub integrated with the Smart Hub

2.4.3 Samsung Food

Samsung Food, originally branded as the Whisk app until August 2023, is a free recipe mobile app for recipe lookups, organization, and sharing. The app is integrated with many large and popular recipe websites such as the Food Network and AllRecipes.com [2.4]. The app initially acted as a meal planner, detailing users' daily view of their lunch, dinner, breakfast, and snacks, and subsequently outputting a health and wellness metric based on users' bioelectric impedance analysis (BIA). Certain advanced features—such as the ability to substitute ingredients in recipes and searching for recipes based on current ingredient inventory—were just recently added in 2023 to enhance users' experiences and engagement.

Users can search for recipes based on a variety of preferences and categories, ranging from international cuisines to specific dietary needs. Recipes can be further filtered based on preparation and cook time, and the ingredient quantities can be proportionally adjusted according to the desired number of servings [2.5]. Recipes are displayed to users on the home page based on their prior recipe preferences, ratings, and views. Samsung Food also automatically computes a health score for each recipe based on its nutritional

contents per serving. Users can add missing ingredients to a shopping list and export it to online grocery store platforms such as Instacart or Amazon Fresh.

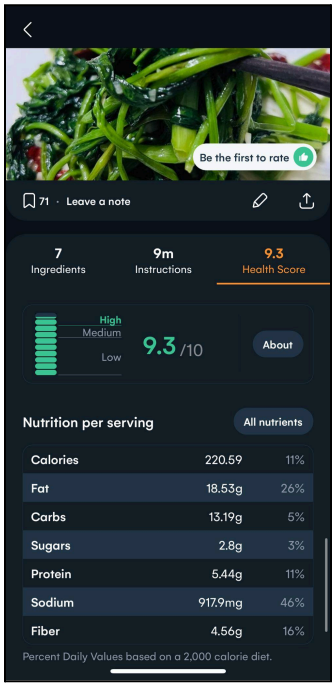


Figure 2.3. Example health score page for a recipe on Samsung Food

Samsung Food performs as a recipe-sharing and community-driven app, where users can follow popular food content creators, share comments, and assign approval ratings to recipes. Users can also create and join communities based on their food interests.

All three inventions exemplify successful launches of a product that eases a consumer's shopping and life experience by allowing them to monitor their purchases and their food inventory without the hassle of shuffling and inspecting through their cart or fridge. Cust2Mate designs a well-functioning smart cart with a simple user interface with a variety of virtual payment methods and omits the waiting time in the check-out lines. Samsung's Family Hub contains a built-in inventory tracker, as well as integration with different Samsung technologies such as Samsung Food, centralizing the fridge as one of the core smart appliances in a modern home.

Each technology provides a solution to a specific step along the meal preparation "pipeline," but none of them is fully encompassing or end-to-end. Our project aims to implement inventory tracking features as seen in Cust2Mate and Samsung's Family Hub, subsequently allowing the user to explore different recipe ideas based on their current ingredients as inspired from Samsung Food.

The RecipeCart is intended as a complementary resource to the existing technology, accommodating to user dietary needs in a seamless manner. It attempts to provide a

pipeline from the raw, physical ingredients to digitally manageable and customizable recipes. The RecipeCart also attempts to address a shortcoming that none of these technologies seem to properly resolve: food object detection and classification.

2.5 Project Specifications and Constraints

Specifications	**Description**	**Value**
Object Classification Time	Time to identify the object and return a list of ingredients	< 5 seconds
Object Classification Accuracy	A measure of the ML model's performance	> 80 %
Recipe Recommendation Time	Time to return a list of recipes given user preferences and a list of identified ingredients	< 5 seconds
Minimum Number of Recipe Outputs	Number of recipe recommendations guaranteed to generate	> 1 recipe
Absolute Weight Error	Tolerated error margin from the scale	< 20 grams
Broadcast Delay	Time to broadcast output from hardware to the server	< 2 seconds
Product Weight	Weight of the hardware	< 4 kg
Power Consumption	Power intake of hardware	< 15 Watts
Battery Life	Battery life before next recharge	~ 4 hour
Budget Management	Total estimate price of the whole production process	< $400

Table 2.1. Engineering design specifications

Among the specifications listed in the above table, the highlighted specifications indicate features that are demonstrated as part of the RecipeCart showcase. The object classification time and accuracy refer to the efficiency with identifying ingredients detected by the RecipeCart, both through barcode detection and image recognition. The recipe recommendation time refers to the time to process the inputted ingredients and generate the appropriate recipe recommendations.

2.6 General Hardware Block Diagram

The hardware architecture of the RecipeCart can be dissected into three main groups of components: the power supply system, the sensors, and the single board computer unit. The camera and weight sensors are powered through the SBC's power module, which gets its own power from an integrated power jack. The microcontroller is readily integrated with all the necessary modules to function as a single board computer. The sensors are wired to the microcontroller unit to forward their observations. The microcontroller pre-processes all observed data before forwarding the request to the backend server via a Network Interface Card (NIC) inside the Wi-Fi module. A Bluetooth audio module is included to provide users with feedback when scanning ingredients into the RecipeCart.

The colored boxes indicate the team member primarily in charge of researching and implementing the enveloped components. The work distribution is further discussed in Section 11.3.

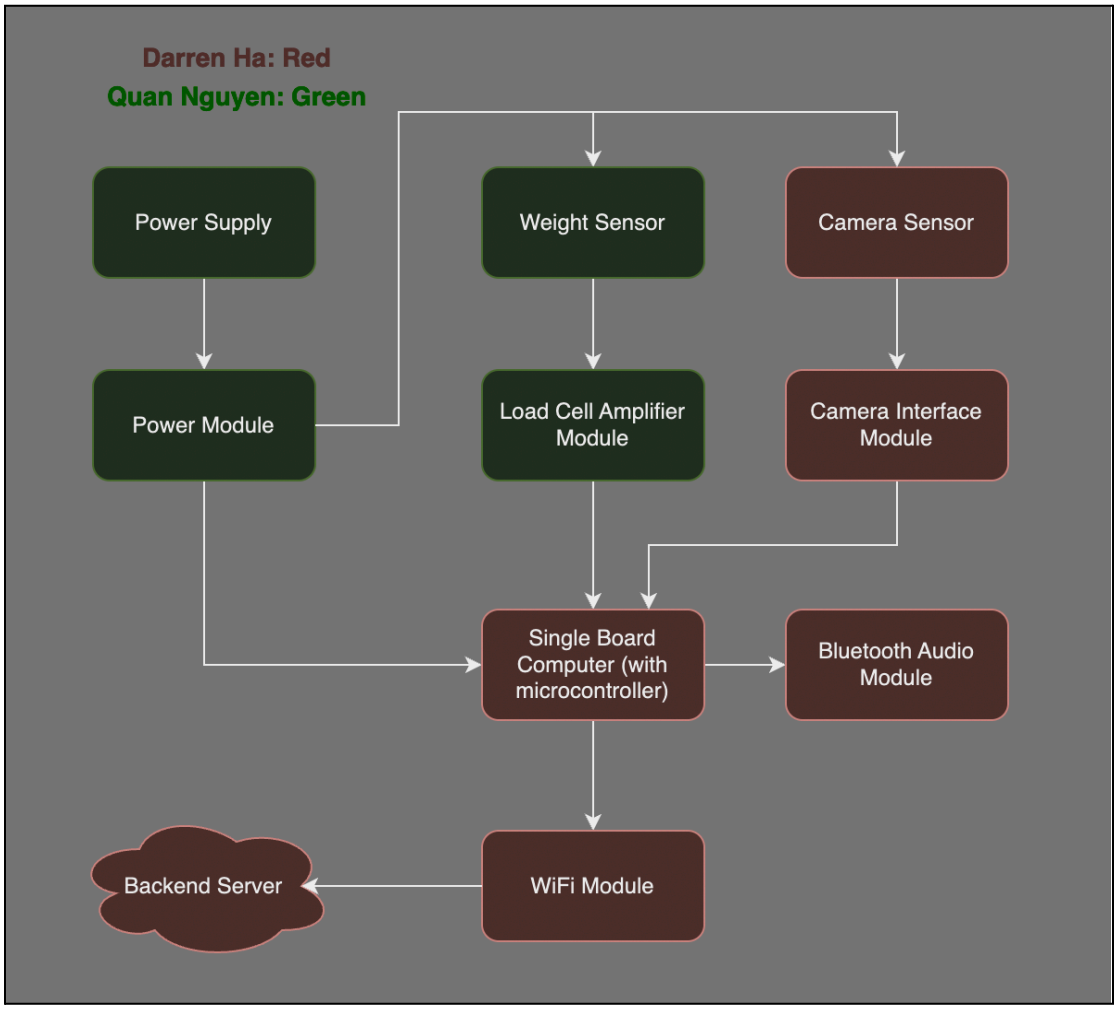


Figure 2.4. General hardware block diagram

2.7 General Software Diagrams

2.7.1 Ingredient Detection and Classification Flowchart

The below diagram depicts the general process of the ingredient detection and classification logic. Given the two methods of identifying an ingredient, the process is split into two primary pipelines: one for barcode querying and the other for an image-based food classification. The barcode-based classification is conditional on the detection of a barcode, while the food object classification method is triggered manually by the user through the mobile app interface.

Again, the colors indicate the team member in charge of researching, designing and implementing the particular components. Please refer to Section 11.3 for more information.

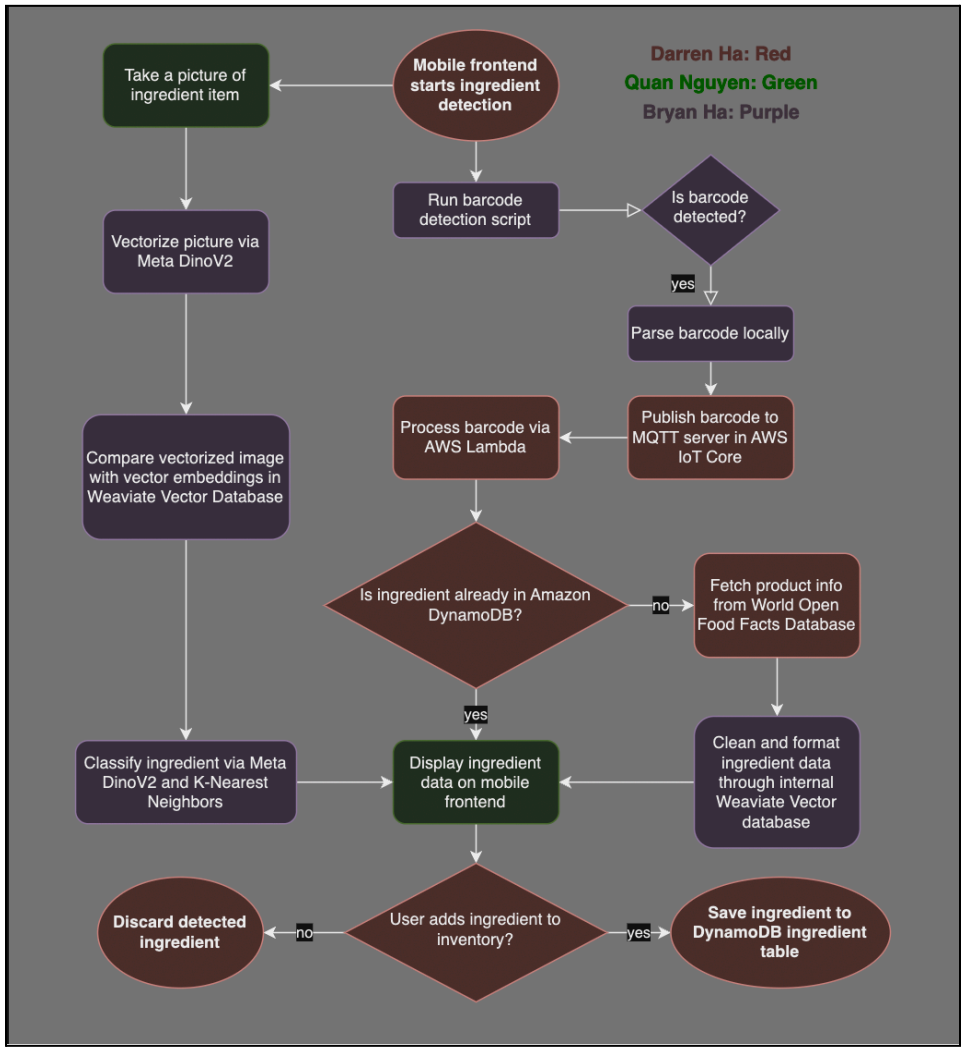


Figure 2.5. Ingredient detection and classification flowchart

2.7.2 Mobile App Flowchart

The RecipeCart is fronted by a mobile app to allow signed-in users to edit their diet, allergen, and preference profiles; view their ingredients; and explore and save recipes. The recipe search is backed by a simple recommendation system employing a combination of filtering and sorting algorithms centered around text-to-vector embedding similarities and based on users' ingredient inventory as well as their personal dietary preferences.

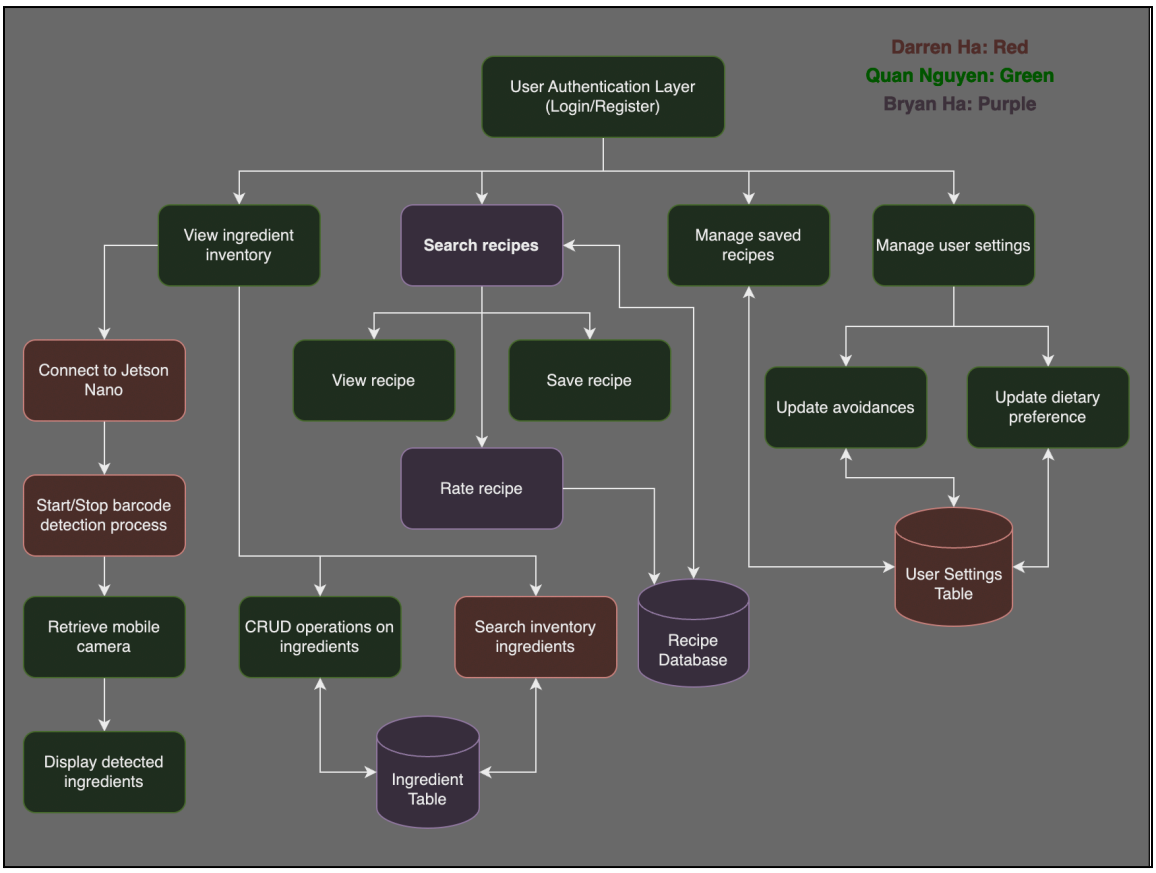


Figure 2.6. General mobile app flowchart

Users can view, rate, and save recipes as desired. Users may also manage their preferences by updating their list of avoidances and their preferred diet. Users may view their ingredient inventory, searching, updating, and removing ingredients.

Users may also initiate the ingredient detection process by connecting to the Jetson Nano or opening their camera app. Detected ingredients are streamed onto the mobile interface as users continue scanning more items into the RecipeCart.

The entire mobile app is secured behind a basic authentication system that requires all users to register an account if not log in. Users must verify their email upon signing up.

2.8 House of Quality

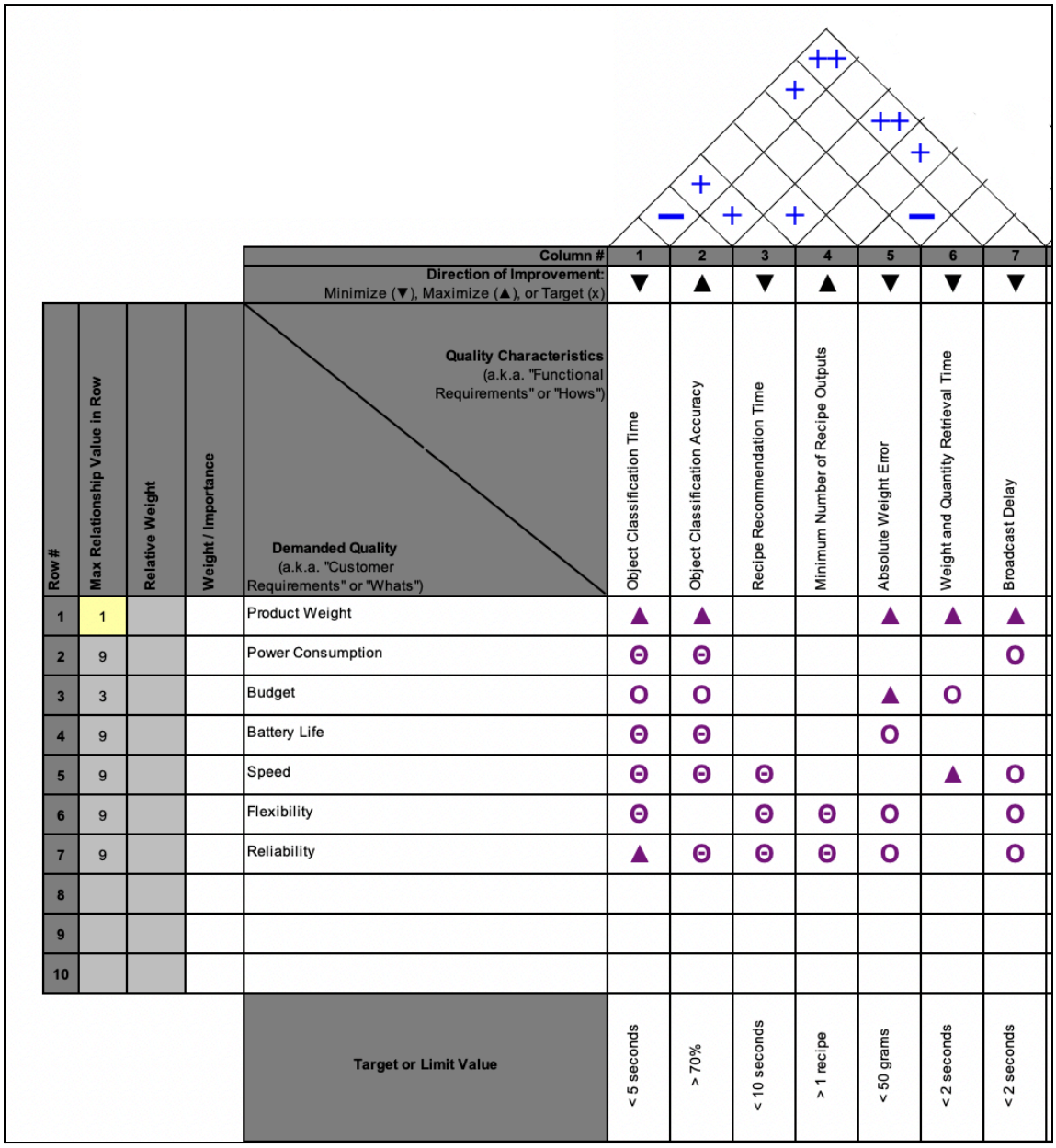


Figure 2.7. House of Quality diagram

The house of quality shows the main areas that should be focused on to maximize consumer satisfaction and its relation to engineering requirements. Analyzing the figure above, the project should focus on maximizing the operational efficiency of the ingredient recognition while minimizing power consumption and expenses. As emphasized in the project specifications section, recipe recommendation time and object classification time as well as the accuracy are the crux of the RecipeCart's showcase and directly influence the reliability and flexibility aspects of the project.

3.0 Hardware Parts Comparisons

Following the general hardware architecture described in Figures 2.1, this chapter examines several hardware technologies and their potential integration with the project's principal pipeline. While most parts highlighted are cross-compatible with one another, we discover that certain parts require extensive costs and efforts to integrate with others and may lead to unnecessary software complications. Parts are compared based on ease of integration, complexity, flexibility, cost, and also their adherence to the project specifications and design constraints.

3.1 Single Board Computers

The microcontroller is the heart of any embedded system and operates as the link between the hardware sensors and the software-side business logic. Typically, the microcontroller exerts central computational control over the various modules of an embedded system [3.1]. Its purpose is to perform efficient edge computing without the involvement of an actual server. In the context of the RecipeCart, we examine two approaches at using the microcontroller, both of which entail having it pre-made and integrated onto a single board computer (SBC). The first method involves the SBC as a simple edge device that captures, lightly pre-processes, and relays measurements to the server, where all the machine learning heavy-lifting is done. The latter method features the SBC as a more independent mini-computer capable of running some basic machine learning inferences—such as real-time barcode or text detection—before sending all of the data to the ML backend. The latter method calls for a well equipped and performant SBC while the former implies a less power consuming and simpler SBC.

3.1.1 Raspberry Pi

The Raspberry Pi Family provides powerful mini-computers that can run operating systems, browse and interact with the internet, and perform edge computing on IoT devices. The Raspberry Pi 4 is the most well-supported, cheapest option among the SBC options considered, while still maintaining the ability to perform basic machine learning inferences. It has for CPU the Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC, with a clock rate of 1.8 GHz [3.2]. The Raspberry Pi 4 is available at different memory sizes with varying prices. It comes with numerous USB and micro-HDMI ports and supports wireless LAN, Gigabit Ethernet, and Bluetooth connectivity. It also has a 2-lane MIPI CSI camera port.

Its main drawback is its weaker graphic processing power relative to the other SBCs, which may limit its machine learning applications depending on the deployed model and the inference type and frequency. The Raspberry Pi 4 can be paired with the Coral USB Accelerator to enhance its machine learning capabilities [3.3]. However, this addition would not only significantly ramp up the total cost of the SBC but also risk overheating

the central module. Running machine learning models for an extended time period on the Raspberry Pi 4 would require installing and configuring an additional heatsink and fan.

The Raspberry Pi 5 is set to be released at the end of October 2023 and is much higher performing than its older counterpart. It comes with a better CPU, the Broadcom BCM2712, 64-bit quad-core Cortex-A76 ARM with integrated L2 and L3 caches. The Raspberry Pi 5 also includes VideoCore VII GPU, allowing it to run machine learning workloads significantly more efficiently and faster than the Raspberry Pi 4 [3.4]. While the Raspberry Pi 5 has an integrated fan for its cooling purposes, running continuous and compute-heavy workloads would still require an additional active cooling system.

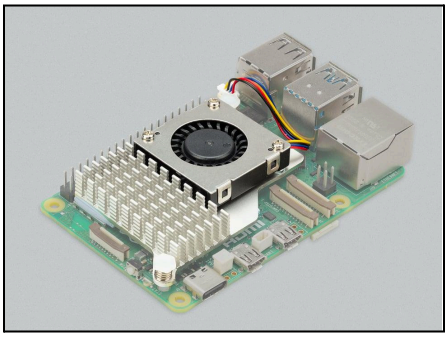


Figure 3.1. Raspberry Pi 5 with integrated fan

Given the novelty of the Raspberry Pi 5, it has relatively sparse documentation and community support, making it potentially more challenging to navigate and integrate than the Raspberry Pi 4.

Selecting the Raspberry Pi for the RecipeCart would be best suited for lighter machine learning workloads being performed on the edge. It would imply having most of the machine learning processing and inferencing occurring in the server-based backend.

3.1.2 Google Coral Dev Board

The Google Coral Dev Board is the most expensive and high performing option out of the bunch. Its CPU is the NXP's i.MX 8M SoC, with quad-core Cortex-A53 and Cortex-M4F—a lower power microcontroller designed to interact with external sensors. Not only does it have the Integrated GC7000 Lite Graphics as its GPU, it is also integrated with Google proprietary Edge TPU coprocessor and is capable of executing 4 trillion operations per seconds (TOPS), consuming 0.5 Watts per TOPS [3.5]. In addition to various USB ports and peripheral connections, it also supports 5 GHz WiFi via a 2x2 MIMO, Giga Ethernet, and Bluetooth 4.1. Video integration is supported through an HDMI 2.0a port and 4-lane MIPI CSI-2 camera FFC connector.

The Coral Dev Board's main flaw is its lack of compatibility with other deep learning libraries and frameworks [3.3]. Given its TPU module, it is specifically tailored to quickly run TensorFlow-based workloads, with limited flexibility and support provided for other deep learning software.

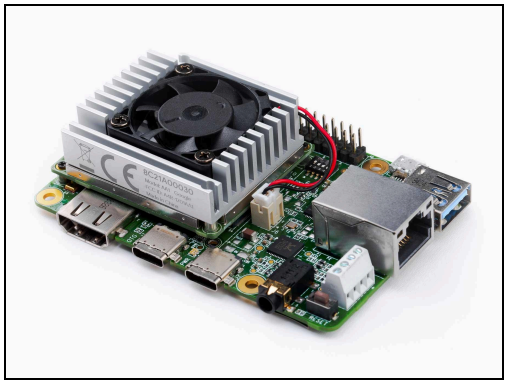


Figure 3.2. Coral Dev Board

The Coral Dev Board would be most appropriate for intensive on-the-edge machine learning applications. In addition to restricting the machine learning framework to only TensorFlow, incorporating it into the RecipeCart would imply having it process most, if not all, the workloads with little reliance on the backend server.

3.1.3 Nvidia Jetson Nano

Nvidia's Jetson Nano is the smallest form factor available as part of the Jetson Family of AI-dedicated SBCs for optimized edge computing. It is much more flexible than the Coral Dev Board, all the while being more powerful than the Raspberry Pi 4 and 5. The Jetson Nano supports not only TensorFlow but also PyTorch, Keras, and scikit-learn among others [3.3]. It also has many SDK libraries and extensive community support. It boasts a quad-core ARM A57 with a clock rate of 1.43 GHz as CPU and is integrated with Nvidia's 128-core Maxwell GPU [3.6]. This SBC comes with 12 lanes of MIPI CSI-2 for camera sensors, several USB connectors, and its own active cooling system.

While the Jetson Nano is less efficient than the Coral Dev Board when data processing and inferencing, its increased compatibility allows it to have a wider range of applications. Enabling WiFi and Bluetooth also requires an external dongle and additional configuration as the factory unit only comes with support for Gigabit Ethernet.

Figure 3.3. Jetson Nano

The Jetson Nano is by far the most flexible and balanced option of the trio as it provides adequate compute power to perform the necessary machine learning workloads on the edge while maintaining compatibility with a variety of deep neural network frameworks. We thus structure the RecipeCart's physical architecture around the Jetson Nano.

Name	**Raspberry Pi 4 / 5**	**Coral Dev Board**	**Jetson Nano**
CPU	Quad-core ARM Cortex-A72 / Cortex-A76	Quad-core ARM Cortex-A53 + Cortex-M4F	Quad-core ARM Cortex A57
GPU	None / VideoCore VII GPU	Integrated GC7000 Lite Graphics	128-core Nvidia Maxwell
TPU	None	Google Edge TPU coprocessor	None
Memory	1, 2, 4, 8 GB / 4, 8 GB	1, 4 GB	4 GB
WiFi Module	Included	Included	Excluded
Power Consumption	3-10 W	6-12 W	5-10 W
Cooling Module	Excluded	Included	Included
Software Compatibility	Most ML libraries	Only with TensorFlow Lite	Most ML libraries
Community Support	Extensive / Sparse	Good	Extensive
Price	$ 60-80 / $ 100-120	$ 160	$ 147

Table 3.1. Single board computer comparisons

3.2 Camera Sensors

The concept of the RecipeCart revolves around the ability to effortlessly and accurately identify food items and ingredients based on real-time video input. Ingredient classification is done through a combination of barcode parsing, text analysis, and image recognition. Since the desired camera sensor must be able to clearly capture the barcodes and printed texts, video quality and performance are highly prioritized. Integration complexity is also strongly considered. We thus examine two classes of camera sensors: the native CSI bus camera modules and the USB cameras.

Given their direct integration and proximity to the SBC, the native CSI camera modules are optimized for computing and memory usage. They are generally smaller, provide low level control, and are highly customizable, allowing for hardware performance maximization.

The USB cameras are more high level and are thus easier to mount and integrate with the SBC; conversely, they offer less control and room for hardware performance optimization [3.7]. Since they operate over the USB bus, they are more subject to latency and consume more CPU time. USB cameras are also more expensive than the native CSI camera modules.

3.2.1 Raspberry Pi Camera Module 3

The Raspberry Pi Camera Module 3 is the newest model in the Raspberry Pi CSI-2 camera series. Its 12-megapixel Sony IMX708 image sensor allows it to record full high definition videos at 50 fps with a very high signal-to-noise ratio [3.8]. The original model has a diagonal field of view (FOV) of 75°, while the wide model is capable of expanding to 120° at the cost of a slightly worse depth resolution. The Camera Module 3 also has an auto-focus feature, allowing for dynamic depth adjustments. The lens on the Camera Module 3 are fixed and cannot be switched out.

The Camera Module 3 produces a slightly distorted far-range view when exposed to intense outdoor lighting. However, this discrepancy can be ignored given that the selected camera sensor for the RecipeCart will primarily be used for close-up views. The Camera Module 3 is subject to “exposure hunting” where it sometimes switches between a lighter and darker tone in searching for the right level of exposure.

3.2.2 Raspberry Pi High Quality Camera

The Raspberry Pi High Quality Camera Module is significantly heavier and bulkier than the Camera Module 3 but has a higher still resolution at 12.3 Megapixels. Its video resolution is worse with 2028 x 1080 pixels at 50 fps. It uses the Sony IMX477 as its camera sensor and comes with a lens mount, allowing for different lens options based on the C/CS- or M12- interfaces and an adjustable field of view [3.8]. The HQ Camera was originally designed as a higher performing alternative to the Camera Module 2. As such, in contrast to the newer Camera Module 3, the HQ Camera offers little to no added benefits in cost efficiency. The HQ Camera produces a cooler, bluer tone whereas the Camera Module 3 produces a slightly warmer, albeit darker view [3.9]. With a 6 mm wide angle lens, the HQ Camera has a variable aperture and a manual focus mechanism.

While the HQ Camera does not suffer from exposure hunting, the mounted lenses primarily determine the quality of the captured footage; a low quality lens evidently produces low quality videos.

Name	~~**Camera Module 3**~~	**HQ Camera**	**Camera Module 2**
Still Resolution	~~11.9 Megapixels~~	12.3 Megapixels	8 Megapixels
Video Resolution	~~2304 x 1296p56~~	2028 x 1080p50	640 x 480p206
Sensor	~~Sony IMX708~~	Sony IMX477	Sony IMX219
Focus	~~Motorized (Auto)~~	Adjustable	Adjustable
Field of View	~~75°~~	Adjustable	62°
Price	~~$ 25~~	$ 68	$ 25

Table 3.2. CSI-2 camera module comparisons

However, we later discover that the Jetson Nano did not yet have the necessary drivers to operate the Camera Module 3; we hence revert to the Raspberry Pi Camera Module 2. The Raspberry Pi Camera Module 2 has an 8-megapixel Sony IMX219 sensor, allowing it to record 720p videos at 60 frames per second [3.8]. Its focus can be adjusted, albeit manually via a pair of tweezers. By default, its focus is set to 50 cm—from the camera lens—which requires us to manually rotate the outer lens apparatus to shorten the focus to 10 cm, whereas the focus could be programmatically addressed with the Camera Module 3.

3.2.3 LogiTech C920e HD Webcam

The LogiTech C920e HD Webcam is a USB camera that has a max video resolution of 1920 x 1280 pixels at 30 fps. It has a diagonal field of view of 78° and provides automatic focus [3.10]. Compared to the Raspberry Pi camera modules, the C920e webcam is easier to configure and mount but has lower resolution and field of view. This USB camera offers a balanced set of features at a relatively low cost and with little setup. It is also often recommended to be paired with Jetson devices due to its easy integration.

3.2.4 NexiGo N60 Webcam

The NexiGo N60 Webcam outputs videos at equal resolution to the LogiTech C310 with 1920 x 1080 pixels at 30 fps. It has a much wider diagonal field of view of 107° and uses a 1/2.9-inch CMOS digital image sensor allowing it to adequately define objects between 20 and 118 inches from the camera [3.11]. Again, the NexiGo N60 Webcam offers high level control and ease of integration at the expense of lower image quality and focus adjustability.

Name	LogiTech C920 HD Webcam	NexiGo N60 Webcam
Video Resolution	1920 x 1080p30	1920 x 1080p30
Field of View	78°	107°
Focus	Auto	Fixed
Price	$ 58.90	$ 29.99

Table 3.3. USB camera module comparisons

3.3 Weight Sensors

In attempting to complete one of this project's advanced goals, we seek to implement a reliable and cost efficient weight measuring system. This section explores a few approaches to integrating a weight system into the RecipeCart.

3.3.1 HX711 Load Cell Amplifier Module with Load Sensors

The HX711 in itself does not measure weight; it is a load module that will be used as an amplifier and ADC, connecting to the load cells and single board computer to work as a digital interpreter—or middleman—for transmitting weighting results [3.12]. The HX711 can be integrated with the common household digital bathroom or kitchen scales or can be assembled with load cells to build a scale from scratch. The latter option involves significantly more effort and material—we may need to consider purchasing load cells, resistors, and additional wires—but offers more flexibility with the size and structure of the scale.

We thus primarily evaluate the HX711 as an implementation method to contrast with more complete weight sensor solutions. The details regarding the load cells or digital scales are neglected since the selection process for those items is fairly trivial: we simply favor the cheaper option with the more appropriate form factor. In fact, we are more inclined to use a kitchen scale given its unit precision in contrast to a bathroom scale.

The HX711's library has been officially tested under RISC-based microcontrollers and Wi-Fi microchips, as well as community-tested on the Arduino and Jetson Nano [3.12]. The load module's library will be compatible with most microcontrollers and single board computers. The HX711 and the loading sensors are also relatively affordable and simple to use. They have been around for a while, so there are many community forums and official support regarding the module, as well as many projects involving the module's usage. As a result, it is easy to set up the circuit and calibrations, as well as data collecting and broadcasting.

One problem that might occur during the process of developing the sensor is that there is a history of unstable, fluctuating value reports due to temperature affecting the performance load cells. In addition, there may be value drifting errors due to the circuit

being sensitive to supply voltage or other mechanical issues [3.13]. Therefore, implementing the weight sensor using this approach may require some additional adjustments to calibrate weight measurement errors.

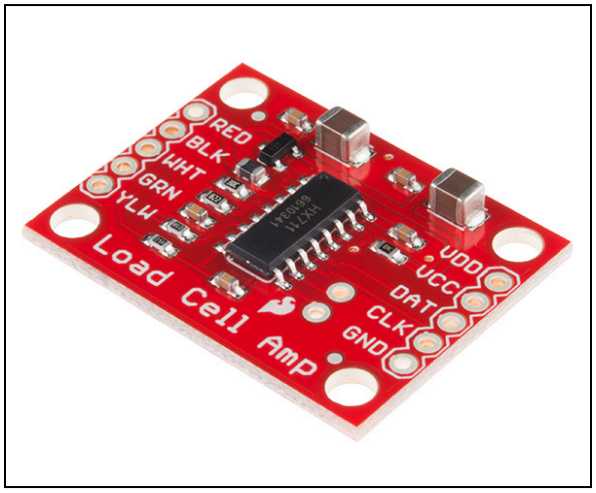


Figure 3.4. HX711 Load Cell Amplifier Module

3.3.2 EK-2000i Compact Balance

The EK-2000i, as a laboratory scale, greatly simplifies data collection through USB ports or similar converters; we can easily collect data onto our computer for simple calculations.

The scale has an FTDI RS to USB converter, which can be fed into any Raspberry Pi microcontroller for data collection [3.14]. This process should be similar for any laboratory scales. Since this allows for easy calculations, no external libraries are needed to acquire weights from the scales, and we can program straight off data collected into the microcontroller.

The balance allows accurate data collection with low risk of errors and hardware complications. In addition, since it doesn't require any external libraries, it is flexible, and can work with any microcontroller given it has the means to feed data into the system.

However, the balance is on the pricey side, with the cheapest unit being $328 [3.14]. The scale is also meant to weigh lighter objects for micro-experiment purposes. Therefore, it is hard to "scale" the project (as in measure heavier products). Another problem is calibrating the scales. Since most lab scales have manual zero buttons, it is hard to zero out the weights automatically whenever the objects are put out of the platform.

Figure 3.5. EK-2000i with FTDI Input

3.3.3 TLE Load Cell amplifier

The TLE load cell contains a built-in analog, with an LCD display for voltage and current [3.15]. This cell replaces more simpler load modules while allowing more ports for parallel load cells and simplifies the troubleshooting process by showing currents and voltage without having to use the electrician's multimeter.

The TLE contains an RS485 port that can be directly connected to any PC's serial port for data extraction [3.15]. For microcontrollers listed above, a serial port to usb adapter is needed. There is no library needed to operate the TLE to obtain and transfer data from the cells to the microprocessor. The amplifier also contains four load cell slots, which can be used to measure multiple items or heavier items.

The TLE is designed to be very user-friendly, with built-in display of volt input and analog outputs, so it is easy to set up and troubleshoot the system. The TLE contains several ports for parallel load cells implementation that works well for extracting weights in a bigger scale. It is also mountable, which helps with pre-built electrical panel enclosures.

The TLE is in the expensive range, with the cheapest product found at $190.00. In addition, the additional ports are unnecessary in our projects, where we only need one port to measure the weight of food inventory.

Figure 3.6. TLE Load Cell Amplifier

Name	**HX711**	**EK-2000i**	**TLE**
Microcontroller Compatibility	Adapter through jumper pins	Native Compatibility	Adapter needed
Programmable	Yes	No	No
Community Support	Good / Extensive Developer Support	Sparse Developer Support	Sparse Developer Support
Additional Resources	Load cells / digital scales	None	Load cell
Power Consumption	1.5 mA	*Not Specified* - 250 hours battery life	0-10VDC or 4-20mA output versions
Price	$2.00 + $18.00	$328.00	$190.00

Table 3.4. Weight sensor comparisons

Integrating a digital kitchen scale with the HX711 load cell amplifier is the most cost effective approach to implementing the weight system, although it subjects the weight values to imprecision and slight inaccuracies due to voltage fluctuations.

3.4 Power Supply Systems

The RecipeCart needs a power supply that could provide adequate wattage to the hardware components while minimizing weight and price. In determining the appropriate power system for the RecipeCart, we must observe the two primary use cases for the RecipeCart: as an extension of a smart fridge or as a component of a smart shopping cart.

As part of the smart fridge, the RecipeCart receives power directly from the wall-powered fridge, whereas it would have to be powered by batteries to maintain the mobility of a smart shopping cart. This section compares various power supplies that may be used for the RecipeCart as a smart shopping cart solution. In order to adhere to the design constraint of a maximum total product weight of 4 kg and a maximum power consumption of approximately 15 Watts, the supply should be at most 1 kg and generate at least 20 Watts.

3.4.1 ZKETECH EBD-A20H

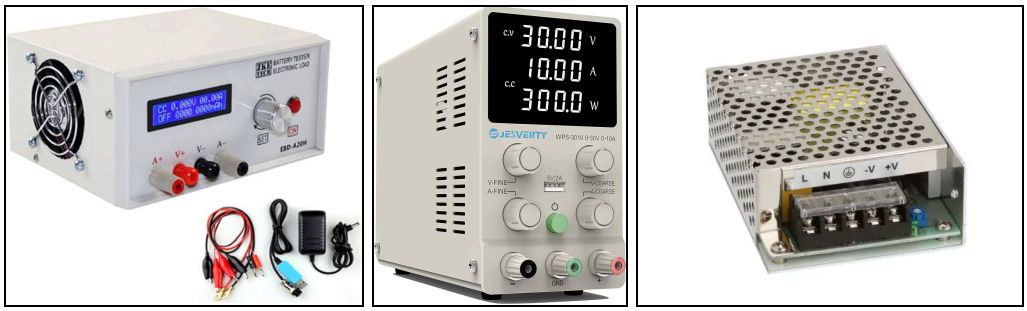
The EBD-A20H is a hybrid DC power supply / load tester that can produce up to 200W under monitor. Its settings can go up to 30.0V and 20.0 A. The supply contains a DSC-CC test mode, with voltage, current and capacity test within 0.5% error [3.16]. To monitor this, it contains an EB software to display the relevant test information. The supply is great for testing out the product, as well as monitoring hardware implementations and benchmarks. As a power supply, it provides more than enough to support the hardware components. It is on the more expensive side, and the hardwares don’t need up to 200W to function properly

3.4.2 Jesverty SPS-3010H

The SPS-3010H is a DC power supply that can produce up to 30V and 10A, including encoder knobs and LED display that provides readings of power. Aside from the grounding and sign ports it also contains a 5V/2A USB port for charging separate components if needed [3.17]. The SPS-3010H is in the cheaper range compared to other DC power supplies. It is also light, and contains silent internal fans, perfect as an add-on to consumers’ devices. Its’ main weaknesses are that the capacitor remains discharging for a long time after turning the supply off, and that it supplies peak voltage on launch, causing some potential safety issues.

3.4.3 XP POWER VCS50US12

The VCS50US12 is an AC-DC converter with adjustable output and universal input that can produce up to 12V/4.2A. It does not contain a battery, and purely functions as a power converter [3.18]. The VCS50US12 is light and it does not have any extra function other than performing AC/DC conversion and power clipper, so it needs a connection to a bigger external power source such as a wall plug. It can function between -25C to 75C [3.18], great for RecipeCart as an addon to smart fridges.



(1) **(2)** **(3)**

Figure 3.7. (1) EBD-A20H Power Supply. (2) SPS-3010H Power Supply. (3) VCS50US12 Converter

Name	**ZKETECH EBD-A20H**	**Jesverty SPS-3010H**	**VCS50US12**
Voltage draw	30 V	30 V (5V USB)	12 V
Current draw	20 A	10 A (2A USB)	4.2 A
Power Draw	200 W	300 W	50 W
Software	Yes	No	No
Weight	8 kg	1.10 kg	Ignorable
Price	$79.00	$47.99	$22.23

Table 3.5. Power supply comparisons

In designing a prototype of the RecipeCart, it is simpler to use a direct power outlet rather than a separate power supply or a pack of portable batteries. Of course, in implementing a RecipeCart for industry purposes, we would be more inclined to integrate a battery supply system to enable mobility. For the purpose of demonstrating and highlighting the RecipeCart as a proof of concept, we opt for a more minimalistic design for our prototype, with little emphasis on the hardware components beyond what is necessary for basic ingredient detection. In the scope of this project, we thus design the RecipeCart as a feature of a smart fridge, which implies a simple wall-plug power solution.

With the Jetson Nano as the core of our hardware architecture, we decided to power the entire system via a DC jack connected to the Jetson Nano's power supply port.

4.0. Software Comparisons

Building on the software diagram in Figure 2.4, this chapter describes available software that may be used towards implementing the core features of the RecipeCart. The different components are examined based on their scalability, efficacy, and cost of service. Moreover, we prioritize systems that can be run locally on a single board computer at low computational costs over dependence on server-based computation resources.

4.1 Barcode Detection Software

Barcode detection is the simplest and most efficient method of identifying ingredients. It is used in stores to check out items, and in smart fridges to keep track of resources. Since the content of the barcode contains an exact product number, it allows for precise and consistent classification of items simply by accessing a database containing the product number. Barcodes must adhere to three rules: it must be unique to each product; it must be permanently associated with the product it is assigned; and it needs to be random to ensure there is enough capacity for future products. As such, it is the preferred method of classifying ingredients for RecipeCart. Barcode detection should be the primary method of detecting and identifying ingredients due to its low error rate and high operational efficiency. As a downside, leveraging barcode detection will require access to a database containing all the product numbers and their associated information, many of which will incur a cost.

4.1.1 Dynamsoft Barcode SDK

The Dynamsoft Barcode SDK is a pretrained barcode detection model implemented in C/C++ for speed in ARM64 processors. Dynamsoft Barcode SDK works by extracting the boundaries, cropping, downscaling, grayscaling, transforming, and then extracting the barcode. It provides quick and easy access to a barcode scanner and removes the need to personally train a barcode scanning model and works well with streamed data. [4.1] Since this is an enterprise product, it performs extremely well and is optimized for efficiency and accuracy. This service requires a license for use with a 30-day free trial, after which will incur a cost about $100 per month. This may be out-of-budget for the RecipeCart prototype but may become a more viable option if we decide to make a real product.

4.1.2 Zebra crossing C++ Barcode Decoder and DaisyKit Detector

ZXing is an open-sourced kit for barcode decoding on image data. This can be used in tandem with an open-source barcode detector like DaisyKit. It does not incur a subscription cost due to being open-sourced. Both the detector and decoder have been implemented in C++ to maximize computational efficiency [4.2]. ZXing and DaisyKit are powerful options for barcode detection and processing, it incurs no cost to the project and is currently the most favored option among the different available technologies.

4.1.3 Web Barcode Detection API

Firefox and Google offer a barcode detection and scanning API, using it requires an active connection with the API and this may incur time costs due to requiring connection to an external server. This option is not open-sourced and will still require a picture or video input for barcode processing. This might result in unfeasible connectivity dependence on the server and result in high traffic. Due to the dependence on an external server to detect and scan barcodes and data streaming, the resolution of the camera will need to be minimized and this may impact accuracy. Compared to ZXing and Dynamsoft, which provide means to run detection algorithms locally, web-based barcode detection algorithms increase the dependency on external services which can incur time delays.

4.1.4 Self-Implemented OpenCV and Pyzbar Program

Another option is to program the detection algorithm using widely available libraries like OpenCV and Pyzbar, and decode the barcode. This alternative consists of using OpenCV to detect the barcode and Pyzbar to decode it. This option is freely available but since Python is noticeably slower than any compiled implementation in C/C++, there may be performance decrease. Furthermore, this option requires more work out-of-the-box than other existing options like the ZXing library.

Name	**Dynamsoft**	**DaisyKit**	**Web API**	**OpenCV**
Price	$100 / month	Free	Free	Free
Implementation complexity	Product ready to go	Product ready to go	Coding detector and requests	Coding detector and decoder
Speed	Enterprise-tier	Slow detector	Slow decoder	Slow overall
Performance	High accuracy	May not detect deformations	Decreased quality due to data streaming	May not detect deformations

Table 4.1. Barcode detection software comparisons

We ultimately opt for the self-implemented approach because it is the most cost effective option and provides more flexibility with regards to formatting and pre-processing the barcodes before forwarding it to a message broker.

4.2 Image-Based Ingredient Classification Software

Another component of RecipeCart's ingredient detection algorithm running in tandem with the barcode detection is the image classification module, which is intended to

classify items whose barcodes are not in sight or deformed. Training a scalable and robust classification model is essential to ingredient detection. The image classification module is expected to run on a server level to enable more powerful models. An ideal classification model must be able to learn to identify new ingredients quickly.

The naïve option is to simply train a model that can recognize and classify the different products. ResNet is a simple model for image classification combining Convolutional Neural Networks with residual layers to ensure limited loss of information and allows for state-of-the-art empirical performance.

However, this method is infeasible since there are thousands if not millions of different products and variations. Furthermore, having a larger number of classes does not scale well with standard classification networks. And for every new class (ingredient), the model must be retrained from scratch, which can be extremely costly.

Instead, we can train a ResNet as a general encoder model that stores latent embeddings for different items and use another method for classifying the encoded data, allowing us to do scalable reverse search on detected items.

Due to the nature of standard classification AI, simply training a regular classifier would not allow for adding new unseen products without retraining. As such, we prioritize algorithms that decouple the number of classes and the training of the detection algorithm. One popular method in industry for few-shot learning is to perform a similarity search with other learned classifications and then adopting the classifier of the k-nearest neighbors, allowing for few-shot classification and scalability with little to no retraining [4.4].

4.2.1 Google Cloud Vision

Google Cloud Vision is Google's pretrained computer vision model, capable of identifying various products from images and has strong performance as a few-shot classification model. It can easily integrate with other Google services like BigQuery and Cloud Functions for end-to-end Google-based implementations. Google Cloud Vision can work on a wide variety of image data including PNG, JPG, GIF, BMP, Raw, WebP, etc. Google Cloud Vision also allows for users to use a free trial version which is free and allows users to determine whether they wish to use the product for their intended use case. [4.5]

4.2.2 Amazon Rekognition

Amazon Rekognition is Amazon Web Service's cloud-based pretrained computer vision model. Like the Google Cloud Vision, it is able to perform few shot classification and can easily integrate with other Amazon products like AWS S3 and AWS Lambda for end-to-end pipeline implementations [4.6]. Unlike Google Cloud Vision, Rekognition only works on PNG and JPG data. While images are processed on a per use basis, Google's pricing is slightly more than that of Amazon's. [4.7]

4.2.3 Meta DINOv2

DINOv2 is Meta's pretrained computer vision model. It is able to detect depth, segmentation, and instance retrieval at high accuracy with few-shot capabilities. In addition it is open-sourced and can be implemented and integrated on the user end [4.8]. As such it does not incur any costs to the user aside from server maintenance. When combined with a Vector Database for similarity search, users may be able to perform efficient reverse vector search. Images are converted to vector embeddings and subsequently compared to a vector embedding repository for classification matches.

Name	**Google Cloud Vision**	**Amazon Rekognition**	**Meta DINOv2**
Flexibility	Classification, Image Segmentation, Image Tagging, etc.	Classification, Image Segmentation, Image Tagging, etc.	Object Detection, Image Segmentation, Image Encoding, etc.
Price	First 1000 / month free, $ 0.0015 / month after	First 5000 / month free, $ 0.001 / month after	Free
Implementation Complexity	Self-contained, complete solution	Self-contained, complete solution	Need to implement a VectorDB

Table 4.2. Image classification software comparisons

We select Meta's DINOv2 to further minimize the project's expenses while also favoring a vector-based similarity search. This approach allows for rapid scalability and classification time at the cost of increased data and storage.

4.3 Recipe Recommendation System

At the core of RecipeCart's algorithm is a recipe recommendation system. The recipe recommendation is expected to tailor and recommend recipes to users that match their user profile and diet. One major consideration is that food apps have historically little feedback from the same user, often leading to a lack of potentially insightful data. Users are not necessarily inclined to like same-tasting food and can get sick of things. An ideal recommendation system should form malleable profiles of each user and recommend a variety of different foods.

Existing methods of recommendation systems are based on similarity metrics, often leveraging similarity between users for recommendation of items or similarity between liked products. These recommendation systems can fall into two major families:

content-based filtering which recommends based on a user's preference history and recommends similar products, and collaborative filtering which leverage user profiles to recommend products that other similar users have liked. Content-based filtering methods benefit from a good embedding of recipes such that similarly preferred recipes are embedded close together while incompatible ones are repelled. Content-based filtering is susceptible to recommending similar recipes which may get repetitive for users [4.9].

An alternate approach is collaborative filtering which recommends recipes to similar users. This method escapes the pitfall of recommending repetitive recipes to users and allows variety at the added costs of learning user profiles, which tend to be dually learned with content profiles using approximate matrix factorization methods or through regression on a neural network. Approximate matrix factorization with alternating least squares tends to converge faster than neural network regression but neural network regression allows for more flexibility on how the embeddings are learned. [4.10]

Most modern recommendation systems attempt to extract the benefits of both content-based filtering and collaborative filtering by implementing a hybrid model using neural networks for combining the benefits of both filtering methods. For example, Netflix's movie recommendation system uses a combination of collaborative filtering and content-based filtering to maximize users' engagement with the platform. In doing so, it can use content-based filtering to give good initial recommendations while it builds the user profile and recommends newer and diverse films so users do not get bored with the same genre. [4.11]

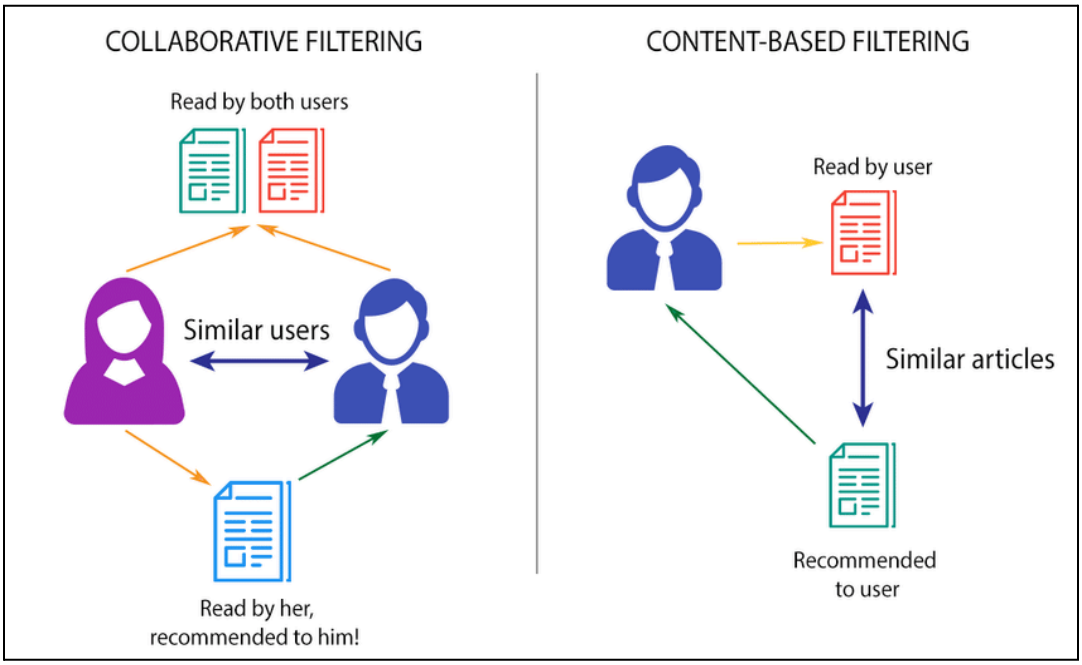


Figure 4.1 (Left) Collaborative Filtering: users with similar preferences are recommended content liked by one. (Right) Content-Based Filtering: similar content is recommended to users based on their historical preferences. [4.3]

One major issue of recommendation systems is the cold start when there has not been a learned profile yet for new recipes or users. In such cases, a neural network-based method may introduce the benefit of alleviating the setbacks from the cold start. Neural networks can be trained on feature data to extrapolate to new data which works in stark contrast to matrix factorization methods which require some user history for effectiveness.

In the context of the RecipeCart, we intend to implement a simple, content-based recipe recommendation system that filters and sorts searched recipes depending on a combination of recipe popularity, user diet and avoidances, and inventory ingredient availability.

4.4 Recipe Generation System

A reach goal of RecipeCart is to implement a human-aligned recipe generation system. There have been works on attempting this goal some have leveraged knowledge graphs while others have trained pure generative models. A notable issue with recipe generation is that the AI system does not receive direct feedback about the recipes it generates and has no sense of what is intrinsically desired by humans. One way to work around it is to perform a selection process on generated recipes based on user feedback. However, this does not resolve the issue of producing better recipes. Instead, it prevents the spread of bad ones. An ideal recipe generation model should be able to detect user preferences and tailor some recipes dedicated to users.

In theory, the recipe generation process must be closely coupled with recipe recommendation. If possible, recipe recommendation should allow a method of modifying recipes based on user input. To that end, a good design choice is to store food in a taste space where certain regions have higher weights than others.

4.4.1 RecipeGPT

RecipeGPT is an open-sourced transformer model that generates a new recipe using text generative pretraining. It generates new recipes on a token to token basis and is a fine tuned version of GPT-2. It performs pretty well on the data it is given. Given that this is a language model. Using reinforcement learning from human feedback, we can fine tune this model to better respond to user demands and preferences. [4.12]

4.4.2 RecipeMC

RecipeMC builds on RecipeGPT by incorporating a Monte-Carlo Tree Search to maximize the performance of the generated outputs of RecipeGPT. This allows the same functionalities as RecipeGPT, except in exchange for the additional compute, there will be more precision on what is generated and generated responses will be more in line with what has previously succeeded due to the added planning aspect and more control over the generated output can be exerted. [4.13]

4.5 Backend Hosting Platforms

The RecipeCart backend server is expected to run potentially intensive workloads on a variety of technologies, ranging from continuous data processing to ML inferencing. To allow for rapid bi-directional communication between the software components, the backend server needs to be adequately equipped with the proper computing resources. Procuring the necessary hardware to build and host our own server is too time-consuming and expensive. The alternate—and more ideal—solution is to deploy and host our backend on a Platform as a Service (PaaS) provider. With a cloud-based server, it is significantly easier to scale and manage the RecipeCart app based on user traffic and data processing complexity. The added benefits include easier integration with other cloud-based development services, a pay-as-you-go model, high availability and reliability, and higher operational efficiency. We thus narrow our search for a mobile app hosting platform to cloud-based services.

4.5.1 DigitalOcean

DigitalOcean provides Infrastructures as a Service (IaaS) in the form of computing resources and storage space. It offers a relatively simpler and more user-friendly interface and process for app deployment and customization among all the considered cloud computing services [4.14]. As an open-source service, DigitalOcean has extensive documentation and community support, allowing for much flexibility in app development and integration. The API is well documented and fully comprehensive, allowing for seamless integration with other development tools. It offers scalable and fully managed database options, such that developers can change their storage capacity or redundancy without much server downtime.

DigitalOcean's App Platform allows developers to build and deploy applications through a simplified web interface without direct concern for the underlying infrastructure. App Platform links to a specified GitHub repository or container registry from which it fetches all the backend code and variables [4.15][4.16]. Databases can be simply added to the architecture through a user-friendly configuration panel. DigitalOcean also provides a public URL for the app and can automatically re-deploy it when changes are detected in the backend.

App Platform does not host the mobile app but rather just the backend API and runtime environment. As such, using DigitalOcean for the RecipeCart mobile app implies incorporating other tools and services into the stack to complete the development and deployment of the mobile app itself [4.14]. DigitalOcean is fairly limited in its services and features and thus relies on external tools and add-ons to provide a complete solution.

4.5.2 Firebase

Google's Firebase is a Backend as a Service (BaaS), an abstraction of a PaaS, and features real-time databases, scalable cloud-based hosting, integrated data analytic services, authentication, and various business-tailored machine learning models. Its

databases are NoSQL-based and can be managed and updated at millisecond-level speeds. Firebase Authentication maintains UI libraries and SDKs to allow for easy user authentication configurations, leaving little operational overhead for the developers [4.17]. The server hosting and deployment process on Firebase can be simplified to a few clicks and commands on their command center interface. With Firebase, developers can spend less time on creating and managing the backend of their mobile application and focus their attention on enhancing their frontend mobile interface.

Firebase Cloud Functions allow for quick, serverless application deployment. In addition to lessening management efforts, a serverless architecture puts less strain on the backend, by processing user requests and data as they are received [4.18]. Cloud functions are only charged when triggered and are billed based on compute time and frequency [4.19]. Cloud functions can be configured to be triggered based on various events, including alerts and messages from other Google Cloud services.

The primary highlight of incorporating Firebase into the RecipeCart software architecture is Firebase's ML kit, which includes pre-made models for text extraction, face detection, image classification, and barcode scanning among others [4.18].

Using Firebase for the RecipeCart may lead to less development time spent on establishing the backend, allowing for more focus on integration with the hardware sensors and the mobile frontend. The downside is that Firebase is not open-source and provides relatively little support for and compatibility with other external software technologies. In regards to machine learning, Firebase has no available means for building and deploying custom-made models, forcing developers to primarily rely on the native Firebase ML models [4.18]. Databases are strictly NoSQL and frequent queries can be slow and expensive. Their most relevant features—the Cloud Functions and the ML kit—are not available in their free tier plan.

4.5.3 AWS Amplify

Amazon Web Services's Amplify is a complete—potentially serverless—solution to the mobile development process, featuring tools and services to create the backend and the frontend and provide mobile website hosting. AWS Amplify provides developers with datastores, allowing for easy leveraging of shared and unshared data across multi-platform local persistent storage [4.20]. In addition to the pre-trained ML services, highly sophisticated, custom machine learning models can be built, trained, and deployed through Amazon Sagemaker, AWS's native ML platform, and easily integrated with AWS Amplify. User authentication and management is also facilitated through AWS Amplify's visual interface.

The cross-platform backend environment can be initialized with authentication, GraphQL and REST APIs, and storage in the matter of minutes. Existing backends can be connected through Amplify Libraries and dedicated SDKs [4.21]. A serverless architecture can be adopted by integrating AWS Amplify with AWS Lambda functions—AWS's equivalent of Firebase's Cloud functions. Like Firebase, fully managed database services are offered to significantly minimize the operational

overhead. Existing AWS resources can be imported and accessed directly through Amplify, allowing for seamless integration between the mobile app and the AWS cloud.

AWS Amplify Studio allows for accelerated mobile UI development through direct design-to-code integration with Figma [4.20]. Amplify UI Components is an open-source UI toolkit made to easily integrate cloud-based workflows with cross-compatible UI frameworks such as JavaScript, Swift, Flutter, and Android.

Employing AWS Amplify for the RecipeCart entails a massive learning curve because of the sheer magnitude and complexity of AWS tools and resources. AWS is by far the most complex cloud-based solution among all the considered development platforms.

4.5.4 Azure Mobile Apps

Azure Mobile Apps is a highly scalable and resilient mobile application development platform that provides a framework for authentication, data query, and offline data synchronization. Designed to integrate with Azure App Service, Azure Mobile Apps comes with various community-supported SDKs, enabling cross-platform deployment and facilitated integration with existing enterprise systems [4.22]. Azure Functions, Azure's version of Firebase Cloud Functions and AWS Lambda, allows for serverless compute capabilities and are ideal for implementing the API in a mobile application backend [4.23]. Like AWS Lambda, the Azure Functions scale based on user traffic and demand and are only billed when invoked.

Custom ML models can be deployed through the Azure Machine Learning workspace and can be subsequently connected to the Azure Mobile backend. The Azure mobile app development process is well documented and community supported—but still less popular than AWS Amplify. Most NoSQL and relational databases are supported on Azure, with several services offering fully managed and scalable database solutions [4.24]. Azure also features the App Center Test, a cloud-based service that allows for automated UI tests on real-world, physical devices.

The drawbacks for using Azure Mobile Apps are in parallel with those of AWS Amplify in terms of complexity, potentially involving a high learning curve. The choice between Azure Mobile Apps and AWS Amplify thus boils down to development preference and familiarity, since architecting and deploying a cloud-based solution is its own complete task. Both services have relatively equivalent pricing options, although Azure is slightly more expensive for general purposes but conversely more cost-efficient for computing purposes [4.25].

Name	**DigitalOcean**	**Firebase**	**AWS Amplify**	**Azure Mobile**
Open-Source Compatibility	Yes	Not natively available	Yes	Yes

Custom ML Models	Yes	Not natively available	Yes	Yes
Serverless Options	None	Included	Included	Included
Mobile Development	Must rely on external SDKs	SDKs included	SDKs included	SDKs included
Supported Databases	MongoDB, Kafka, PostgreSQL, MySQL	NoSQL only	Most databases	Most databases
Mobile Testing	None	Included	Included	Included
Complexity	Low	Medium	High	High
Pricing	Starting at $ 7 / month	Starting at $ 12 / month	Starting at $ 14 / month	Starting at $ 18 / month

Table 4.3. Backend hosting platform comparisons

We select Amazon Web Services as our backend hosting platform for its wide range of services, facilitating the integration between IoT devices like the Jetson Nano with the cloud-based backend and the mobile front end.

4.6 Databases

The RecipeCart employs a mixture of databases and persistent storages. We thus require a database containing all the relevant UPC barcodes and their product information, a vector database to store vector embeddings, and a database to store all the user-driven items.

4.6.1 Barcode Databases

In order to effectively turn objects detected from hardware inputs into food vectors, we seek databases that can effectively look up barcodes and return product information, which can then be filtered into simple names to pass into the main recipe recommendation model.

First, we have a look at the UPC Database. The UPC database contains information up to 4.3 million barcode numbers from around the world, which includes the UPC and the EAN numbering systems [4.26]. They also contain an API for barcode lookup with a free starter feature. The other subscriptions are very cheap as well for accessing barcode databases, ranging between $2.50 and $25 monthly. The only weakness of the cheap price is the low allowed daily look up rate that reflects on the limits of the subscription. As a starter, it is a good free database to access product barcodes.

Another extensive barcode database is Go-UPC. It contains more than 500 million barcodes for different products and covers across all kinds of industries. The strongest advantage of this database is the amount of niche industries it covers, which includes a lot of liquor brands that may be used, allowing a wider variety of rare recipes that may catch users' interests. One main weakness of this system is the limited number of calls per month in comparison to the price, which starts at $35 monthly for the cheapest plan in exchange for 5000 API calls [4.27]. In addition, the output is displayed in the form of JSON files (and CSV for more expensive plans) which limits or requires lookup results to go over extra conversion steps.

We may also consider using Upcitemdb. It is similar to Go-UPC, being a massive database with more than 545 million barcodes for different products [4.28]. Upcitemdb contains the advantages of both the listed databases, featuring a limited free option to access through API. Its pricing is more expensive than the other two databases with a $99 monthly subscription fee in exchange for more API calls per month.

Lastly, we look at the World Open Food Facts database, which is a non-profit food products database with a collection of over 3 million registered food products [4.29]. Unlike the other UPC databases, the WOFF database primarily contains barcodes for food-related products and is completely free and open-source. The product metadata associated with each barcode is significantly richer in data and provides more relevant information into the type of food item being scanned.

Name	**UPC Database**	**Go-UPC**	**UPCitemdb**	**WOFF**
Open-Source	No	No	No	Yes
Scope	4.3 million barcodes	500 million barcodes	545 million barcodes	3.1 million barcodes
Pricing / month	$2.50-$25	$35	$99	$0

Table 4.4. Barcode Database Comparisons

We select the World Open Food Facts database as our UPC repository not only because it minimizes our software expenses but also because it provides us with just the necessary, food-related data. Although it only contains 3.1 million barcodes, all items are food-related and are thus more relevant in the scope of the RecipeCart.

4.6.2 Vector Databases

The last database that the RecipeCart wants to utilize is a database designed to help with the training process. We thus explore lookup vector databases that mirror the input, and

hook up with the trained model to produce the trained result. The vector databases listed below help storing vectors to eventually map out an appropriate training plan.

One of the databases under our consideration is Weaviate. It is a free, open-source vector database that can be deployed on the cloud, and piped through ML models to optimize and design specific next-gen search based on our goals. This database matches Recipecart's endgoal, as it is great for recommendation applications by utilizing the trained search mechanics. It is scalable and flexible, allowing for agile project designs [4.31]. Its main constraint is the database does not have as much popularity as others due to its recent deployment and lack of features. Despite this, Weaviate possesses two features which single-handedly fulfills the requirements for RecipeCart's use case: first, it allows manual input of vectors (which are useful for storing DINOv2 embeddings) and the formation of complex nested schemas (which enables graph-like structures and facilitates ingredient search), and second, it provides open-sourced modules for generating NLP embeddings (which are useful for comparing the gist of recipes).

Another database of interest is Pinecone, a vector database that allows for reverse image search. Leveraging reverse image search can allow for classification by approximate nearest neighbor to similar images [4.30]. This way, by having a few different images of the same object, we can few shot classify different items by comparing their visual similarity to other items. Pinecone allows for an efficient and enterprise-level search of items with the added bonus of enabling efficient search on top of metadata filtering to match certain conditions. However, it is proprietary and will induce a cost in production for a license.

For budget purposes and our initially relatively simple use case, we opt to employ Weaviate to store vector representation of images encoded by DINOv2 and later to store natural language representation of search terms for recipes and ingredients. It is important to note that while Weaviate itself may be cost-free, its compute and storage requirements when placed on a cloud are not. Despite this, compared to other cost-efficient databases, Weaviate scales well with increased user throughput and integrates seamlessly in the AWS Cloud landscape.

4.6.3 User-Driven Model Databases

In order to personalize users' experience, the RecipeCart wants to create a database to store signup-information and learned recipe recommenders catered to the users' inventory produced from the learned model. The database systems listed below aim to effectively manage these users' information using common hosting platforms such as AWS.

MySQL is one of the most popular databases for user information storage and querying. It is an open-source SQL database management system that is very flexible with dealing with user store and lookups simultaneously. It is simple to pick up, with high performance and scalability by utilizing multi-threading [4.32]. MySQL is free, perfect for smaller scale application developments. One known downside for this system is the low performance towards high-load traffic [4.32]. Overall, mySQL is a good free system

to utilize for startup applications with low-moderate user traffic flow, perfect for managing budgets while keeping fast lookups.

MongoDB is a more modern and different approach to the traditional query system. It is a noSQL document-based database that is primarily used for high-load data storage. MongoDB is special, that data is stored in the RAM and also allows indexing, allowing for an all-around quicker access. MongoDB is also easy to set up, and it allows integration of modern JavaScript frameworks. Most importantly, it utilizes horizontal scalability by "sharding" the acquired data, allowing data to be divided and distributed among servers to help optimize overall capacity [4.33]. One weakness of MongoDB is the nature of sharding data makes it hard to duplicate documents, as well as opening the risk of faulty indexing and resulting in data fault. Similar to mySQL, MongoDB is a good option for budget database systems.

Amazon's DynamoDB is a NoSQL, serverless solution, meaning that there is no overhead in managing the database server. Data is stored in key-value pairs and can be rapidly queried through a combination of global secondary indexes and GraphQL API. A NoSQL database makes relational data more complicated to represent but provides more flexibility, speed, and storage efficiency when properly implemented.

With the backend server deployed on AWS, we are more inclined to build the rest of our processes with AWS services. We store the recipes, ingredients, and user settings data in tables in Amazon's—serverless, NoSQL database—DynamoDB. DynamoDB is chosen over relational databases like MySQL and PostgreSQL for its scalability, allowing it to perform at millisecond-level speed even when dealing with large data. It is also natively integrated with AWS Amplify.

4.7 Mobile Front End Development Frameworks

In order to give the customer utmost convenience in optimizing their inventory, RecipeCart aims to launch the frontend platform as an IOS application, then expand to Android to accommodate all devices, and eventually PC web-app. The goal for the frontend is a personalized application with a user-base system that saves and broadcasts the inventory detected from the hardware, as well as the generated recipes from the recipe generation system.

4.7.1 Xamarin

Xamarin is a framework for developing cross-platform mobile applications with .NET (C#). It has built-in, unified support for both Android and IOS, which allows for an easy integration process. A C# native app can be wrapped so that its backend code can be shared with other platforms using a portable or .NET library [4.34], which saves a lot of development time—creating a UI for every operating system can be tedious and impractical. The framework is backed by Microsoft and is updated frequently.

The primary weakness of Xamarin is that the app building and compiling time is slow, so it is not so useful when trying to emulate more complex apps [4.34]. It is also limited

from building more high-end UI or scaling products due to the limited features sacrificed for cross-platforms development.

Overall, Xamarin should be considered when developing a multi-platform application, which is what RecipeCart is aiming for, considering its user-driven goals. Although the framework constrains the developer to a simpler UI, it is enough to develop a simple user-based system, with a basic and easy to look interface that shows them what they have and what they can make.

4.7.2 Flutter

Flutter is a development kit created by Google, designed to develop cross-platform applications. Similar to Xamarin, Flutter allows us to develop applications across different operating systems by rendering a single codebase. A Flutter system contains a framework in Dart, a graphics/accessibility engine in C/C++, and a platform-specific embedder [4.35]. Flutter is simple, as its widgets are easy to customize, and Dart is a relatively easy language to learn. It is also easy to build scalable apps using this framework, as the requirements for Flutter applications are low.

One main weakness of Flutter is the language that it uses. Dart, being a relatively new language, is not a groundbreaking one [4.35]. In comparison to other languages, third party support is not the best, as well as the lack of community support and company recommendation. As a consequence, Dart still falls short compared to other mobile languages such as Kotlin.

Despite Flutter's weaknesses as a relatively new kit with little community and documentational support, application development using this framework has been growing rapidly. It is a strong contender for developing simple mobile applications.

4.7.3 React Native

React Native is an open-source UI framework developed by Meta (Facebook), used to build mobile apps for IOS and Android, as well as compatibility with web browsers. It contains basic core components for the Native framework, and components containing wrappers commonly used for IOS and Android development. NodeJS is used as the foundation to connect the backend with the React application UI [4.36]. React Native applications can be integrated directly with the native code, which allows users to be able to take full advantage of the device [4.37]. It also has a live reload feature that allows the user to quickly view updates, test in the market and gather feedback before smoothly committing to the extensive integration and development processes.

One main weakness of React Native is that due to it running through the JS engine rather than compiling straight to the native code, its performance is lower compared to other frameworks such as Flutter or Xamarin [4.36]. In addition, the nature of layering JS on top of the native code makes updating the React engine or debugging complex application codes more complicated than usual.

Aside from the complications of maintaining React Native applications, its simplicity and ease of integration between different platforms makes it one of the most popular—and still growing—frameworks in the community. The growing public interest and usage also opens the door for support and new developments to libraries and tools. It is a perfect tool for simple projects with a common native code and easy-to-migrate UIs.

Name	**Xamarin**	**Flutter**	**React Native**
Programming Language	C# / .NET Framework	Dart	JavaScript
App Performance	Good, uses native engine	Good, uses native engine	Fair, uses JavaScript engine
Community Support	Managed by Microsoft	Open-source / Extensive	Open-source / Extensive
UI components	Native	Native	Third Party

Table 4.5. Mobile frontend framework comparisons

We pick Flutter as our front end framework of choice for its variety of out-of-the-box UI components as well as its compiling speed and relatively good support. Flutter is also supported by AWS Amplify—albeit not as well as React Native—contributing to a more seamless integration process.

5.0 Standards and Design Constraints

In light of the complexity of the RecipeCart, we review a selection of relevant design standards as well as realistic—or industry-grade—constraints that may help guide our implementation of the main hardware and software frameworks.

5.1 Related Standards

With effective and uniform usage, standards are set as conformance measures to increase efficiency, open trades, encourage widespread consumer confidence and understanding, and potentially reduce costs and efforts relating to accommodating various modalities. Although most components considered for the RecipeCart likely already follow these standards, acknowledging these standards allows for more seamless implementation and integration with other standardized inventions. Most of the relevant standards considered are taken from the Institute of Electrical and Electronics Engineers (IEEE), the

International Organization for Standardization (ISO), and the International Electrotechnical Commission (IEC).

5.1.1 Video Packaging and Streaming Standards

Given that the RecipeCart's main source of data input is video-based, it is imperative to consider standards relating to video packaging and streaming and the appropriate media data format for computer vision applications. A standardized method of compressing and streaming video-based data allows for efficient and rapid transfer of information across various electronic devices, minimizing the time spent on interpreting and converting video files. Likewise, standardized video formats promote rapid cross-device communication and ease of information dissipation.

The IEEE 1857 provides efficient coding tool sets for the compression, decompression, and packaging of video input data for telecommunication purposes . The standard defines a set of methods for video encoding and a corresponding decoding procedure [5.1]. The highlighted methods include directional intra-prediction, variable block-size inter-prediction, and context adaptive binary arithmetic coding. The standard targets services involving Internet-based videos, video surveillance footage, IP-based video conference recordings, wide-area television broadcasting, individual user-generated multimedia content, and digital video storage and communication. A popular and industry-grade single board computer—like the Jetson Nano— typically readily comes with the properly standardized video packaging and decompression software such that no additional effort is needed on the developer's part.

The Open Network Video Interface Forum (ONVIF) provides a common, standard interface between different IP-based physical security devices, regardless of their manufacturing origins [5.2]. The standard's purpose is to facilitate interoperability and data exchange between differently branded devices. The Real Time Streaming Protocol (RTSP) defines the method for video and audio transmission between two endpoints such that the latency over an internet connection is minimized. The APIs and libraries supplied by integrated camera manufacturers all follow RTSP-based methods.

The IEEE 2671 standard provides baseline requirements for data format, data quality, application scenarios, and related performance metrics for evaluating and improving online detection deployment. The standard is particularly tailored to address online defect detection through machine vision technologies for manufacturing lines, PLCs, and other backend product lines [5.3]. This standard outlines several approaches and processing pipelines for a detection algorithm software that may help guide the implementation of the RecipeCart computer vision pipeline.

5.1.2 IEEE Federated and Shared Machine Learning Standards

The crux of the RecipeCart lies in its ability to aggregate different machine learning algorithms from various repositories and sources spanned across the Internet. Furthermore, these machine learning models may need to fetch and "crawl" data from

various sources owned by distinct entities. The complexity of such an architecture warrants a defined and standardized framework for a federated machine learning system to avoid encountering any privacy or security issues.

The IEEE 3652 standard describes a machine learning framework that allows a model to crawl data distributed across various repositories or devices. The standard blueprints methods of fetching the data while abiding by the appropriate privacy, security, and regulatory requirements. Raw data should not be exchanged directly in a manner that allows a particular party to interfere or interact with the private information of another party [5.4]. With a FML framework, all forms of raw data owned by the various stakeholders are protected by secure, privacy-preserving techniques, such that no information can be leaked or reverse-engineered [5.5].

FML can be categorized into horizontal FML, vertical FML, and federated transfer learning. Horizontal and vertical federated machine learning techniques dictate a method to aggregate data sources that complement each other without directly manipulating the data itself [5.4][5.5]. Horizontal FML specifically concerns datasets with significant overlap in the feature spaces and little overlap in the sample spaces—analogous to adding more data of a similar feature set. Meanwhile, vertical FML addresses the opposite case where the aggregated data share the same sample spaces but not the feature spaces—tantamount to adding features to existing data. Alternatively, datasets that have little overlap in the informational sample space can benefit from FTL which uses transfer learning techniques to exploit reusable and common knowledge shared across different feature spaces.

A federated machine learning architecture should also be sectionalized into layers such that each layer only communicates with the layer directly adjacent to it to minimize unnecessary data exposure and privilege escalations. The IEEE 3652 splits any implementation of FML into five general layers: service, algorithm, operator, infrastructure, and cross-layer [5.4]. The service layer implements the business logic and provides a user service interface to allow FML users to access FML services for modeling and inferencing purposes. The algorithm layer contains the FML algorithmic logics and directly supports the service layer. The operator layer provides the basic operations necessary for the implementation of the more complex algorithms and includes basic machine learning tools such as aggregators, activations, regularizations, and optimizers. The infrastructure layer provides the raw capabilities of computation, storage, and communication necessary for the operator layer. The cross-layer mainly includes programs and functions for service cataloging, auditing, and monitoring across the other four main layers.

Federated machine learning is especially necessary for IoT-based, edge-computing applications, considering the need to aggregate and analyze data from various platforms to a centralized, cloud-based processing domain. The general FML framework allows independent machine learning users to jointly train a model by supplying their individual and proprietary data under the coordination of a central server [5.4]. FML gives the individual ML user more control over the devices and the accessible data, reducing privacy leakages and communication costs from a centralized perspective.

The IEEE 2830 standard additionally defines a framework for which a machine learning model is trained with encrypted data aggregated from a variety of sources and processed by a third-party trusted execution environment (TEE) [5.6]. The standard discusses shared machine learning as an alternative to FML. In TEE-based SML, the data is encrypted and forwarded to a third-party TEE to construct the model which then becomes the sharing vector. Rather than sharing the data directly, a TEE-based SML opts to compile the data and train the model directly and only share the final deployable model.

The TEE-based SML architecture features a centralized TEE equipped with the appropriate decryption and authentication modules to readily receive the encrypted data from multiple selected sources. The aggregated data is then processed and fed to the ML model as usual and the finalized model is packaged and encrypted for sharing with clients [5.6]. For increased security and privacy, the TEE can combine the crawled data like in FML via horizontal or vertical zipping or transfer learning techniques.

Using a cloud-based machine learning development platform generally implies implementing a FML or SML framework. The principal ML involvement in the RecipeCart comes from external and readily deployable models that risk violating privacy and security concerns, unless they are regularly maintained by a highly regarded service provider. As such, in compliance with the many privacy and security regulations, the RecipeCart will primarily function on reputable, cloud-based ML platforms.

5.1.3 Network Communication Integration Standards

The RecipeCart communicates with the backend server via wireless communication through a Wi-Fi module that is either readily integrated into the SBC or must be attached externally. With network communication being the primary means of binding all components of the RecipeCart together, this section investigates the standards that guide their implementation in modern communication technology, particularly highlighting the ISO's OSI model and the IEEE 802.11.

The International Organization for Standardization's (ISO) Open Systems Integration (OSI) model dissects the general network communication architecture into seven layers such that standalone systems can communicate on designated layers of operation. Each layer can directly communicate with the layers adjacent to it, but data encoded in a certain layer is encrypted and remains unmodified by subsequent enveloping layers.

Layers are sectionalized by functionality such that their respective operations are performed independent and in isolation of one another [5.7]. The physical layer includes the physical mediums and the technologies that transmit data across physical channels such as fiber-optic cables, copper-insulated wires, and air. The data link layer refers to the technologies and endpoints that operate on top of the physical layers such as MAC addresses and Ethernet ports. The network layer lies on top of the data link layer and includes protocols for routing and forwarding packets across different network nodes. The transport layer packages content and ensures that data is transmitted unaltered and in order; it includes protocols such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The session layer maintains user sessions across multiple

applications such that streamed data is mapped to the appropriate sessions. The presentation layer refers to the syntax and formatting of the application data such as JSON or HTML. Finally, the application layer consists of user-facing applications and communication protocols such as HTTPS and IMAP.

Generally, a technology may implement a group of adjacent layers rather than the entire OSI model. An outbound transmitted packet traverses from the application layer to the physical layer, with each layer adding a header to the packet. Conversely, an inbound packet traverses from the physical layer up to the application layer, with each layer stripping away and interpreting their appropriate header.

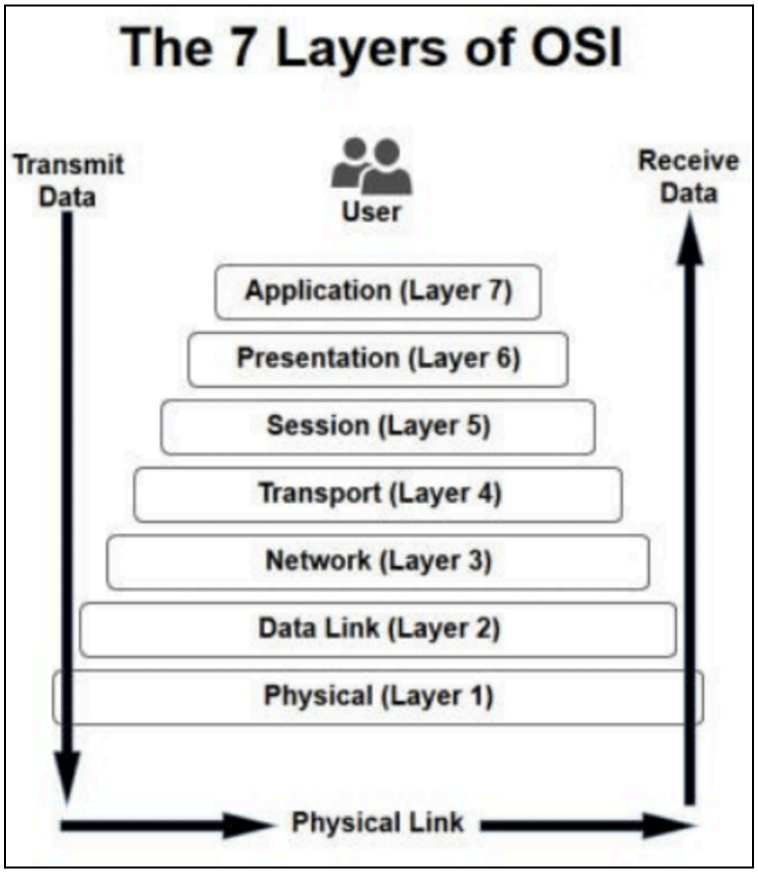


Figure 5.1. The OSI model

The IEEE 802 family of standards consists of specifications for local area networks (LANs), personal area networks (PANs), and metropolitan area networks (MANs). The 802 family contains 24 standards, most of which have been disbanded over the years due to lack of popularity [5.8]. Currently active standards include the 802.1 for higher layer LAN protocols, the 802.3 for Ethernet protocols, the 802.11 for wireless LANs, the 802.15 for wireless PANs, and the 802.18 for radio-based system regulations. These standards are regularly amended to reflect state-of-the-art capabilities and advancements, ultimately shaping the modern mediums of network connectivity.

The IEEE 802.11 was established and maintained since 1997 to specify media access control (MAC) protocols and physical hardware configurations regarding wireless local area network (WLAN) communication [5.9]. The standard describes the services and equipment needed to operate within infrastructure networks as well as in the mobile transitions between such networks. MAC protocols and translation techniques are defined such that wireless communication layers are independent of the device physical origin and do not infringe on user confidentiality. The standard also includes specifications for various common frequency bands such as the 2.4 GHz, the 5 GHz, and the 6 GHz.

Most, if not all, wireless communication protocols and devices follow the 802.11 standard, including but not limited to smartphones, laptops, air printers, and common household IoT gadgets. Wireless communication on any industry-grade SBC complies to the IEEE 802.11 protocols.

5.1.4 Image Sensor Integration Standards

The MIPI CSI-2 protocol is widely adopted for high-speed serial transmission of still and video images from camera sensors to software applications. It allows for a broad range of use cases and applications such as machine vision, imaging, biometric recognition, and contextual awareness [5.10]. The MIPI CCS streamlines the configuration of image sensors on mobile devices by defining a complete command set of functions for implementing and controlling image sensors. This command set allows for rapid integration of the MIPI CSI-2 camera with other devices without needing device-specific drivers [5.11]. The MIPI CSE refers to a set of extensions on the MIPI CSI-2 protocol for automotive and IoT applications. Its inclusion of functional safety enablers and extended virtual channels and data types allow for transmission of higher quality image data, enabling use cases such as robotics and automated visual quality control, autonomous driving systems, and virtual side mirrors [5.12].

5.1.5 Weight Scale Integration Standards

The IEEE 11073-10415 standard defines the methods of communication between personal weighing scale devices and compute engines, particularly in the context of enabling rapid integration and plug-and-play interoperability. The standard specifies term codes, formats, and behaviors in the context of broadening communication functionalities of weighing devices [5.13]. The standard establishes four general groups of object access services: GET, SET, event report, and action. Concerning the RecipeCart, the GET service is of utmost importance, since it defines the methods for retrieving valuable data and attributes from the weighing scale. The event report service may be helpful for acknowledging event triggers to indicate result readiness such as when an object has been detected on the scaling platform. The action service meanwhile can be used to switch the scale's measurement unit between the metric and imperial systems. Alas, integrating the weighing scale should—at the bare minimum—involve maintaining the GET and event report services, and optionally the action service to ensure complete interoperability.

5.1.6 Power Efficient Systems Standards

To minimize the RecipeCart's power consumption, the proposed system should limit operating in an idle state. The system should perform on low-power mode for the majority of the time, only switching to a more power-demanding mode when necessary. The triggers for dynamically alternating between different power modes may include the visual detection of an object or a direct request from the mobile app. This section highlights some standards regarding energy-efficient architectures, contrasting the advantages between power proportional and low power systems.

The IEEE 1924 standard includes guidelines for the development of power-proportional digital architectures. A power-proportional system entails that energy is consumed only during computations and is reduced during idle, non-operating states [5.14]. In such a system, power supplies must be able to respond to nanosecond variations in the computing load with respect to the complementary metal-oxide semiconductor (CMOS) static power dissipation.

The IEEE 61523 standard defines a format for low power electronic systems, specifically regarding the supply network, switches, power isolation, and power retention. In particular, the standard highlights the Unified Power Format (UPF) which specifies a set of hardware description language (HDL) packages to facilitate the expression of power units and functionalities in digital systems [5.15].

A power proportional system seeks to draw power proportional to the number of computational units or operations, while low power systems are designed to simply reduce the overall power usage in both the on and off states. As such, power proportional systems can scale their power consumption to match computational needs while low power systems cannot. Since the RecipeCart's workloads are primarily focused on machine learning inferences, we opt for an SBC that supports a power proportional system.

5.1.7 Software Development Standards and Best Practices

Software development standards guide the software systems design and implementation process such that the finished product is consistent, reliable, portable, and scalable. A standardized software also allows for easier management, debugging and testing. This section examines some software development and testing standards as well as common best practices.

The ISO 12207 standard establishes a software development lifecycle framework that is often used to support an agile workspace. The framework itself can be subdivided into primary lifecycle processes, supporting processes, and organizational processes. In complying with the ISO 12207, the project criticality and procedures must be identified and consistently surveyed by a designated administrator. Furthermore, programmers should not be burdened with administrative activities and documentation, which in turn implies that the latter be developed and managed independently of the programming

team. This delegation of responsibility ensures that the software can last beyond the employment of the involved entities [5.16].

The ISO 29119 discusses best practices for software and system testing such that testing techniques are not tied to a particular development model. The standard requires documentation of test automation in the form of session sheets with test case specifications, a defect report with the relevant acceptance criteria, and a report on test results [5.17][5.18]. The ISO 30130 defines the framework for software testing tools to enable rapid and continuous deployment of high-quality software [5.19]. The standard explores functional and nonfunctional testing as well as different methods of unit and system testing. Specifically, the ISO 30130 focuses on categorizing software test entities and the associated tools, distinguishing each software testing category, and mapping testing tools to capabilities and potential use cases.

For product consistency, reliability, and portability, software must be developed following common best practices. As emphasized by the ISO 29119 and ISO 30130, system testing should be done early in the development cycle to identify issues and resolve bugs before the code increases in complexity. More importantly, testing should be automated to reduce human error while also allowing for more rapid and diversified testing coverage [5.20]. Code should be simplified and sectionalized whenever possible to improve readability and allow for unit testing. In a similar manner, classes, objects, and functions should occupy their own file. Version control should also be maintained at all times to track changes and enable the necessary rollbacks.

A continuous integration and continuous deployment pipeline leverages automation to roll in new software updates and changes without having to re-integrate the entire existing system [5.21]. The continuous integration phase includes automation for building, testing, and merging the software, while the continuous delivery and deployment involve automatically releasing the updated software in the repository and deploying it to production, respectively.

An agile working environment allows for collaboration and flexibility by delegating specific tasks and roles to designated people. The agile methodology involves decomposing a particular software project into phases in a cycle of continuous planning, execution, and evaluation [5.22]. The agile methodology is centered around constantly adapting the software to meet customer needs. The RecipeCart may particularly benefit from an agile methodology because of its intrinsic focus on user feedback regarding the recipe generation and recommendation systems as well as the ingredient detection.

5.2 Realistic Design Constraints

In order to realize the Recipecart, a few guidelines are put in place, reflecting different physical aspects that the product must meet in order to comfortably go into deployment. The conditions described below lists different aspects our application must follow in order to ensure accomplishment within the allotted time and budget, as well as safety of the product not only for the end user but also for the surrounding environments.

5.2.1 Financial and Time Constraints

Given the goal of the project, most of the costs should cover the implementation of the hardware to mimic the performance of a smart inventory system with the ability to detect object input, then process and output the information regarding that input. Given the software commitments on the development process of the RecipeCart, hardware pricing should be kept minimal despite taking the majority of the budget, and should only perform enough to provide inputs and broadcast to the training model. The remaining budget will mostly be the coverage of hosting database servers on the software side. Depending on the audience the deployment plans to target, the servers hosting cost should cover between a hundred and three hundred concurrent users at a time with at most a two second display latency.

The RecipeCart aims to be a software-heavy project. As a result, we seek to minimize time spent on the hardware aspect, as the main purpose is to mimic smart inventory. This process should not last longer than a third of the development process, and the budget prototype should do enough to at least pass the object input in the form of images to pass information for preliminary training. The software process will occur simultaneously and constantly throughout the whole development period. The majority of time will be spent on developing and optimizing algorithms so that we can maximize our performance on the recipe-generation model. Development of the front-end will also be extensive, as we aim to produce a user-friendly environment, so application interaction should be comfortable and easy to navigate. Recipecart is a solid project that will need to be tested on the market and adjusted according to reviews due to the applications' dependency on users feedback. Therefore, flexible approaches such as Agile should be integrated to quickly deploy the fundamentals of our product, then acquire reviews and update the application accordingly.

5.2.2 Environmental, Social, and Political Constraints

During the development process for Recipecart, aspects regarding the hardware development should be addressed. Firstly, the inventory that will be created to showcase the software prototype aims to be made of a commercial recyclable plastic container that should already have safety-conditions measured [5.23]. The inventory should not have additional build-ons that may cause chemical changes to the inventory itself, and after the showcase it will either be safely disposed of or contained for reusage.

Another consideration is regarding the software aspect, knowing that over-dependent on hosting elaborate servers may cause unnecessary waste and energy consumption, hosting servers should be minimized so that it only fits within the designated range of users in mind [5.23]. Other hardware parts such as cameras and weight measuring technology can also consider reusing old hardwares to minimize environmental waste. The power system, if not designed to integrate to a wall plug, will be selectively chosen so that it is renewable or utilizes solar power. Lithium batteries will be avoided.

The Recipecart, as a software, will be designed so that it will be available to as many users that have access to smart inventory devices. The primary access to the software will

be a user-friendly phone app that keeps track of the inventory, generates and suggests different recipes to allow users to create dishes while maximizing their resources. Overall, RecipeCart aims to become a convenient add-on that promotes optimization and effective planning in consumers' daily lives.

The RecipeCart will be open-source, promoting others to collaborate and improve the software as a whole. It will only collect the user's inventory as input and purely use that data to generate personalized outputs. Other forms of data collection, such as age or household number, will be in the form of optional surveys and abstractive so that the application can improve without taking in unnecessary information that may cause harm to users' privacy.

The RecipeCart keeps up to date with any federal restrictions, as well as safety measures and ethical approach towards data collection. As a brand, it is best for the software to reinforce and protect users' data by preventing any kind of information, especially users login information, from potential malwares or leakage.

5.2.3 Health, Privacy, and Safety Constraints

The RecipeCart should take in the users' physical and mental health into consideration during its deployment. Recipe generations should prioritize healthy products in order to promote a healthy lifestyle [5.24]. The application's interface should also be clean and display all products inside the user's inventory to help the user keep track of their ingredient storage, which helps relieve some stress relating to daily planning and organization. Secured data collection and storage are also important, as they relieve the users from fear of getting their data breached, as well as building trust between the customer and the application. As a consumer-based product, it is important for the RecipeCart to uphold these characteristics as an application to deliver all listed features above efficiently to all users to ensure healthy connection and lifestyles.

Most importantly, in order to ensure our users' health throughout the RecipeCart's deployment, our trained recommendation system must prioritize both high accuracy and precision. The suggested recipes should match with the macros that the average human needs everyday, and should not under- or overestimate any amount that will be suggested to the user to make into reality. Additionally, the generated recipes must be reasonably safe for consumption, reflecting proper proportions, quantities, and ingredient combinations. The model's macro accuracy should be at a 99.0% accuracy, with at least 95.0% precision, and all of these generated recipes must be safe for consumption. Users' health and safety are extremely important, which is why the RecipeCart must maintain a high accuracy standard regarding its recipe recommendation system.

As stated above, the RecipeCart prioritizes users' security. One approach to this is only collecting necessary data and minimizing the extent of data collection. This can be done by implementing extensive, clear and informed consent processes for data collection and usage. We should also ensure robust measures to protect user data from unauthorized access or breaches [5.24]. The application also aims to implement methods that allow anonymous users behaviors, only allowing learned data to pass when they launch the

application and keep it saved in the hardware in case servers get breached beyond our reach.

Careful implementation of cloud security structure is also necessary to ensure the promised secured delivery of the product. This is accomplished by complying with relevant safety standards and certifications for hardware or software products, and providing users with necessary instructions and safety guidelines to minimize accidents or misuse. Most importantly, the application should be regularly updated and patched so that the software can address vulnerabilities that could compromise user safety.

5.2.4 Manufacturability and Sustainability Constraints

The RecipeCart aims to be an add-on software product to smart inventory devices. This means that it can be integrated to different hardware, assuming they have the same input hardware features, and the input information can be piped into the system to run training models. The product should also be scalable, avoiding unconventional methods that do not effectively optimize the overall production stage. It should also avoid environmentally damaging building blocks such as lithium-powered batteries. For the software aspect, the RecipeCart application should be accessed for most devices across different operating systems (iOS and Android for primary phone applications, and potentially expand to Windows and MacOS to cover both platforms) in order to ensure user friendliness.

Regarding sustainability, the RecipeCart is more concerned with the longevity of the application. This calls for usage of promising or popular technologies that can guarantee operation for at least 5-7 years, as well as usage of stable libraries that are constantly kept up to date. Older hardware reuse may be considered as a part of the development process in order to minimize waste in general.

6.0 ChatGPT Applications and Limitations

ChatGPT was designed by the artificial intelligence research lab OpenAI, who have made significant contributions to the generative AI including but not limited to GPT-3 which set new benchmarks for generation-based natural language processing, Codex which can be used as the backbone for GitHub Copilot by enabling autocomplete-style suggestions, and Dalle-3, which generates high-resolution images based on text instructions.

ChatGPT has explosively risen to prominence due to its versatility and ability to perform beyond its initial scope. Because of this, Large Language Models have gained much attention from industry giants and scientists alike. Many organizations have begun to incorporate LLMs into their support chatbots and as workflow assistants. Among these LLMs, ChatGPT has distinguished itself as a game-changing technology that many speculate to show early sparks of Artificial General Intelligence.

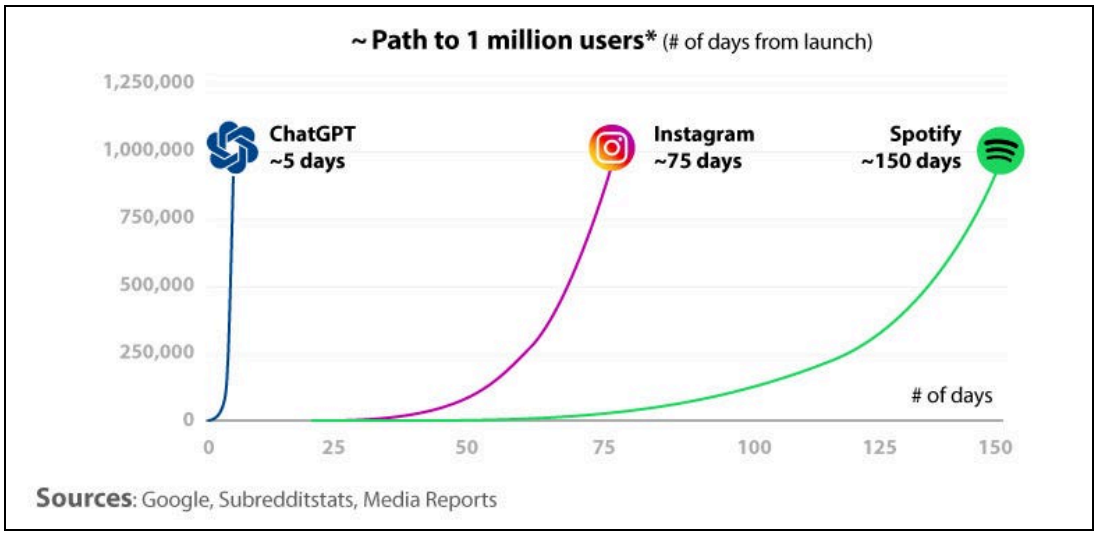


Figure 6.1. Users vs Days since release for ChatGPT vs popular platforms [6.1]

ChatGPT has disrupted much of content creation and search as we know it, with its ability to construct mostly-coherent and human-like responses to user prompts and its extremely user-friendly interface. ChatGPT was trained on a wide range of conversational text and fine-tuned to several specific tasks such as question-answering and dialogue generation. This section explores ChatGPT and its potential application to our products and its limitations. [6.2]

6.1 Large Language Model Historical Background

Big Tech companies have always dedicated effort towards research on making better and more robust natural language processing models. This focus is driven by LLMs' ability to determine complex dynamics between words and allows companies to tap into the large data reserves of the internet and form meaningful insights on the available data.

Prior to the introduction of the transformer, NLP architecture largely revolved around RNNs and its gated variants like the GRU and LSTM. However, RNNs were subject to instabilities that simple feedforward did not suffer from. Moreover, RNNs due to their design required sequential processing and could not be parallelized, resulting in slow training times and convergence, and most solutions to its stability incurred additional costs to training time. Once the Transformer architecture enabled parallelizable and highly efficient natural language translation. It largely became the core of the success behind current state-of-the-art LLMs. Shortly after its release, Google released BERT which used the encoder of the transformer to receive state-of-the-art in NLP benchmarks. Concurrently, OpenAI released GPT which used the decoder aspect of the transformer to generate human-like text and achieve high accuracy in its performance.

6.2 ChatGPT Architecture

Previous language generation models did not have an interface, making it difficult for non-developers to fully leverage. ChatGPT is designed for non-technical users to be able to use without much configuration. This design philosophy is embedded in the underlying architecture of ChatGPT. The GPT-3 Model is fine tuned by human-feedback Reinforcement Learning and then a safety layer is added to ensure appropriate information is generated.

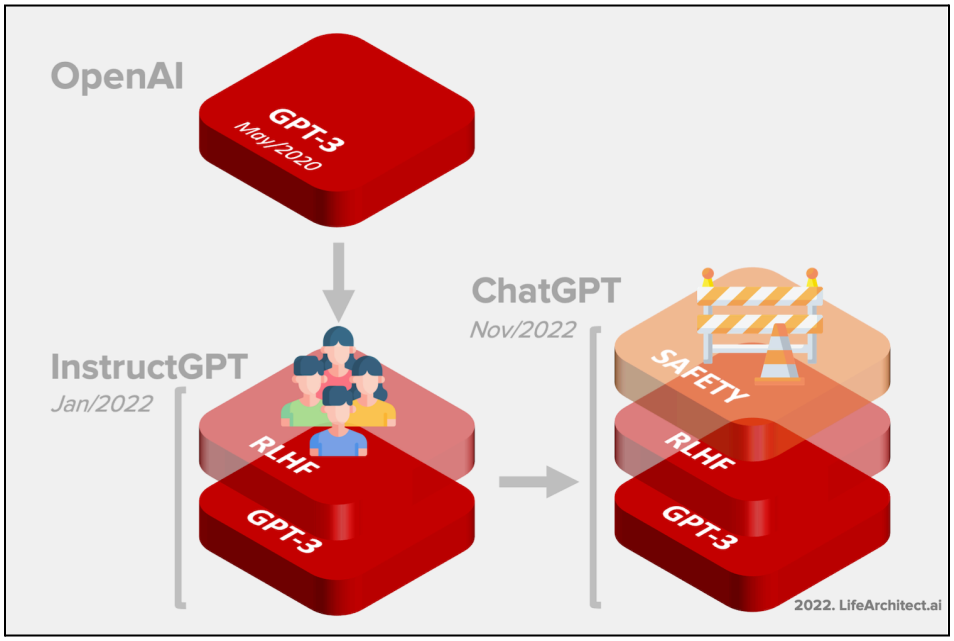


Figure 6.2. ChatGPT general architecture [6.3]

6.2.1 Generative Pre-Training

ChatGPT was trained on a pre-trained GPT-3 model. Using the decoder for the Transformer architecture, GPT was trained to predict subwords called tokens in an autoregressive manner—that is to say, one by one. Doing so, GPT was able to learn semantic embeddings for each token, allowing it to emulate context-aware semantically-sound content. [6.3]

6.2.2 Reward Model

Once GPT-3 has been trained, it now needs to produce content that is desired by the user instead of irrelevant and unrestricted human-like blabber. OpenAI's solution to this was to introduce reinforcement learning from human feedback (RLHF). People were requested to write responses to sampled prompts and GPT was fine-tuned to predict desired responses to the prompts. People were then requested to rank generated responses according to the best to the worst. This would train a reinforcement learned reward model which would allow the model to predict which responses best respond to the required

context. GPT is then fine-tuned according to the reward model, and this allows for responsive generated content. And this cycle of human feedback and self-supervised reinforcement learning would repeat until the model reached a sufficient level of performance. The model this yielded was called InstructGPT and it was almost ChatGPT except it did not have a filter on what kinds of content it outputted. [6.3]

6.2.3 Content Filter

After GPT has been fine tuned to generate content based on a given task. It must now be filtered to prevent the generation of inaccurate or inappropriate information. This is done by applying a meta-filter over the generated content. Generating appropriate content is framed as a meta prompt to every sub prompt. [6.3]

6.3 Analysis of Originality and Notable Limitations

ChatGPT is hailed for generating “original " content, but there exist some notable caveats to the extent of the originality of the generated content. Given the way the GPT model is trained, where it is trained on regenerating the original content of its dataset, and trained to decrease the difference between its generated content and the dataset. It is severely subject to reiterating content of its dataset without actually generating novel and thought-provoking content. To add to the matter, ChatGPT does not plan its responses nor does it backtrack to correct information. As such, its generated content, more often than not, is some form of reformulated content from the dataset it was given. Contents that seem well-thought out are often more wordy than necessary due to the tokens needed to be placed for the model to process information.

Despite this, ChatGPT is still capable of generating plausible-sounding yet incorrect but original content. In a study, where reviewers were blinded and reviewed abstracts between real content and ChatGPT generated content, 32% of the generated abstracts fooled reviewers into believing it is human-generated content [6.4].

6.4 Prompt Engineering

While ChatGPT is not capable of independent thought, it is a useful tool for eloquating and verbalizing its given human input. By giving it either sufficient feedback or sufficient content, users can inject knowledge into ChatGPT which can lead to better responses. This insight has led to the rise of prompt-engineering, in which prompts are engineered to draw out more out of the model than it was initially designed to. With detailed and engineered prompts and given explicit information, ChatGPT can be trained to elicit certain responses that seem to have more coherence. [6.5]

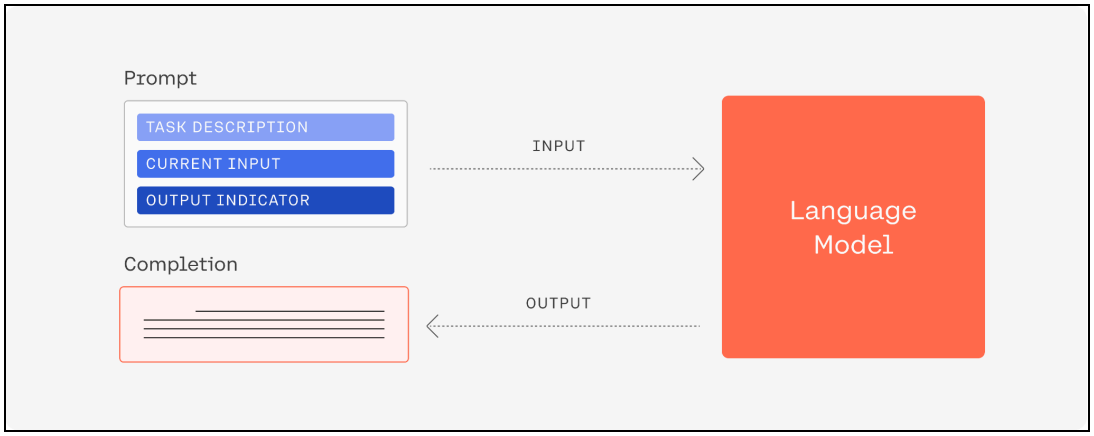


Figure 6.3. Prompt Engineering [6.5]

Users have developed a method to simulate critical thinking in ChatGPT by making it elaborate on its generated content and force it to programmatically perform tasks at a step-by-step basis, resulting in stronger and more consistent performance at logical tasks.

6.5 Potential Integration with RecipeCart

For the purposes of our project, we consider a few options for incorporating LLMs into our model ranging from simply having ChatGPT recommend a recipe to having it react to user feedback and conversationally manage complaints.

6.5.1 Recipe Generation by Prompt Engineering

One option of integrating chat-based generative LLMs into RecipeCart is to prompt the model to generate recipe modification suggestions like increasing the salt or decreasing other ingredients or changing the cooking time. Such changes require a semantic awareness of each cooking instruction and a view of the ingredients involved. By using prompt engineering and user-feedback, slowly but surely the recipe will be fine tuned to the user. However, there are a few noticeable caveats. Unlike in text generation-based training like for ChatGPT, recipe-based fine tuning will require the recipe to be cooked and this will cost time and money to make and learn. Additionally, the autoregressive text generation may not be an optimal model for representing recipes and their ingredients.

6.5.2 Review Tagging System

Another option for integrating LLMs is to generate tags for textual reviews that can be aggregated and sent to the recipe author or a recipe generation model for improvement. This is similar to the first idea except that the recipe generation is decoupled with the language model, allowing it to use embeddings more appropriate to recipes and ingredients.

6.6 Recent or Related Developments of ChatGPT

Since its initial release in 2021, ChatGPT has undergone a few notable changes that have improved its performance. Firstly, its base model was upgraded with GPT-3.5 and GPT-4, the latter only being available to premium users. The GPT-4 model has more than ten times the number of parameters of the original GPT-3 model, can process images and documents, and can integrate easily with external tools like matlab to compensate for its inability to perform calculations. Though these models enhance the abilities of ChatGPT, many of these modifications do not yet resolve the underlying limitations. [6.6]

Microsoft has incorporated a chat LLM based on the Prometheus model which later became ChatGPT to Bing Search. This has led to AI-assisted searches. However, in the early days of Bing Search AI, there have been numerous attempts to prompt inject Bing AI. In response to this, Microsoft has drastically limited the capabilities of Bing AI, applying limited prompt context, and removing non-search related conversation. [6.7]

In response to ChatGPT, Google introduced Bard, which is a chatbot based on the LaMDA family of models which later became PaLM and PaLM-2. Bard has similar functionalities as ChatGPT except it is able to access data past the year 2021. However, due to some complications during its release, Bard has lost much of the traction it could have gotten at the peak of the Chatbot hype. [6.8]

LLaMA is another chatbot AI developed by Meta. Using only 65 million parameters as compared to GPT-3's millions of parameters, it is trained longer on significantly less parameters than ChatGPT, this makes it more viable for lower end hardware. LLaMa-2 has been released by Meta and the model is available on a case-by-case basis. [6.9]

Concurrently, there have been attempts to extend ChatGPT by iteratively prompting it to elaborate on its own outputs to form more and more clear pictures of abstract ideas. One notable model is the Voyager model which uses GPT-4 to code functions through iterative prompting for navigating Minecraft or other exploratory games, and construct a skill library which can be stored and retrieved from a vector database. From this result, the power of the zero-shot generalization that LLMs can provide brightens many avenues for research in extracting knowledge from LLMs. [6.10]

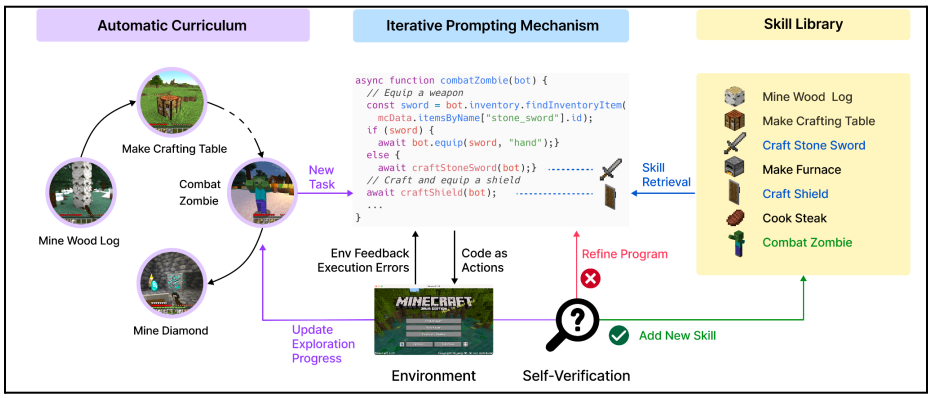


Figure 6.4. Sketch of Voyager's pipeline [6.10]

7.0 Hardware Design

The hardware design of the RecipeCart revolves around the principal hardware components explored in Chapter 3: the single board computer (SBC), the MIPI CSI-2 camera, the weight scaling system, and the power supply system. The design process itself simply involves direct integration of the camera and weight scaling system to the SBC, while providing an adequate and reliable power supply to all of them. Having obtained the permissions to use the Jetson Nano as a complete solution to the microcontroller and PCB design requirements, the brunt of the hardware-related workload is dedicated to the implementation of a weighting system with the HX711 load cell amplifier and an external scaling platform. In contrast, the installation and configuration of the Raspberry Pi camera is relatively trivial, given its compatibility with the Jetson Nano.

This section hence details the overall hardware architecture and explores the design schematics of the Jetson Nano, the Raspberry Pi Camera Module 3, and the HX711 load cell amplifier. The bills of materials associated with complete integration of each hardware component are also included to provide a holistic understanding of design costs and efforts.

7.1 Initial Hardware Architectures

The purpose of the RecipeCart is to demonstrate the portability and compactness of the ingredient detection solution. All the hardware components are thus to be assembled onto a portable container. With the SBC functioning as the heart of the architecture, the objective is to place the camera and loading platform in areas of close proximity to the Jetson Nano to minimize broadcast delay and wire entanglement.

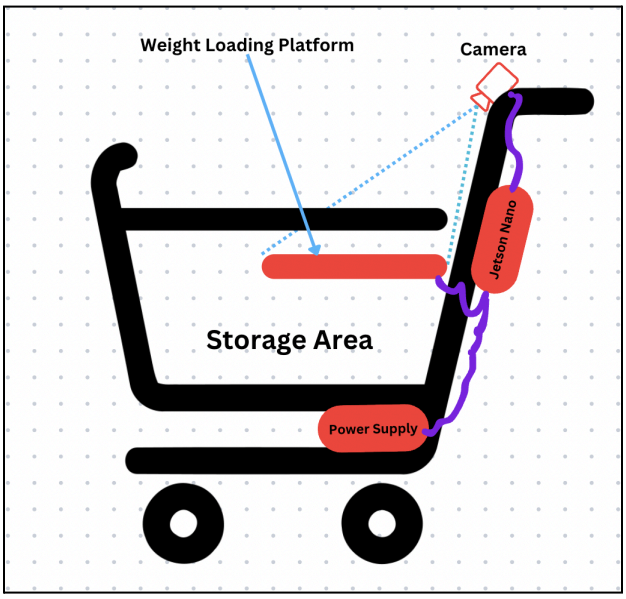


Figure 7.1. RecipeCart original physical architecture as intended

Figure 7.1 describes the RecipeCart's physical design as originally intended as per its namesake. The design adheres to the requirements of installing the SBC in a centralized location with direct connectivity to all other hardware modules via the purple-colored wires. Users scan the ingredient of interest by hovering the barcode in front of the camera and subsequently place the item into the cart's storage area. In the absence of a barcode, the user may place the item on the weight loading platform, triggering the image-based object classification pipeline. This design also entails obtaining a real-size shopping cart to use as a vehicle, which may be challenging and arguably unnecessary to purchase. Instead, we opt for a more simplified design that involves much more readily available materials but nonetheless embodies the portability principles of the original RecipeCart design.

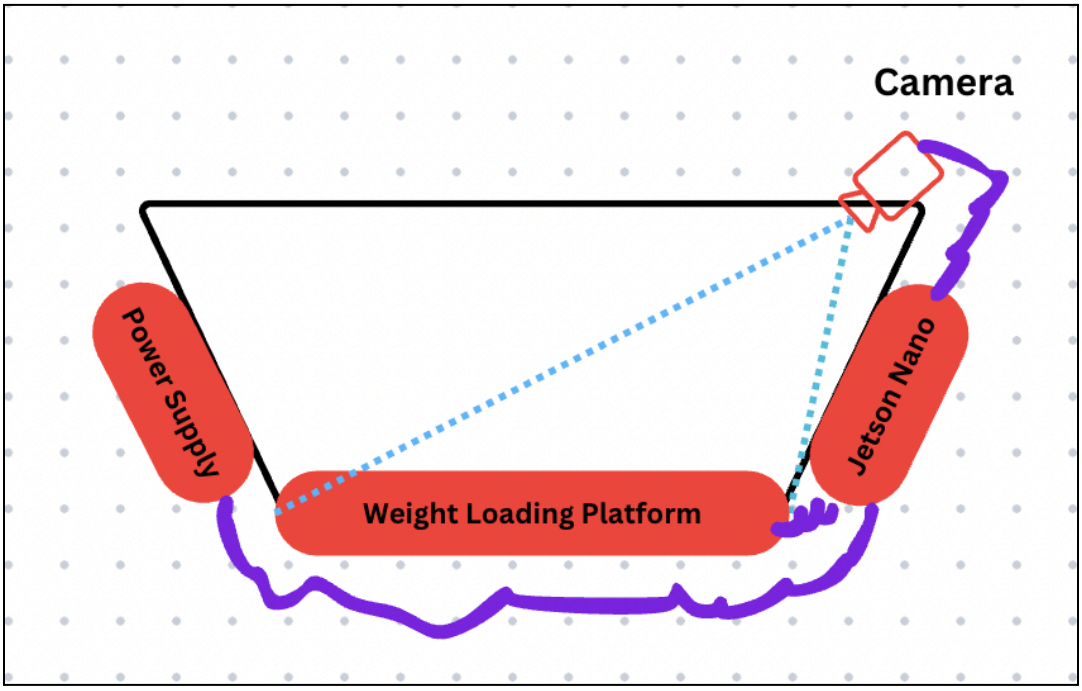


Figure 7.2. Proposed RecipeCart hardware architecture

Figure 7.2 shows the simplified prototype version of the RecipeCart. In this design, we replace the shopping cart with a wide, open-top container with a flat bottom. The specifics of the container are trivial so long as the dimensions are sufficiently large for the weight loading platform and the sides are block-colored and non-transparent.

The Jetson Nano is again sandwiched between the camera and the weight loading platform. The power supply is shown to be mounted on the side opposite to the Jetson Nano for illustration purposes only and may be relocated to a closer position upon implementation. In fact, since the main focus of the hardware design will be the ingredient detection and classification, the power supply's placement will be relatively trivial.

7.2 Jetson Nano Design and Peripherals Overview

In the absence of a custom printed circuit board design, this section presents an overview of the Jetson Nano carrier board design. The carrier board houses the Jetson Nano's compute and power control modules and is optimized for compute capabilities in power-limited, edge-computing environments [7.3]. The featured Nvidia Maxwell architecture is designed to extract maximum performance per consumed watt.

The carrier board combines multiple sets of GPIOs, system clocks, and data communication lanes for each available feature. There are two camera connector lanes, with each supporting the CSI-2 interface. Serial data from the cameras are controlled via two independent master clocks and GPIOs, and camera commands can be issued via I2C. The carrier board also supports USB 2.0 Type A and Type B which are used to hook up a keyboard and mouse to navigate and configure the Jetson Nano. The power module consists of two channels: a DC jack which directly feeds into the carrier board's V_{DD} and a microUSB 2.0 port to provide power to the board via a secondary machine. On the Jetson Nano itself, in addition to the Quad-core Cortex-A57 CPU, we find the 4GB DDR4 RAM memory module and the 4MB QSPI-NOR flash memory module. There is also an HDMI Type A port and a DisplayPort for accessing the Jetson Nano operating system GUI. [7.1]

Since the Jetson Nano does not readily come with wireless connectivity, the board also contains a socket for an optional WiFi and Bluetooth module. WiFi data is transmitted through a PCIe lane with its own clock and control system, while Bluetooth data is governed under I2S and UART. Both modules have their own GPIOs. The carrier board contains some additional "expansion" connectors, which may be relevant for integrating the HX711 load cell amplifier with the rest of the hardware architecture via jump wires.

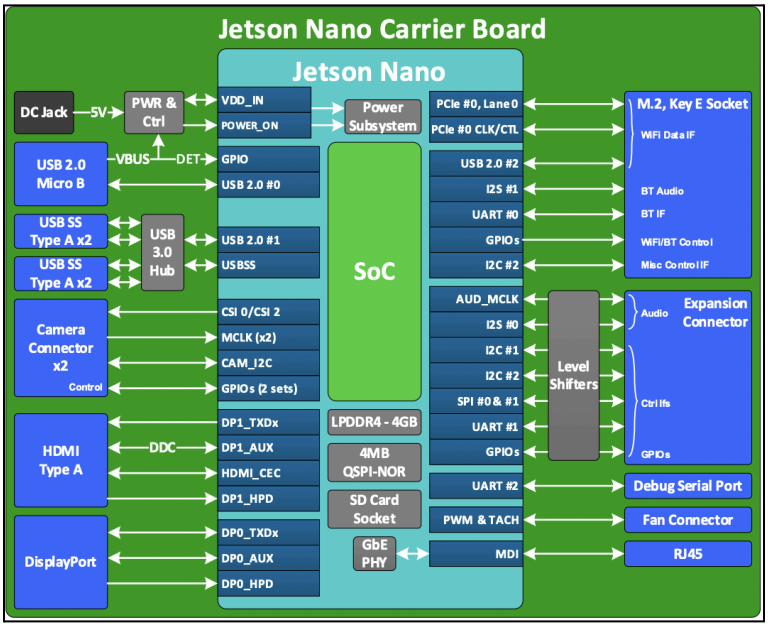


Figure 7.3. Jetson Nano carrier board block diagram [7.1]

The carrier board also supports audio data channels, but this feature is relatively irrelevant to the design of the RecipeCart.

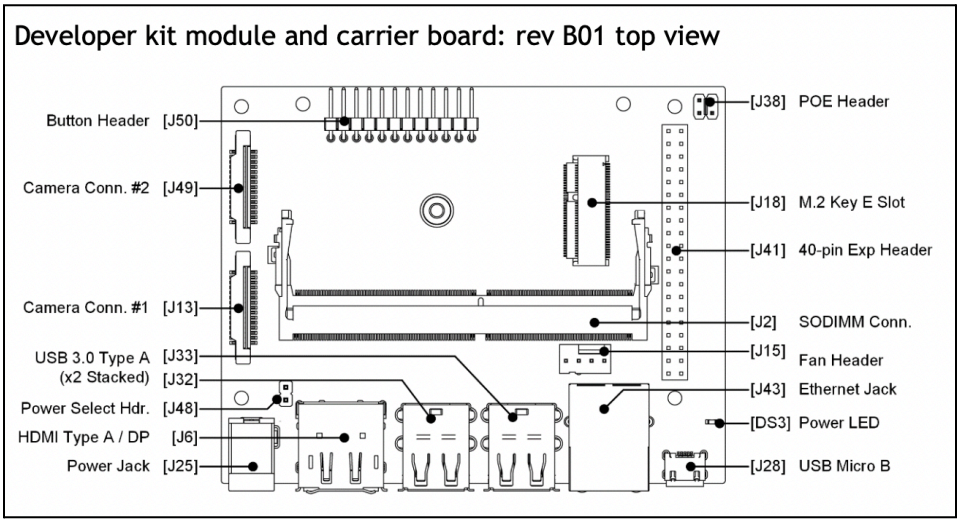


Figure 7.4. Jetson Nano carrier board schematic [7.2]

The above schematic breaks down the Jetson Nano carrier board into its various hardware components. The components of interest are the camera connectors located at J13 and J49, the M.2 key slot at J18, the 40-pin header at J41, the USB 3.0 ports at J32 and J33, the power jack at J25, and the button header at J50—for extraneous needs.

7.2.1 Wireless Communication Modules

To enable wireless communication via both WiFi and Bluetooth, we equip the Jetson Nano with a wireless AC8265 NIC card module with two antennas, ensuring that the obtained AC8265 NIC module has a form factor fit for the Jetson Nano. The AC8265 follows the IEEE 802.11ac standard for WiFi and implements dual mode Bluetooth 4.2, enabling the Jetson Nano to act both as a hub and a peripheral simultaneously.

Figure 7.5. AC8265 NIC module with two antennas

7.2.2 Data Storage and MicroSD Card Slot

To properly operate the Jetson Nano, we must provide it with adequate storage space via a microSD card, which fits into a slot located under the Jetson Nano compute module. A generic 64GB microSD card with performant read/write speed—170MB/s read and 80MB/s write—provides sufficient storage space for the Linux-based operating system with ample room for future developments.

7.2.3 Jetson Nano Power Jack

To power the Jetson Nano, we opt for a 5V DC power jack rather than a microUSB power supply. A DC power jack provides power to the system more consistently, especially under constant and strenuous computational tasks. Any generic 5V/4A DC power jack suffices; we opt for the cheapest available option.

7.2.4 Raspberry Pi Camera Module 2 Mechanical Schematics

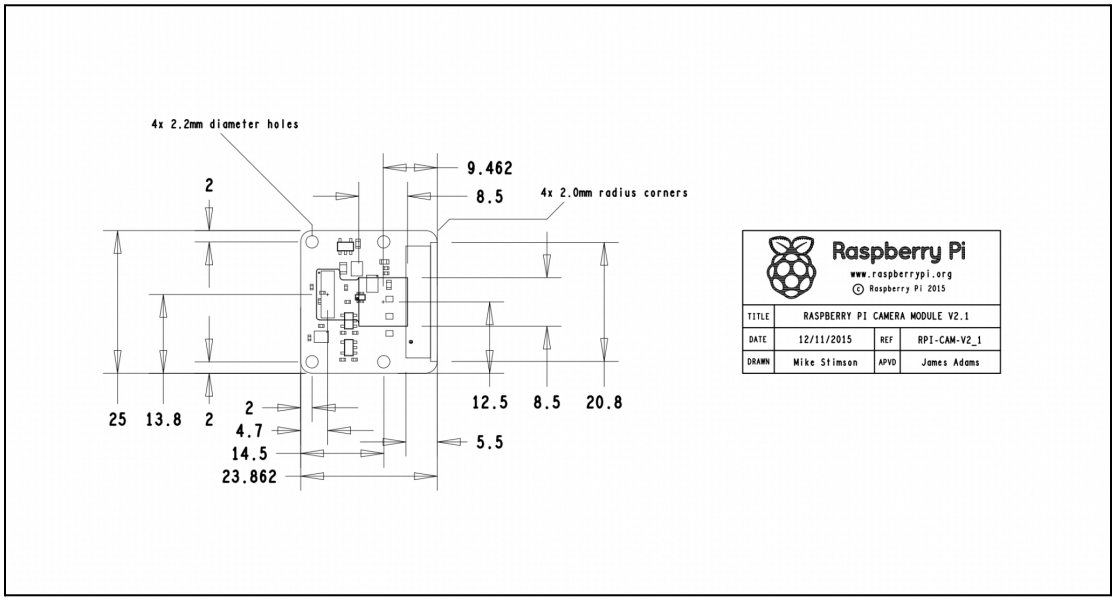


Figure 7.6. Camera Module 2 mechanical schematics (in mm) [7.4]

The Raspberry Pi Camera Module 2 is relatively small and compact in design as suggested in Figure 7.6. Attached to the bottom of the camera module is a band that can be directly inserted into the CSI-2 lane upon lifting the locking latch. The band itself is short, preventing the camera module from extending beyond the Jetson Nano's immediate proximity. A cable extension kit may be included to mitigate this issue for an additional cost ranging between $8 and $60; the extension method involves using the Jetson Nano's Ethernet port as an intermediate. For demonstrational purposes regarding the RecipeCart prototype suggested in Figure 7.2, an extension kit may be unnecessary as the Jetson Nano can be easily moved to accommodate the camera's placement.

7.3 Weight Scale System Design

Our design of the weight scale system leverages the HX711's ability to extract and amplify weight data from a generic digital bathroom scale. The HX711 is directly connected to the PCB embedded inside the bathroom scale's main circuitry compartment. This entire process is the most challenging hardware-related task in the project. This section examines the HX711 schematics in detail and outlines the assembly process.

7.3.1 HX711 Load Cell Module Schematics

The below schematic is obtained from SparkFun Electronics and depicts the HX711 PCB design. With most of the shown outlets being grounded, only ten pins are physically visible on the board. The E+ pin refers to the positive end of the board and connects to its V_{CC}, while the E- pin refers to the negative end or its ground. The A+ and A- pins represent the amplifier circuits and are for connecting to the white and green/blue wires on the load cells, respectively. Along with the yellow shield pin, the E+, E-, A+, and A- pins are dedicated to receiving input from the load cells [7.5]. On the other side of the HX711 PCB are the CLK and DATA gates, which are useful for regulating board operations and data transmission, respectively. Finally, we observe two gates for transmitting power to the HX711 and another GND.

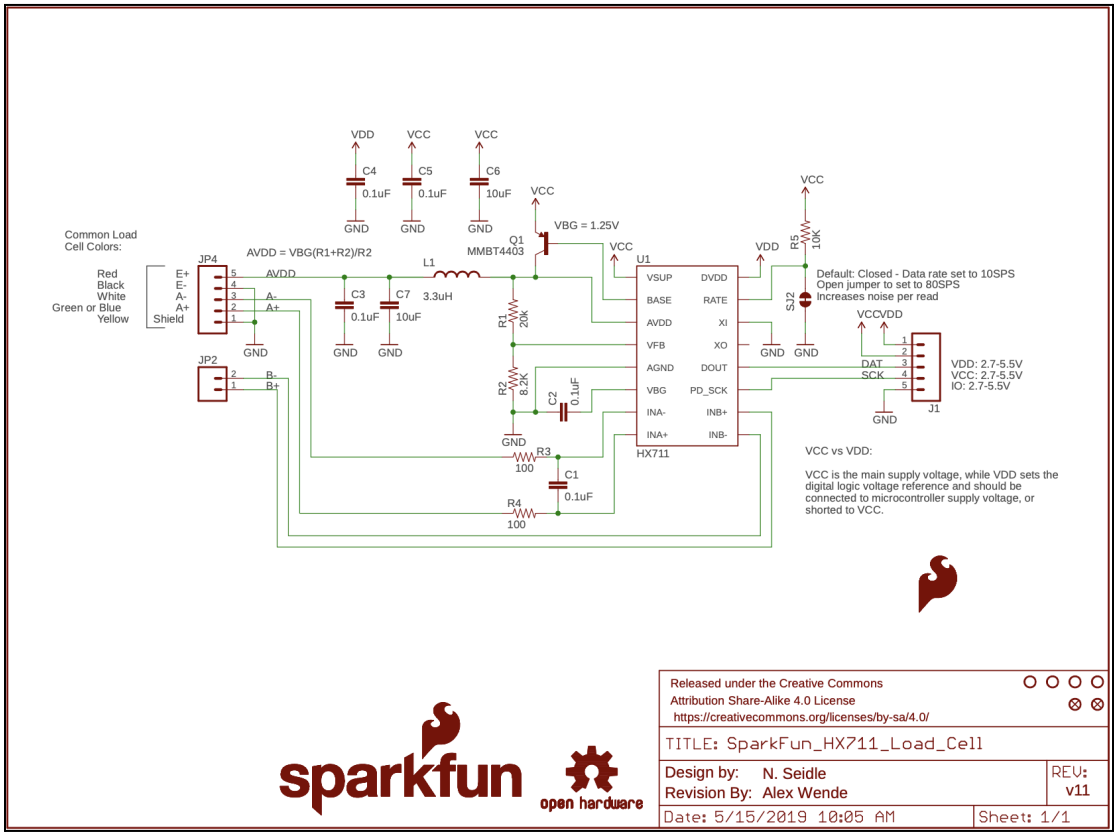


Figure 7.7. HX711 reference PCB board schematic [7.5]

The accompanying PCB board layout has slightly different naming conventions for each of the pins since it is taken from AVIA Semiconductors. Nonetheless, the design matches with the SparkFun HX711 schematic as S+ and S- match to A+ and A-.

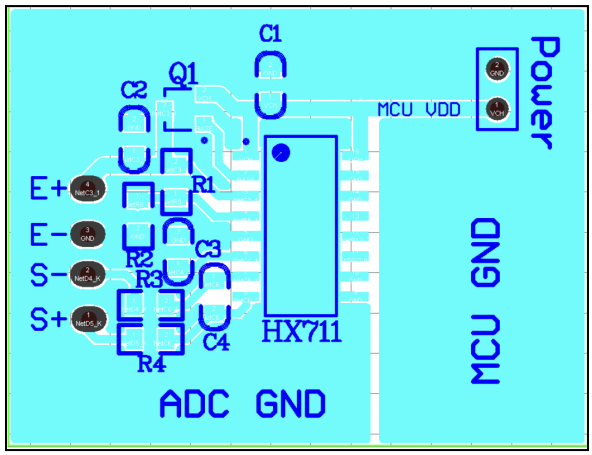


Figure 7.8. HX711 reference PCB board layout [7.6]

7.4.2 Digital Kitchen Scale

Figure 7.9. GGQ Digital Kitchen Scale

Any digital kitchen scale can be integrated with the HX711. The GGQ digital kitchen scale is selected for its easily accessible circuit compartment and its size. The circuit compartment can be quickly disassembled using a screwdriver, allowing for rapid integration with the HX711. With an 11 3/4 in. x 11 3/4 in. area, the scale should also be relatively large enough to hold a sizable quantity of individual vegetables and fruits.

7.4.3 Weight Scale System Assembly

In building the weight scale system, the initial objective is to solder the scale's data wires to the HX711's E+, E-, A+, and A- gates. The V_{CC} and GND gates are soldered to the Jetson Nano to provide power to the board. Finally, we need to wire the DATA and CLK gates to the Jetson Nano's auxiliary pins to enable data serialization.

Provided a successful linking process between the Jetson Nano and the HX711, the next step is to load the Jetson Nano with a driver program to control the HX711 and interpolate the received data. Driver programs are publicly available on the Internet, and we plan to opt for a Python-based implementation.

7.4 Hardware Architecture Bill of Materials

Below is the bill of materials—pre-tax—associated with assembling the RecipeCart's hardware architecture. The Rubbermaid White Dishpan is primarily selected for its size—a 12 in. x 14 in. area should adequately house the HealthWise bathroom scale. The jump wire and soldering kits are included as part of the list of materials and equipment needed for the hardware architecture, but they are freely available and do not incur any additional costs.

Part Name	**Source**	**Quantity**	**Price**
Rubbermaid White 11.4-Quart Dishpan	Amazon.com	1	$12.95
Jetson Nano Developer Kit	Amazon.com	1	$149.00
Wireless-AC8265 NIC WiFi and Bluetooth 4.2 Module	Amazon.com	1	$24.00
SanDisk 64GB MicroSDXC with Memory Card Adapter	Amazon.com	1	$11.70
5V/4A DC Jetson Nano Power Jack	Amazon.com	1	$17.00
Raspberry Pi Camera Module 2	Amazon.com	1	$25.00
HX711 Load Cell Amplifier	Amazon.com	6	$12.00
GGQ Digital Kitchen Scale	Amazon.com	1	$17.00
Jump Wire Kit	Home	1	$0.00
Basic Soldering Kit	Home	1	$0.00
	TOTAL:		$268.65

Table 7.1. Bill of materials for the hardware architecture

8.0 Software Design

As detailed in Section 2.7 and Chapter 4, the software design of the RecipeCart is characterized by two general processes: the machine learning architecture and the mobile app. This section examines the proposed machine learning architecture, details the procedure for setting up the cloud-based backend, and provides the visual groundwork for the mobile app frontend design.

8.1 Machine Learning Architecture

The machine learning featured in the RecipeCart falls into three categories: ingredient detection and classification, personalized recommendations, and custom recipe generation. Due to the cost of development and training machine learning models, it is optimal to maximize the utility of open-sourced, readily-modeled projects. Consequently, much of the design for machine learning components of the RecipeCart will tend to revolve around chaining these models together to form a larger pipeline.

8.1.1 Ingredient Detection and Classification Pipeline Design

Ingredients can be detected in passing data through one of two different pipelines: the barcode detection or the image classifier. In the barcode detection pipeline, the SBC actively searches for a barcode in its camera view. Upon the detection of a barcode, the script decodes the UPC barcode via Pyzbar and publishes it to the server where an AWS Lambda fetches the product information about the scanned ingredient from the World Open Food Facts database. This process should return the exact information about the scanned ingredient, including information about its quantity or weight. The product info is then parsed and loaded into a temporary ingredient object to be forwarded and processed by a separate AWS Lambda. This secondary Lambda function checks the ingredient for conformity and standardization by classifying it with its closest neighboring ingredient as stored in the Weaviate vector database.

The alternate approach to classifying an item is to take a picture of it directly from the phone's camera and send it to the server for classification. The current chosen implementation of the image ingredient classifier is Meta's DINOv2 for extracting and embedding the image of the ingredient which is then sent to Weaviate's vector database where the ingredient is classified based on the labels of its nearest neighbors. This method of reverse searching the image for classification scales well with new ingredient data assuming that the image embeddings reflect visual similarities between ingredients. To this end, we acquire an ingredient dataset with images to aid in the recognition of food ingredients. We thus obtain our grocery product sample dataset from Kaggle, which contains roughly 40 distinct grocery items with 20 pictures each [8.2].

8.1.2 Custom Recommendation System Design

When prompted by the user, a request is sent with the ingredients catalog packaged, recipes are recommended based on user preference, while accounting for user personalization. The choice of recommendation system is somewhat nuanced due to the reproducibility issue of recommendation system algorithms: different algorithms perform the best and worst on different datasets. As such, it is difficult to distinguish the effectiveness of the many variants of recommendation algorithms. One expensive way to find out which algorithm is the most efficient is to employ multiple of them and to observe which one performs best on the task. However, this test is beyond the scope of the project, and for the sake of demonstration we will employ a combination of content-based filtering and collaborative filtering to make recommendations to the users.

One notable weakness of collaborative filtering recommendation systems is that it has a cold start issue. As such, newer or niche content tend to be under-recommended. To alleviate this issue, RecipeCart's chosen recommendation system will also leverage some content-based filtering to form soft cliques of recipes which can help group similar recipes. Due to content-based filtering's inherent dependence on the choice of embedder, a neural network is needed to learn latent representations of different recipes. This design choice may prove useful for scoring AI generated recipes given user profiles.

After filtering the incoming data, some recommendation systems employ a scoring function to rank the content for strict constraints. For RecipeCart, this is where recipes that match users' current ingredient inventory and adhere to dietary constraints, are scored higher and non-compliant ones are filtered out. At this stage, other recommendation systems ensure content freshness by considering user history to further rerank the candidate content. Due to the complexity of implementing a novelty checker, we will not be reranking the recipes after the first stage of ranking, though it can always be implemented at a later time to increase the freshness of results.

8.1.3 Custom Recipe Generation System Design

RecipeGPT works well as a standalone recipe generator. Unfortunately, it suffers from incoherence and an inability to adhere to certain constraints. Inspired by RecipeMC, we can impose constraints by constructing an overarching planner over its architecture, whereby we can use a reward function based on recommendation embeddings to better predict user preferences and tailor recipes. This design works by allowing the generative model to search the different variations of recipe to be generated and select the most novel or promising one. To ensure that the tree generated by the search is not too large, RecipeMC employs a popular planning strategy in chess AIs, Monte Carlo Tree Search (MCTS).

The reward function for RecipeMC can include weak positive rewards to encourage generation of coherent recipes. This mechanic can be extended in RecipeCart by discouraging generation of recipes that stray from a user's dietary constraints with negative rewards, and encouraging generation of good food by giving positive rewards for high ratings and negative rewards for negative ratings. A potential issue with issuing

rewards based on user feedback may be that since the outcome of the recipe is not immediate. It may be difficult to store the weights of the reward function until users test the recipe and give feedback.

Alternatively, one way to ensure that generated recipes are performant is by comparing them to existing recipes that are filtered for the user. Instead of evaluating the generated recipe as is, we can evaluate it by comparing it to the performance of similar recipes by leveraging the content-based embeddings of the generated recipes. By constructing a reward function that estimates the value of a token based on the monte carlo tree search rollouts, we can induce user personalized recommendation based on the given input without retraining the generation model.

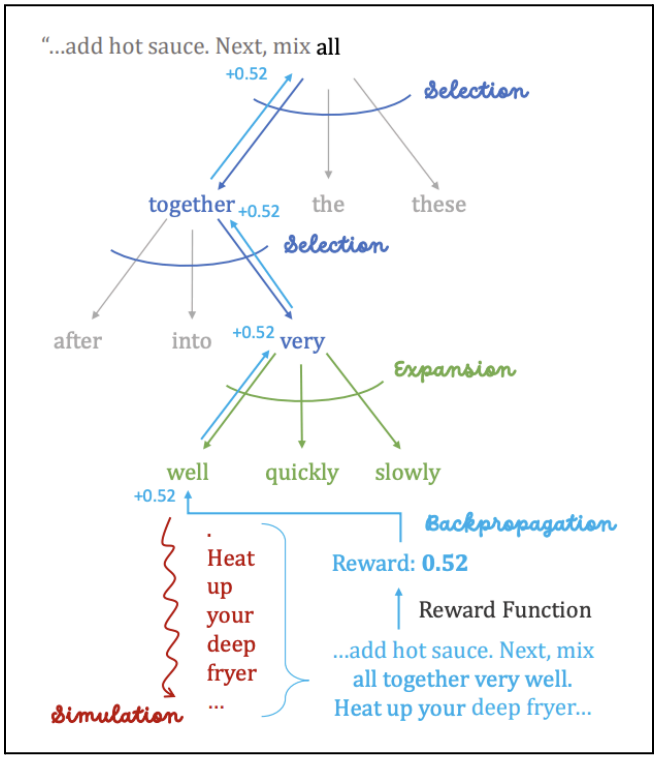


Figure 8.1. RecipeMC [8.1]

Figure 8.1 shows the architecture for RecipeMC. At the top, each token is selected by sampling from RecipeGPT, which will produce different possible tokens at each step. RecipeMC extrapolates RecipeGPT by performing a tree search on each token to select the token with the highest predicted reward. In MCTS, at each node, different promising choices are sampled and evaluated by rollout simulations, the most novel or promising tokens are expanded on and rewards are propagated back through the path it is discovered. The reward function can be designed to increase the coherence of the generated text and impose additional constraints to improve personalization. [8.1]

8.2 Backend Design and Databases

The AWS Amplify server maintains three natively configured services: an authentication system, a GraphQL-based API, and a set of Lambda functions.

The authentication system is managed by Amazon Cognito, which stores and provides the user credentials to the Flutter-based frontend. Credentials are hashed and access tokens are managed in Cognito, alleviating the user login and sign-up logistics. Any access to internal AWS resources is federated by Cognito, ensuring authorization and accountability with each API request and function call. Implementing Cognito also facilitates future integration with third-party social identity providers such as Google, Facebook, or Outlook. The authentication layer is added via the *amplify add auth* command.

A GraphQL-based API ensures data integrity when querying or mutating data stored in Amazon DynamoDB. The GraphQL API itself is managed and executed through AWS AppSync, which supports a subset of basic GraphQL commands and access patterns. The app's access patterns define the GraphQL schema. Particularly, we are most interested in retrieving recipes given a set of input ingredients. This custom query must be directly handled via a Lambda resolver since AppSync does not natively support such a reversed filter search. The API layer is added via the *amplify add api* menu.

AWS Lambda functions are integrated into the Amplify backend to handle the machine learning requests and responses. We employ three main Lambdas: one for submitting a photo of an ingredient to the object classification pipeline, one for controlling the barcode detection script on the Jetson Nano, and one for parsing the barcode and processing the product metadata. The functions are added and configured via the *amplify add function* prompt menu.

8.2.1 AWS Amplify Setup

The first step in setting up AWS Amplify is to install the Amplify CLI on the developer's IDE to enable rapid resource deployment via AWS CloudFormation templates. The Amplify CLI also provides a streamlined automated process to integrate Flutter applications with AWS backend resources.

In configuring the Amplify CLI, we create a federated IAM user through the AWS console for the Amplify CLI to assume when executing cloud-based commands. The user is given administrator access to Amplify for simplicity, but in hindsight, the permissions should follow the principles of least privilege to minimize security vulnerabilities. We then run *amplify init* to instantiate the backend resources and the appropriate configuration and header files in our IDE. We deploy Amplify as a backend platform for the RecipeCart such that the mobile frontend is manually developed and maintained by us developers. We also add the necessary dependencies to the *pubspec.yaml* file to enable AWS Amplify on Flutter.

```
dependencies:
  amplify_api: ^1.0.0
  amplify_auth_cognito: ^1.0.0
  amplify_authenticator: ^1.0.0
  amplify_flutter: ^1.0.0
  amplify_storage_s3: ^1.0.0
  cached_network_image: ^3.2.3
  flutter:
    sdk: flutter
  flutter_riverpod: ^2.1.3
  go_router: ^7.0.0
  image_picker: ^0.8.0
  intl: ^0.18.0
  path: ^1.8.3
  riverpod_annotation: ^2.0.1
  uuid: ^3.0.7
  provider: ^6.1.2
  amplify_datastore: ^1.7.0
  aws_iot_api: ^2.0.0


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.5
  camera: ^0.10.5+9
  path_provider: ^2.1.2
  mqtt5_client: ^4.2.2
  aws_lambda_api: ^2.0.0
  ndialog: ^4.3.1
  http: ^0.13.6
  flutter_dotenv: ^5.1.0
  flutter_rating_bar: ^4.0.1
  collection: ^1.18.0

```

Figure 8.2. AWS Amplify Flutter Dependencies

8.2.2 AWS IoT Core Setup on Jetson Nano

AWS IoT Core must be installed on the Jetson Nano to enable bi-directional communication between the Jetson Nano and the AWS Amplify backend while remaining within the AWS internal network. Configuring the AWS IoT Core agent on the Jetson Nano is preferred because communicating within the AWS internal network is significantly faster and more secure than simply exchanging data through HTTP requests.

We instantiate an IoT Thing to represent the Jetson Nano in the AWS cloud namespace. We then download the necessary certificates and keys to the Jetson Nano, allowing the device to properly authenticate itself in AWS. We then run the default configuration kit on the Jetson Nano which installs the AWS CLI, establishes a connection with the AWS platform, and registers our device as a Thing.

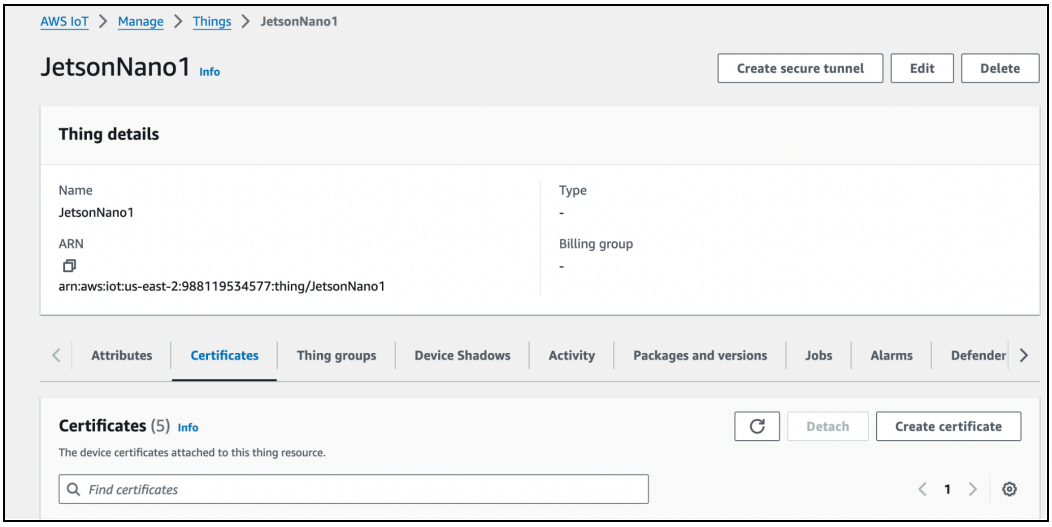


Figure 8.3. AWS Amplify Authentication interface

8.2.3 Entity Relationship Diagrams

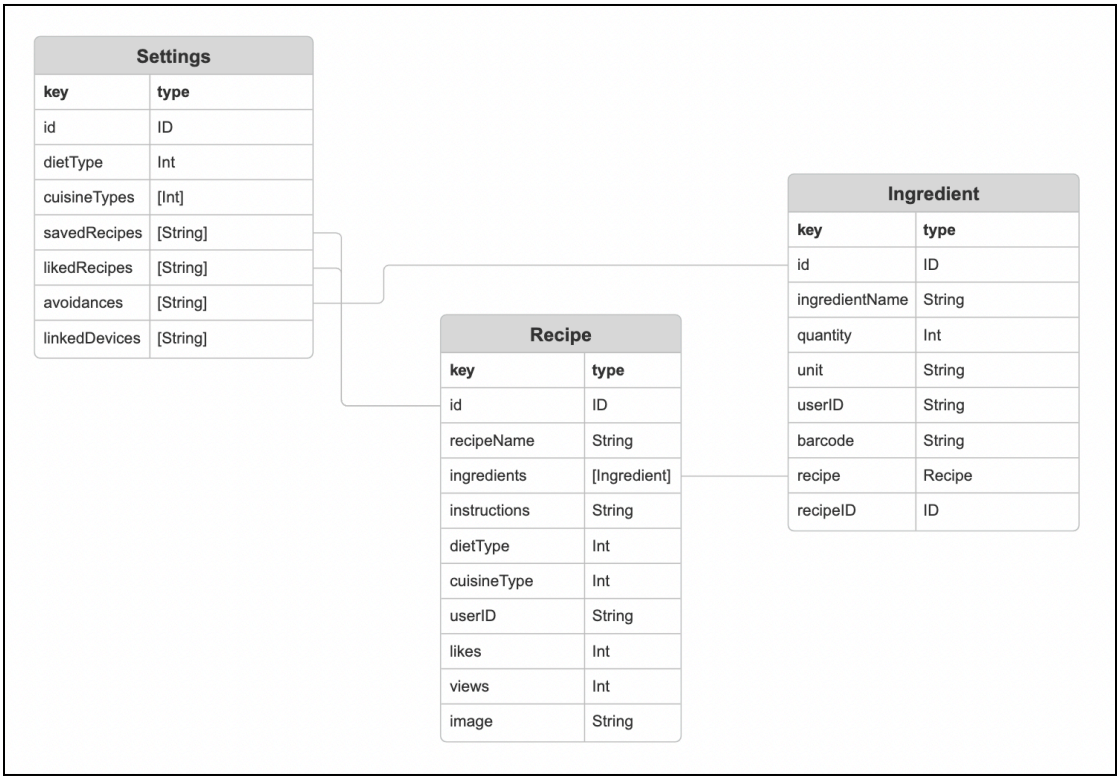


Figure 8.4. User-Recipe-Ingredient Entity Relationship Diagram

The RecipeCart requires three DynamoDB tables: an ingredient table, a recipe table, and a user settings table. Figure 8.4 shows an entity relationship diagram between user settings, recipes, and ingredients.

The ingredient table is the crux of the RecipeCart app as it maintains connections with the Recipe and User Settings tables. The ingredient table stores both the user-owned ingredients and the ingredients required for each recipe. We distinguish the former by the presence of a userID—provided by Amazon Cognito Identity Pools—a barcode entry, and a removed status boolean, while the latter is denoted by the presence of a recipeID and recipeName. Additionally, the required quantity field for each ingredient reflects either the quantity currently owned by a user or the quantity needed in a parent recipe, respectively.

```
50
51  type Ingredient @model @auth(rules: [
52    {allow: private, operations: [read]},
53    {allow: owner, ownerField: "userID"},
54    {allow: public, provider: apiKey}
55    ]) {
56    id: ID! @primaryKey
57    # ingredient name as obtained from WOFF
58    ingredientName: String! @index(name: "byIngredientName", queryField: "ingredientByName")
59    # owner of ingredient (i.e. user)
60    userID: String  @index(name: "byUserID", queryField: "ingredientByUserID")
61      @auth(rules: [{allow: owner, operations: [read]}, {allow: private, operations: [read]}])
62    barcode: String @index(name: "byBarcode", queryField: "ingredientByBarcode")
63
64    # field to be populated by Weaviate ingredient database
65    # list of related names stored in Weaviate. Used to search for recipes containing similar ingredient names
66    relatedNames: [String!]!
67
68    # boolean to check if removed from user inventory or not. Default to false upon creation
69    removed: Boolean!
70
71    # user's ingredient quantity
72    quantity: Float
73    unit: String
74
75    # user's ingredient quantity in grams
76    standardQuantity: Float
77
78    # recipe: [Recipe] @manyToMany(relationName: "recipeIngredient")
79    # recipe: Recipe @belongsTo(fields: ["recipeID"])
80    # recipeID: ID @index(name: "byRecipe") # customized foreign key for parent primary key
81  }
82
```

Figure 8.5. Ingredient GraphQL Model

Each recipe in the recipe table has a name, a diet type, a cuisine type, a text block of instructions, and a list of ingredients. User-created recipes contain a userID entry. Each recipe also contains several fields to aid with implementing the recipe recommendation system such as the number of likes, the average rating, and the number of views.

The user settings table contains all the user metadata not managed by Amazon Cognito—i.e. the non-credential data. The user's preferred diet type, cuisines, avoidances, and their saved and liked recipes. The avoidances field map to a list of ingredientName corresponding to ingredients that the user would like to avoid for

personal or allergic reasons. The savedRecipes and likedRecipes fields contain a list of recipeID.

Figure 8.6 shows an entity relationship diagram between the user settings and its components. A user setting contains a private key for identifying it. Since there is a unique mapping between users (not shown in the above diagram) and user settings, they are often paired together (a one-to-one only relationship).

```
83
84  type Settings @model @auth(rules: [
85    {allow: owner, operations: [read, update, delete]},
86    {allow: public, provider: apiKey}
87    ]) {
88    id: ID!
89    email: String!
90    owner: String! @index(name: "ByOwner", queryField: "settingsByOwner")
91    dietType: Int
92    savedRecipes: [String] # recipeID from Weaviate recipe database
93    ratedRecipes: [String] #recipeID from Weaviate recipe database
94    avoidances: [String] # ingredientName
95    language: Int
96    notifications: Boolean
97    linkedDevices: [String]
98  }
```

Figure 8.6. User Settings GraphQL Model

As depicted in the figure, there are multiple diet types, ingredient types, cuisine types and devices. The user setting stores the user choice of diet or lack thereof (a one-to-zero/one relationship). This means that users can only select one type of diet at a time, so as to avoid allowing users to select contradictory diets (meat only and vegan).

Settings also store a set of ingredient types that are avoided. This is a one-to-zero/many relationship, as users may opt to not avoid any ingredients or avoid many ingredients. Similarly, users can select preferred cuisine types, which would also characterize a one-to-zero/many relationship. Users may select a language (a one-to-one relationship). Users may connect devices to their account; this is a one-to-zero/many relationship as users can use the app without a smart device connected or integrate the app with a smart device.

The recipes are immutable and can only be viewed and searched. The only mutable field is the ratings of the recipe. As such, to facilitate our recipe search and recommendation algorithms, we store the recipes inside the Weaviate vector database as text-to-vector embeddings, allowing for lexicographic comparisons, filtering, and sorting between recipes.

8.2.4 Backend Pipelines

The barcode detection algorithm is loaded onto the Jetson Nano to simulate integration with a smart fridge, where the barcode detection is performed externally and subsequently forwarded to the RecipeCart backend server. The entire process is initiated from the frontend via a Lambda function. The Lambda function sends a shell script command—containing the current user's Cognito Identity ID as a parameter—to the AWS Systems Manager Agent installed on the Jetson Nano to start or halt the barcode detection Python program. Upon decoding the detected barcodes, the Python script packages the JSON result into a MQTT-based message to be published to a topic in AWS IoTCore—the topic name is determined by the Cognito ID of the user who initiated the barcode detection process. A Lambda function subscribes to the topic and consumes the MQTT messages. The Lambda function processes the barcode, checking if the ingredient product already exists in the DynamoDB ingredient table. If the product is not found, the Lambda must query and retrieve information from the external World Open Food Facts database. Otherwise, the ingredient is sent back to the frontend for display and user confirmation with the associated quantity and userID. The user may then choose to add the ingredient to their inventory or remove and rescan it.

In addition to the barcode-processing Lambda, we configure an Amazon Data Firehose sink that captures all the incoming MQTT messages in AWS IoTCore and dumps them into an S3 bucket for record-keeping and management.

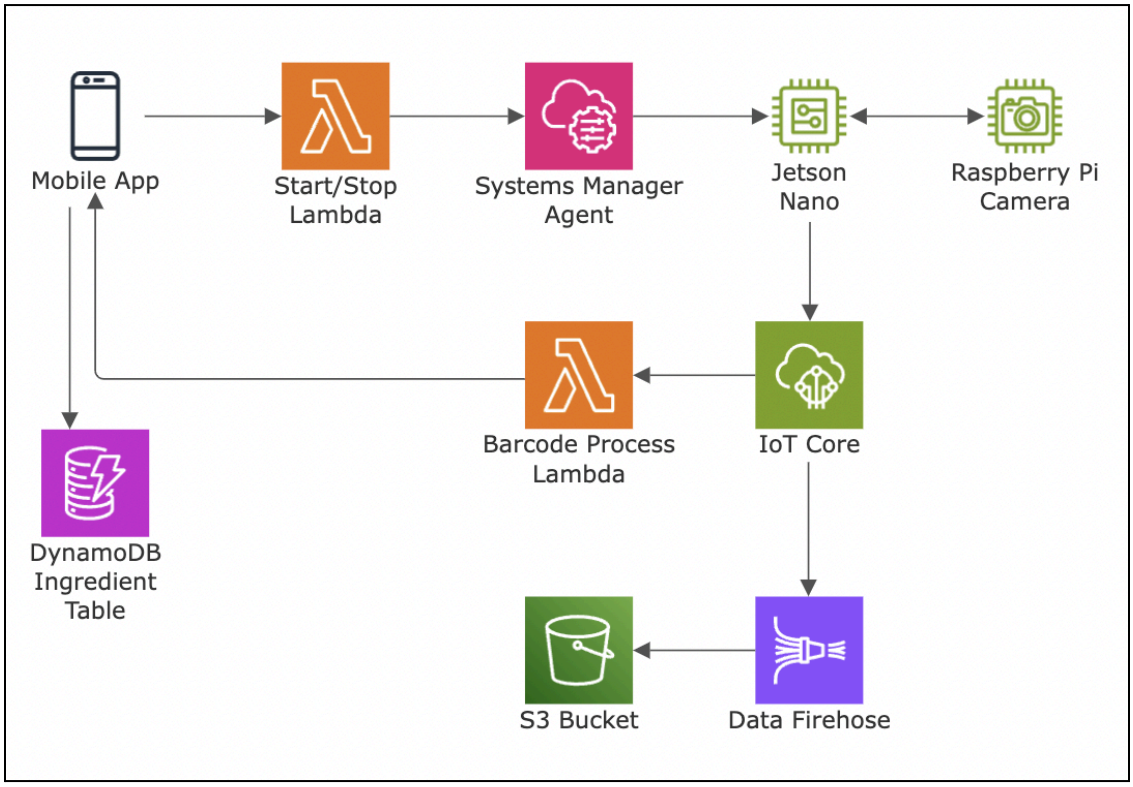


Figure 8.7. Barcode Detection Pipeline Diagram

The alternate approach to adding ingredients to the user inventory can be initiated via the mobile camera feature in the RecipeCart app. The picture is put into an S3 bucket which triggers a Lambda function to process the event.

The image pre-processing code, the Meta Dinov2 model, and the Weaviate client are packaged as a Docker container image, which itself is uploaded to Amazon Elastic Container Registry (ECR). From there, the Lambda function exposes a facet of the container image. Containerizing this pipeline not only allows for portability and resilience but is also necessary to allow the Lambda function to run the required PyTorch machine learning library; the PyTorch library is too large to be included into the Lambda function as a Lambda layer dependency.

Within the container, the input image is resized, cropped, standardized, and converted into tensor before being fed into the PyTorch-based Meta Dinov2 model. The Dinov2 model turns the image tensor into a one-dimensional tensor, which is then flattened to produce a vector embedding of the image.

The Weaviate client initiates a connection with the Weaviate cluster hosted on EC2 and queries it. The Weaviate host effectively performs an approximate K-Nearest-Neighbors (KNN) classification and returns a list of vector embeddings similar to the input image and their associated label. The labeled vector with the smallest KNN distance is returned to the mobile frontend as the classification result. The user is then prompted to confirm the identified ingredi- ent and enter the associated quantity. Upon confirmation, the ingredient is put into the DynamoDB ingredient table via the GraphQL API.

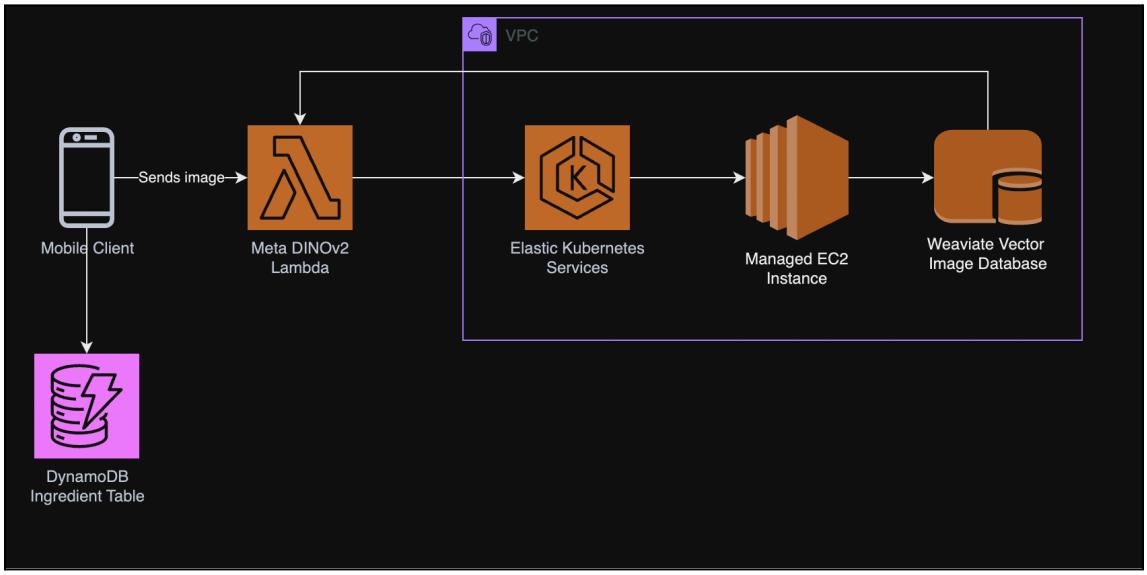


Figure 8.8. Image-based Classification Pipeline Diagram

8.3 Mobile Frontend Design

The frontend design for the mobile app is characterized by its primary functionalities: to display a AI-driven recipe generation system for the user, a list of ingredients currently in the user's inventory, a list of the user's saved recipes, a user settings menu, and an "Explore" functionality for looking up ingredients and recipes. All mobile pages are designed through Figma such that AWS Amplify or any similar AI-powered software development platform can easily interpret the components and facilitate later implementations of the frontend through Figma-to-Code generation.

8.3.1 Login System

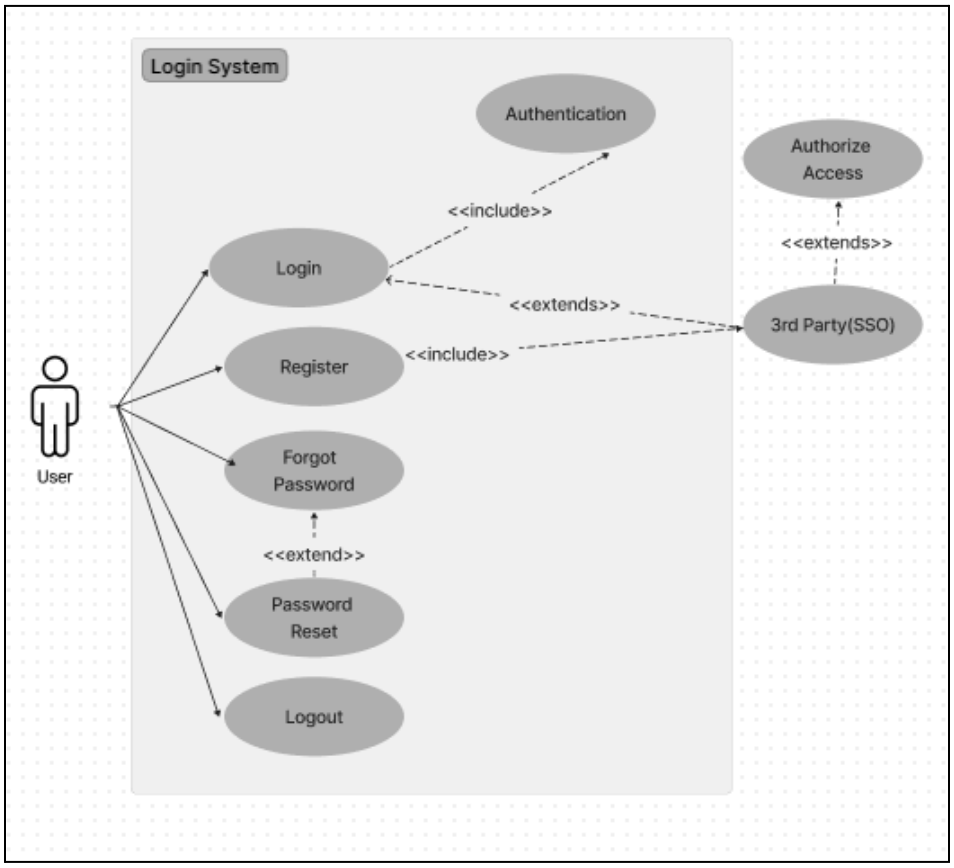


Figure 8.6. Login system use case diagram

Figure 8.6 describes the user's interaction with RecipeCart's login system. The main use case is login, where the user logs into the system by providing their username or a third-party provider such as Google or facebook. The user can also have the options to recover the account through third-party authentication and reset their password. Lastly, the logout option releases all connections regarding the user connection and third-party sources.

8.3.1.1 Login and Create Account Pages

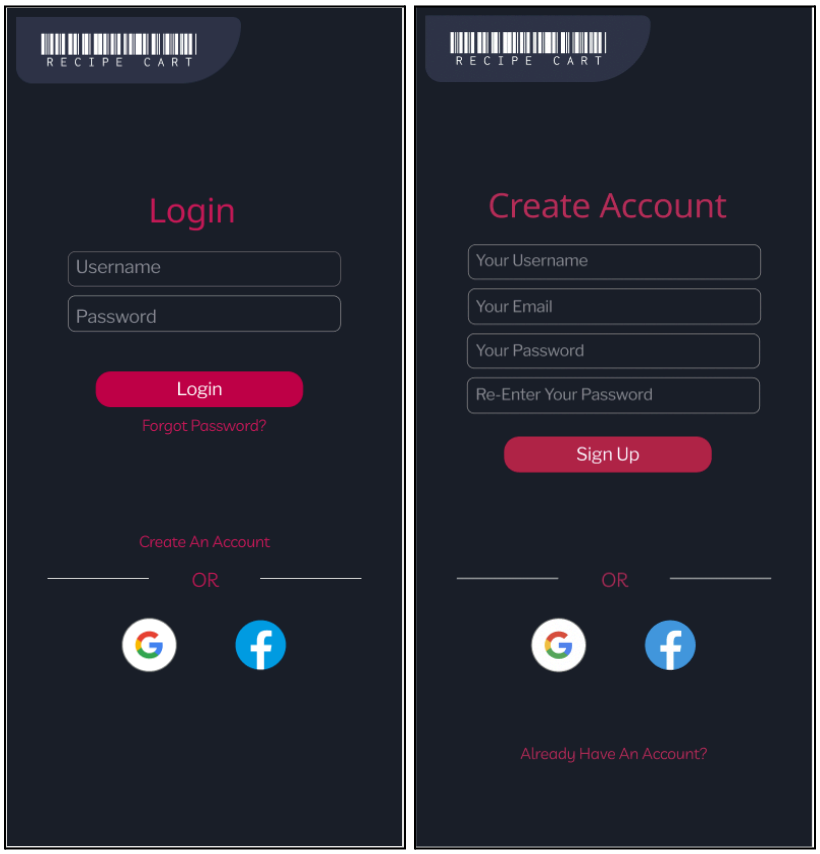


Figure 8.7. Login and Sign Up Figmas

The login page is the default landing page upon opening the app for the first time. The page involves three main user interactions: a login button, a "Forgot Password?" hyperlink, and two user registration options. The login button simply handles account verification: if the username exists in the user database and the password corresponds appropriately, then the app redirects the user to the home page (See Section 8.3.3). The "Forgot Password?" link allows the user to enter the account recovery page shown in the following Section 8.3.1.2.

The app has two user registration options. The "Create An Account" button directs the user to the sign up page shown in Section 8.3.2. The user may also sign up via a third-party token authentication platform such as Google or Facebook.

The sign up page is redirected from the login page. It would collect users' sign in information to create a new account when they fill out all the fields and click on the sign up button. Users can alternatively opt to sign in or sign up based on the existing Google or Facebook account. Users are redirected to the login page upon pressing the "Already Have An Account?" hyperlink.

8.3.1.2 Account Recovery and Password Reset Pages

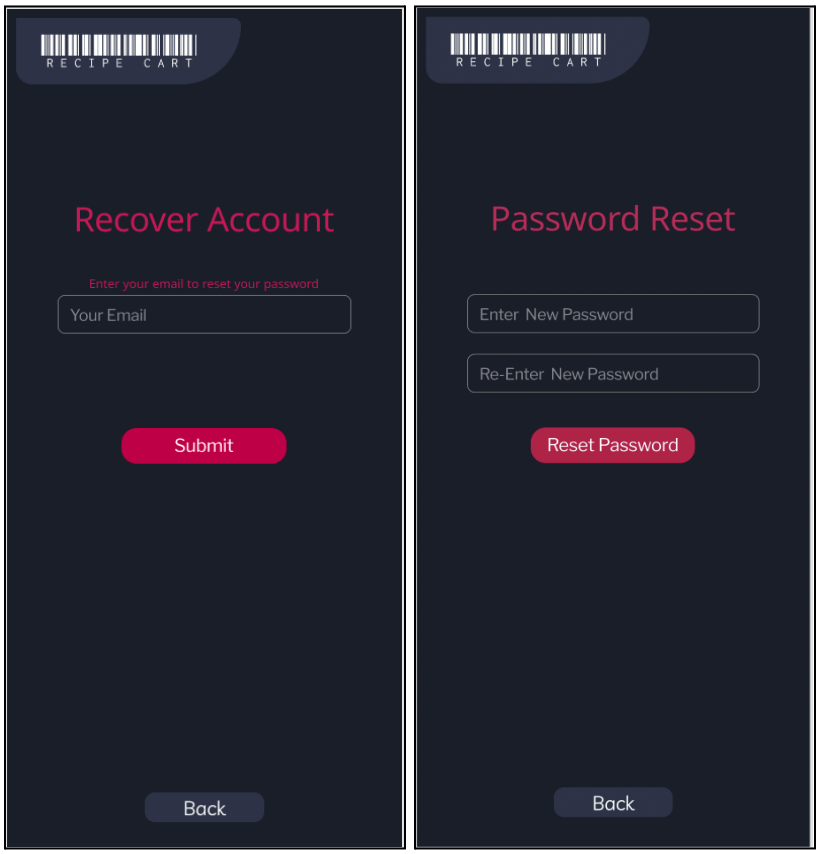


Figure 8.8. Account Recovery Figmas

The account recovery page is only accessible through the login page. Its main function is to allow the user to recover a lost password. Users should enter their account email and press the “Reset Password” button, which would prompt the server to validate the existence of the email in the user database and email the reset link accordingly. The email should relocate users to a recovery page where all the account fetch functions would happen. After following instructions on the recovery site, users may attempt to login again.

8.3.2 Recipe System

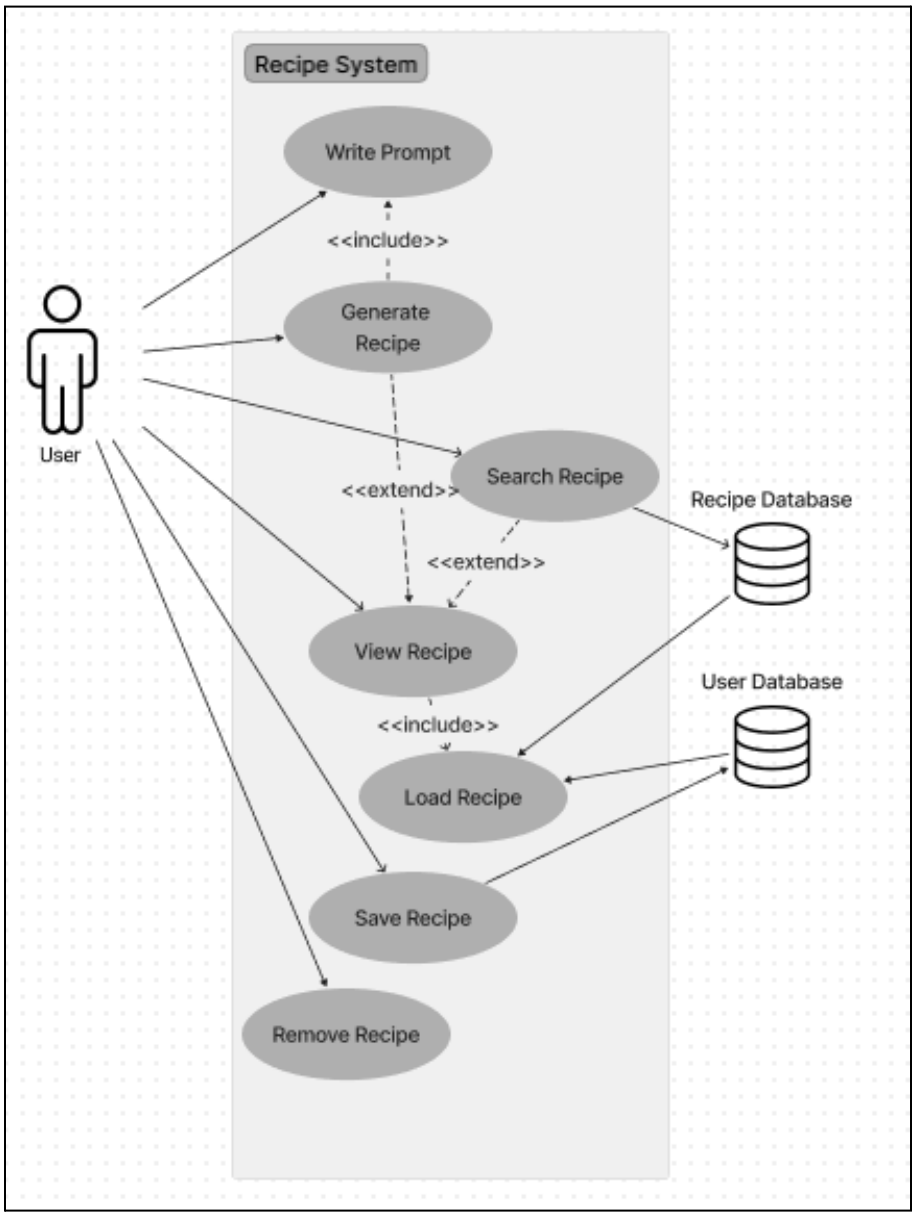


Figure 8.9. Recipe system use case diagram

Figure 8.9 illustrates all user's privileges when it comes to the Recipe System. This encompasses the entirety of functionalities and access rights the user has towards accessing and managing the recipes database through searching using keywords to filter out recipes, viewing and loading the recipe. In addition, the user can generate their own recipes by providing a prompt as a means to provide context for creating unique recipes with specific features catered to their needs. The user can then save the generated or searched recipe as part of the user database, which they can conveniently view or remove from the saved list.

8.3.2.1 Explore Recipes Page

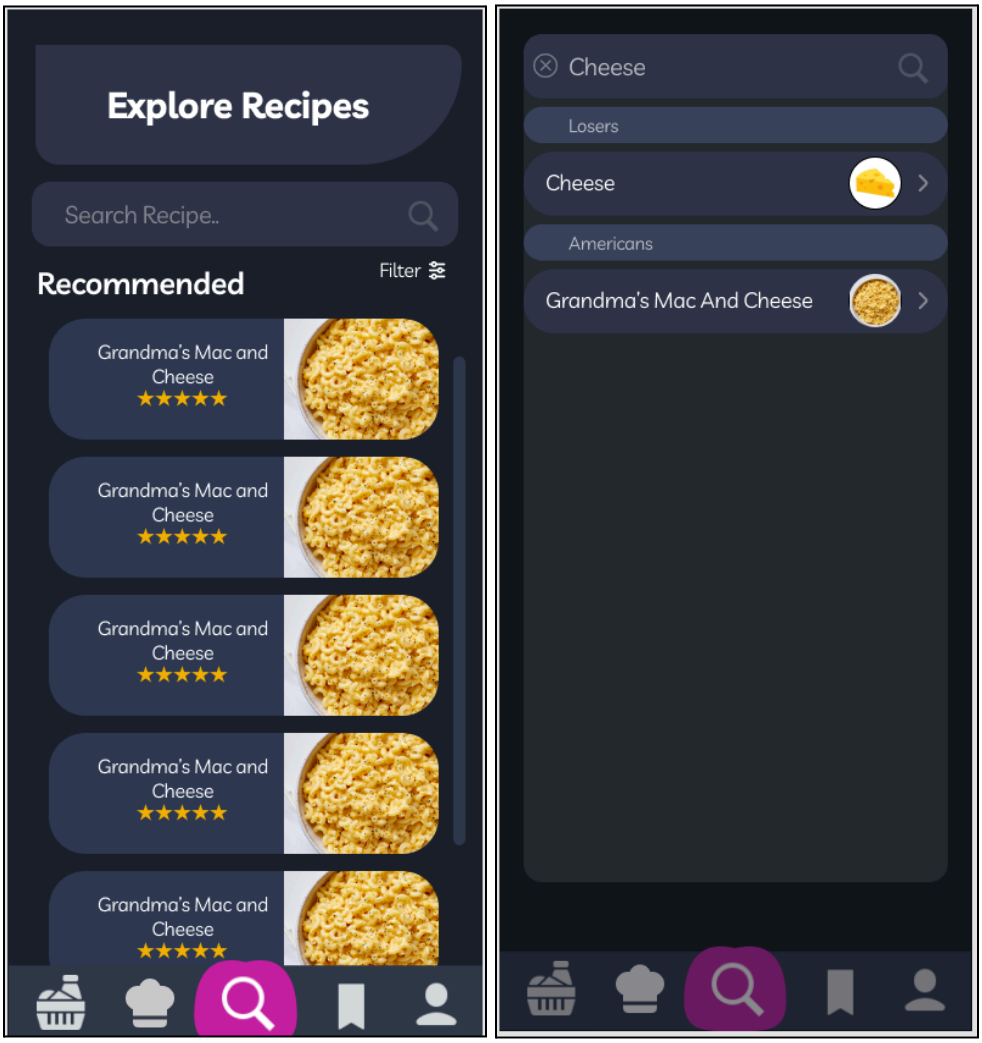


Figure 8.10. Explore Recipes landing page and search dropdown Figmas

Figure 8.10 features the main explore page, where users see all of their recommended recipes or search for new recipes. The visual layout for the explorer is explained in the general layout shown, where the user is recommended a list of recipes catered to their food preferences, as well as behaviors throughout their process of interacting with the app. These behaviors can be what they search up most often or the type of recipes they click on and save.

Users can conveniently filter out popular keywords that will be listed under the filter button and the layout will be tweaked so that all recipes shown will be within a specific diet or avoid selected avoidances. The user can also search for specific ingredients or dishes in the search bar, where they will receive a list of categorized results by dish/ingredient or type of cuisine that fits them the most.

8.3.2.2 Recipe Generation and Generated Recipe Pages

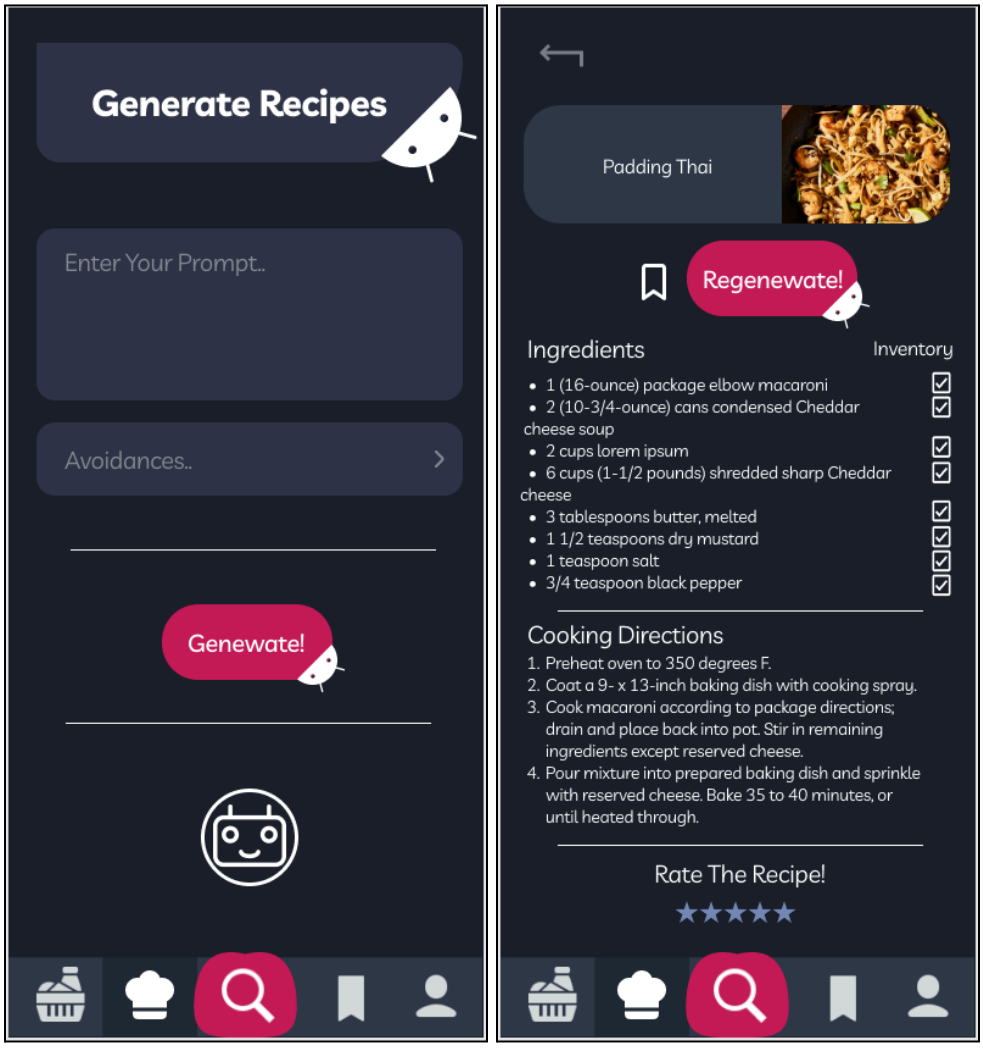


Figure 8.11. Recipe Generation Figmas

Figure 8.11 displays the Generate Recipes page, where the user can enter their prompt and avoidances, then the system can take their inputs and existing user preferences information to dish out a unique recipe. These prompts can be a list of ingredients, a set of commands regarding cooking time or experience, or a combination of the two that provides RecipeCart's ai context to consider in order to provide recipes that the user wants.

The avoidances option expands into a checklist similar to the avoidances search bar shown in Figure 8.23. This pattern similarity is necessary to skip the hassle of going into the preference settings or regenerating the recipe in case the user wants to temporarily filter out specific ingredients that they don't want in the generated recipe whether they expired or that the user doesn't feel like using them at the moment.

8.3.2.3 Saved Recipes and Selected Recipe Pages

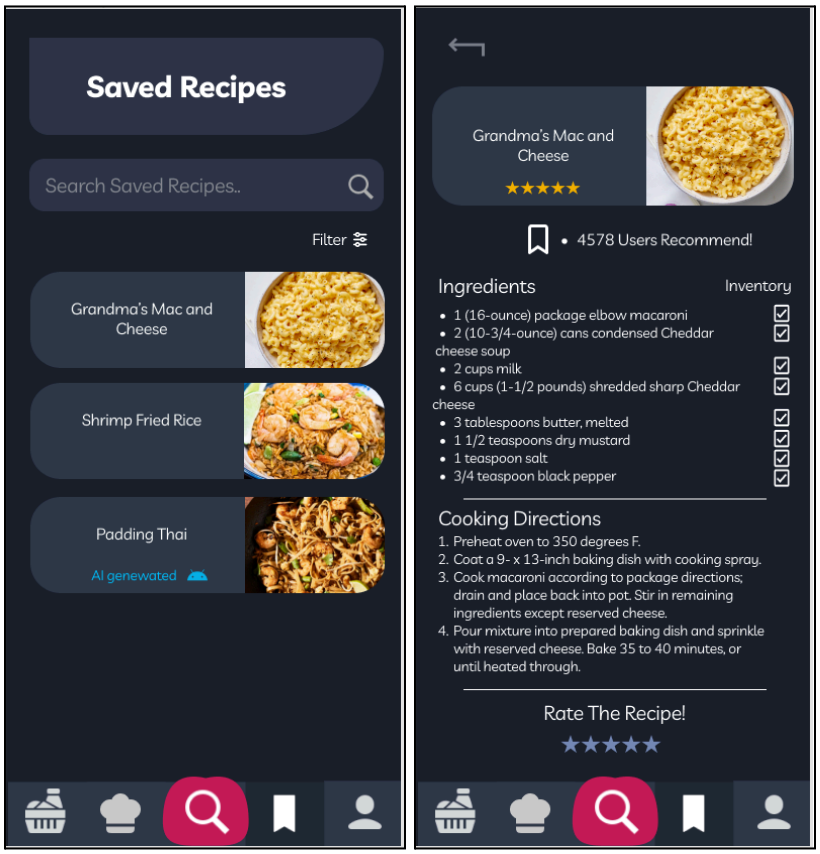


Figure 8.12. Saved Recipes and Selected Recipe Figmas

Figure 8.12's left panel displays the Saved Recipes page, which shows a list of saved recipes in which users can browse and click on to view. At the top, users can search for a specific recipe or filter out their saved list by ai-generated/existing recipes, or other famous keywords that they can filter out to look for the recipes they want. Each shown entry can be clicked on which will redirect the app to another screen with the recipe details for each recipe.

The right side of the figure displays the sample detailed layout of a selected recipe. This page lists out basic information for the recipe such as required ingredients and cooking directions. It also takes in manual data such as rating the recipe in order to display the value of it. Such ratings are also important forms of user feedback, and will affect the recommendation system depending on how many votes there are—the sample displays around 4500 votes which is a fairly sufficient number and will be recommended for many users that enjoy cheesy dishes. The page also syncs with the inventory, displaying which ingredient in the inventory that matches with the required ingredients in the list. If the user contains the ingredient in their inventory, it displays in the recipe page as a check mark for each item. The reverse arrow on the top left navigates the user back to the main explore page for when they finish cooking or they are no longer interested in the recipe.

8.3.3 Ingredient Inventory Tracking System

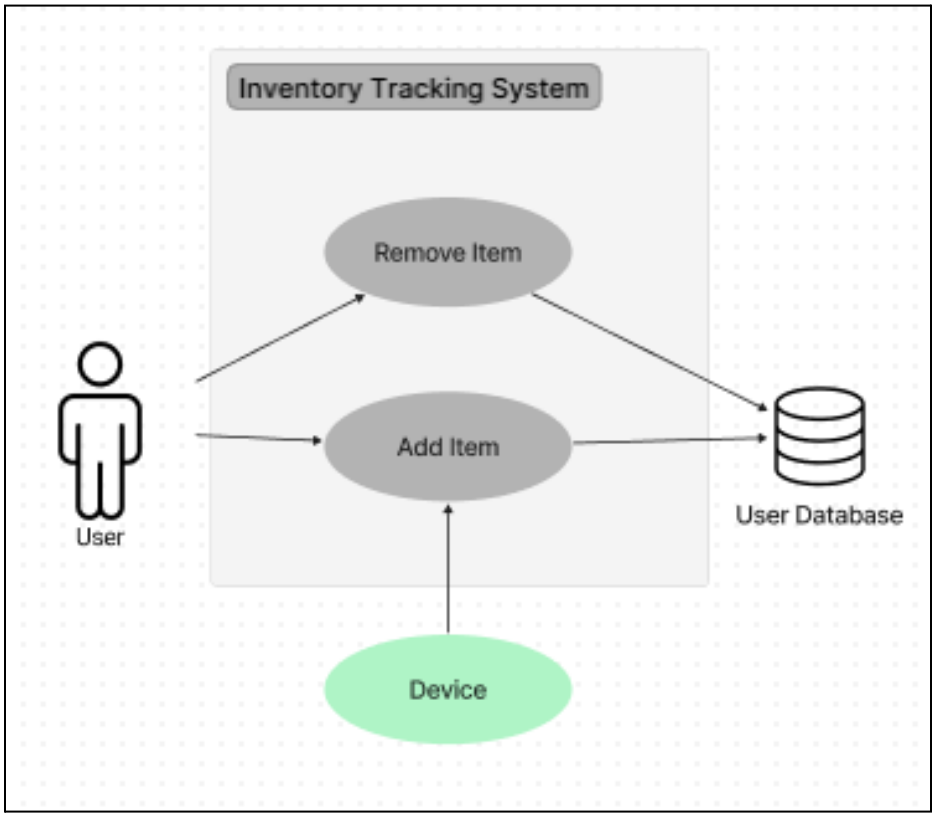


Figure 8.13. Ingredient Inventory Tracking system use case diagram

Figure 8.13 visualizes the user's access and privileges to the inventory system. When it comes to inventory tracking, the system will do most of the work, where it will track the inbound and outbound flow of ingredients within all smart inventory devices that the user connected, then save them under user database where it will be used as information for the recipe search and generation system (see Figure 8.9).

Another feature the application provides to the user is the ability to manually remove or add items into the inventory. Sometimes, there are certain food items that the user doesn't put into their fridges (onions, potatoes, …) but would like them to be considered in the inventory system to track their food or use for the recipe system. This also helps with situations where the inventory device fails to detect the incoming or outgoing ingredients due to specific features—the watermelon is cut out and yellow when the inventory is trained to specify the meat color to be red—and requires a manual fix.

The inventory tracking system provides a convenient way to track the user's inventory, while also giving them the freedom to access and edit their own storage data to satisfy personal needs.

8.3.3.1 Ingredient Inventory Page and Search Inventory Dropdown

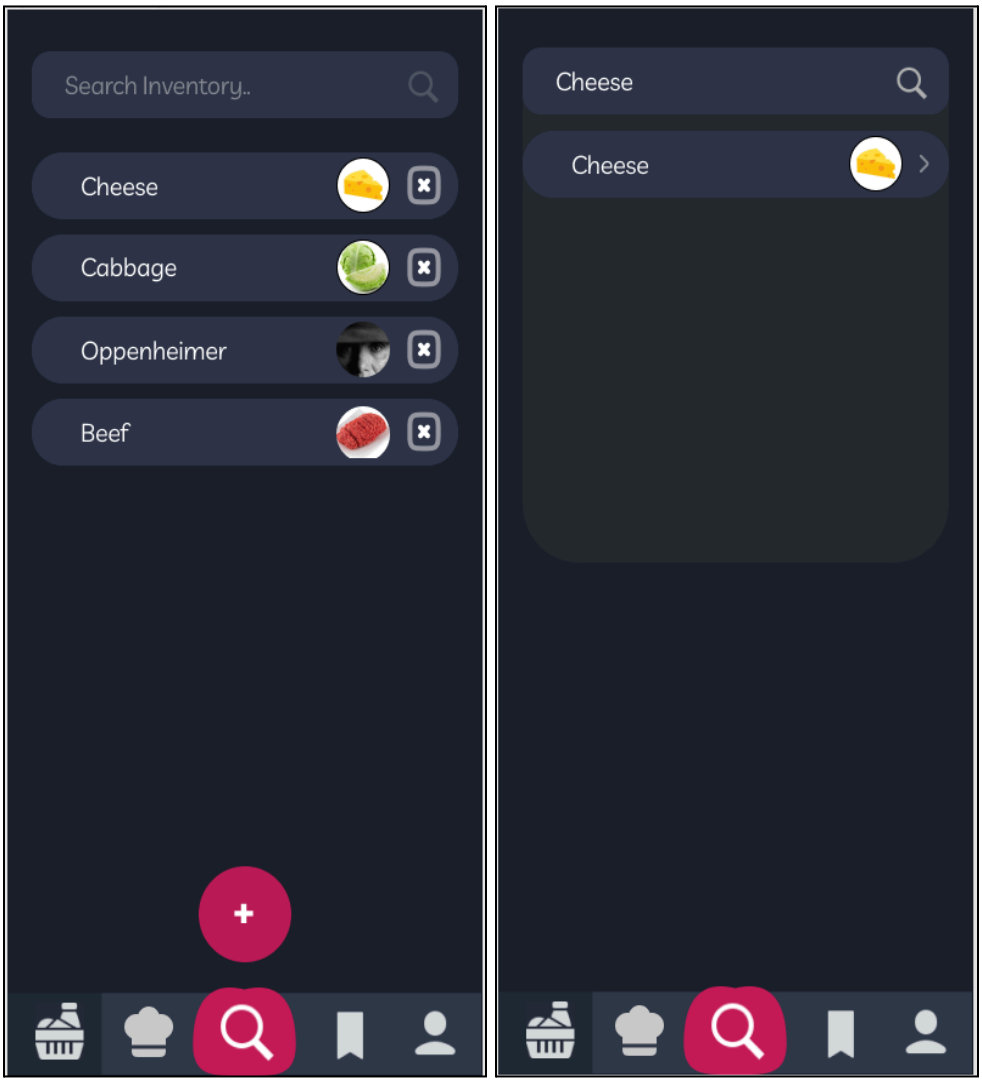


Figure 8.14. Ingredient Inventory Page and Search Dropdown

The left figure shows the ingredients inventory page. The default layout will show all ingredients that the inventory device—provided by the user—picks up during the initial setup stage. At the top, users can query their current ingredient inventory and check if the ingredients are in the user's inventory. The x button in each ingredient allows the user to manage their inventory and remove unwanted items, in case there is an error, or unwanted ingredient sitting in the cart.

The right figure shows how the user can look up within their inventory to see a filtered inventory based on typed keywords. The user then can choose to remove them straight from the search bar, or view details of the ingredient by tapping on it. In order to escape the search bar, the user simply taps out of the area to navigate back to the default inventory layout shown on the left figure.

8.3.3.2 Add Ingredient to Inventory and Add Quantity Modals

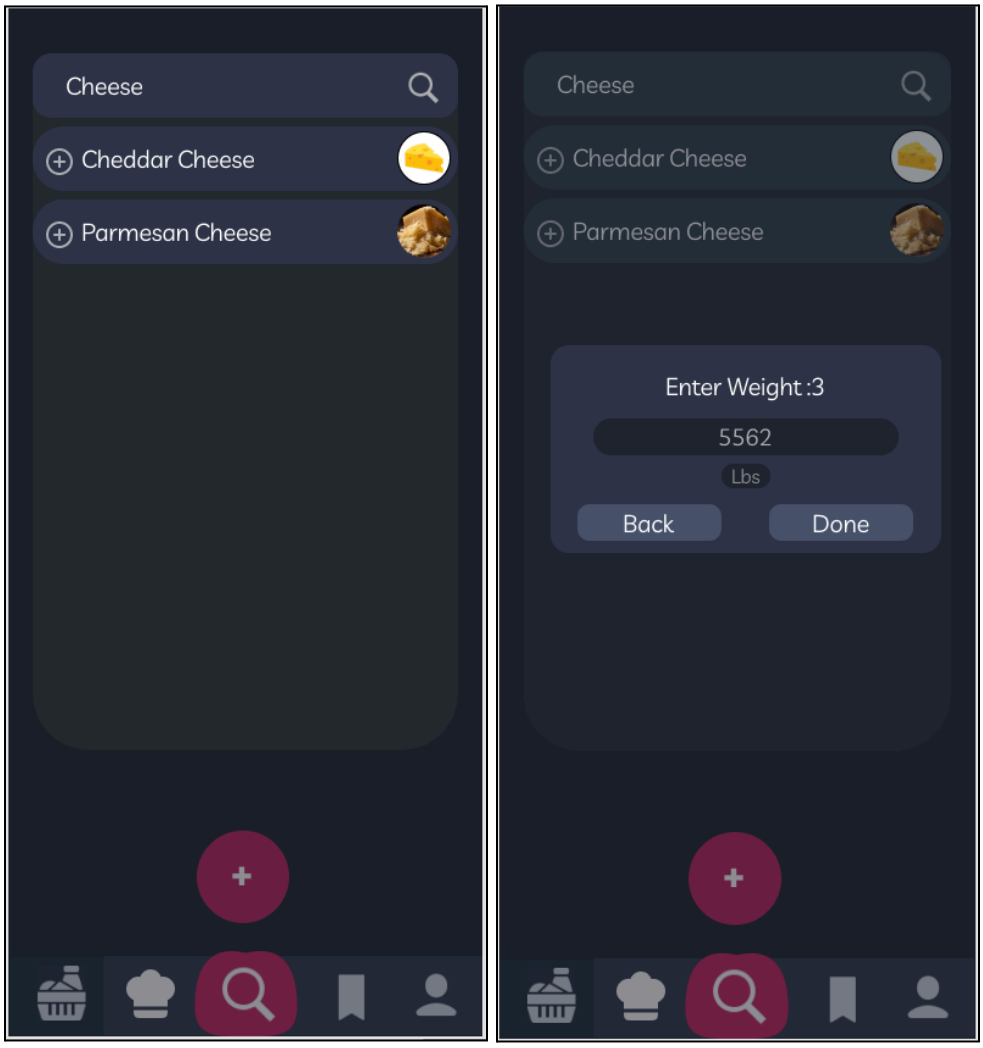


Figure 8.15. Add Ingredient to Inventory Figmas

The left figure shows ingredients being searched to be manually added to the user inventory. This feature is accessed by tapping on the "+" button in the inventory layout (Figure 8.14, left panel). This feature exists to account for items that are too difficult to be detected by RecipeCart due to lacking features in the food detection algorithm, or for ingredients not in the direct presence of the user. This search is similar to the ingredient avoidances search described in Figure 8.23.

The right figure shows how the user can then enter the weight of the ingredient they chose in order to have better clarity towards their inventory. In assumption of the item always existing in their house, the user can leave the window blank and the system will assume that the ingredient is always available. After inputting the food weight, the user presses done to save and add the item to their inventory listing, or choose to press back or any area outside the popup window to navigate back to the search panel.

8.3.4 Settings System

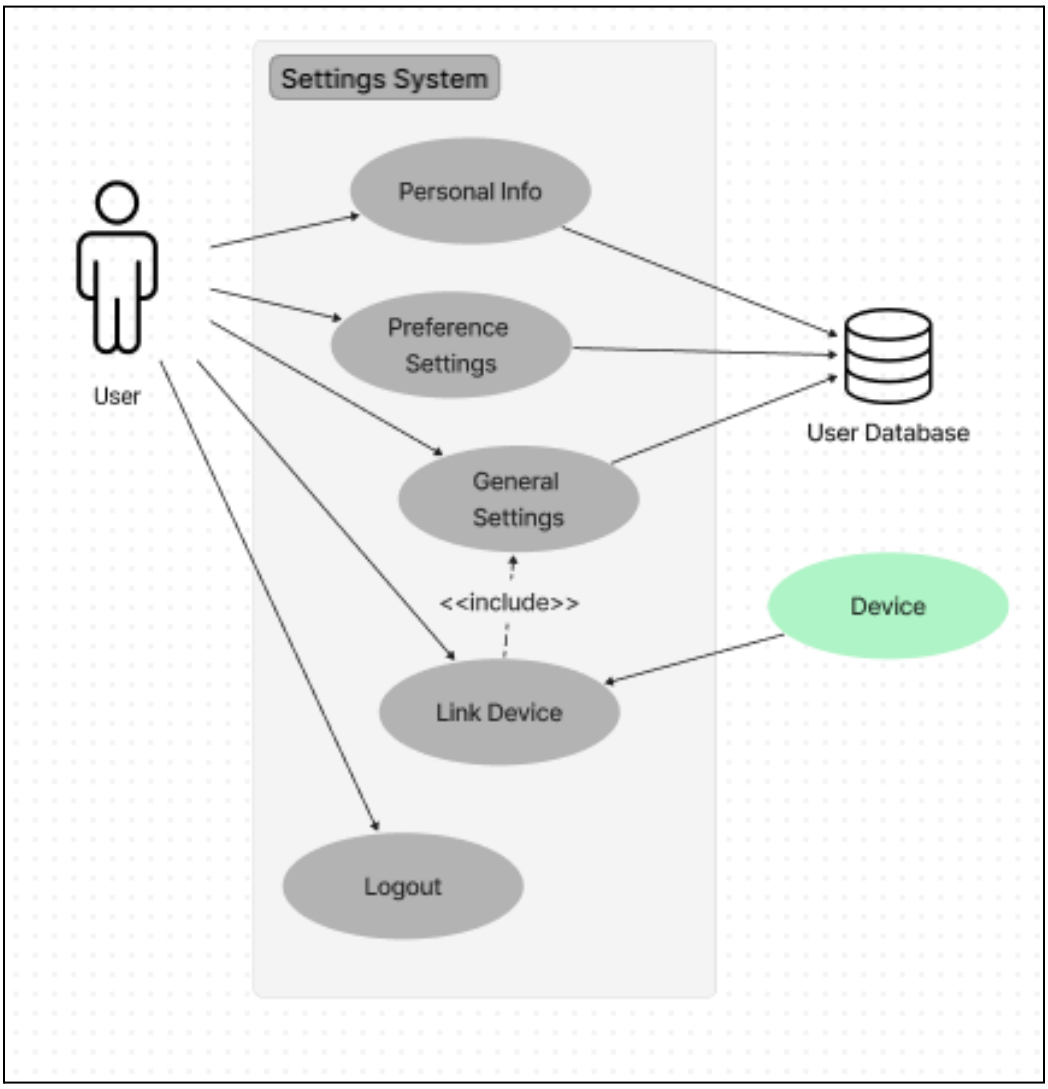


Figure 8.16. Settings system use case diagram

Figure 8.16 details user's access to their application settings. These include feature such as: personal info where they access/change their name or login credentials, preference where they reroute their diet choices, general settings and device linking that deals with application layouts and device linking, and logout where the user cut out all connections with account authorization and user database until the next relogin.

Overall, the Setting system allows the user to customize their experience with the app by allowing the right to access and edit specific areas of the database system that will completely change how the app generate recipes or display recommended recipes based on how the user want their diet styles to be, or how many smart inventory devices they want to include for tracking and extracting information.

8.3.4.1 User Settings Page and Logout Modal

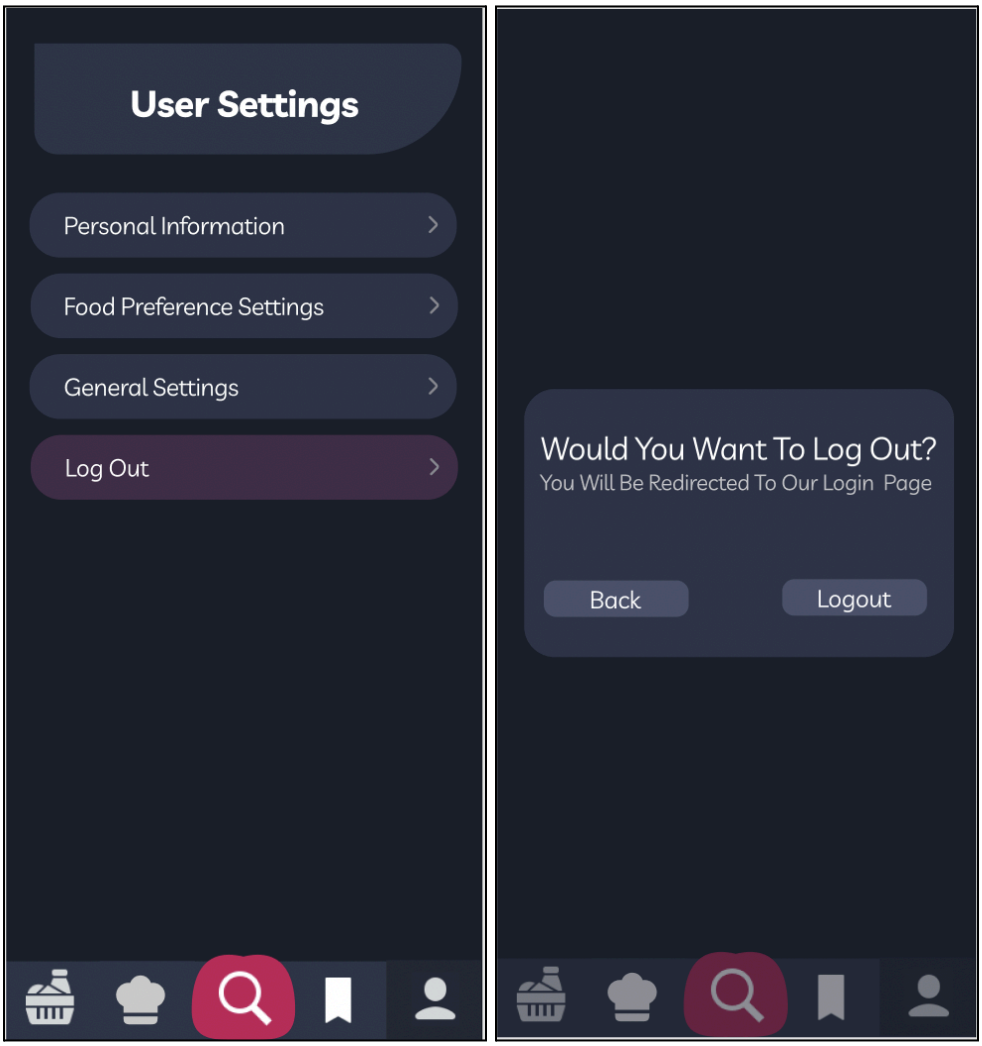


Figure 8.17. User Settings Page Figma

Figure 8.17 features the overall layout of the custom profile settings page. This includes features such as general settings and personal information where the user can change personal data they initialized during the signup stage, or changing certain hardware features to their preferences. Notably, preferences can be changed when users press on the preferences settings. This will redirect users to Figure 8.19, where they can change their eating preferences. Under devices linking, users may connect supported smart hardware like smart fridges to sync with the user inventory.

Users may also opt to log out of their account here, after which they will be directed back to the login screen (Figure 8.7). When this happens, the application will completely remove and lock all information regarding connection between the application and the user's database until their next relogin to the account.

8.3.4.2 Personal Information Page and Change Username or Password Page

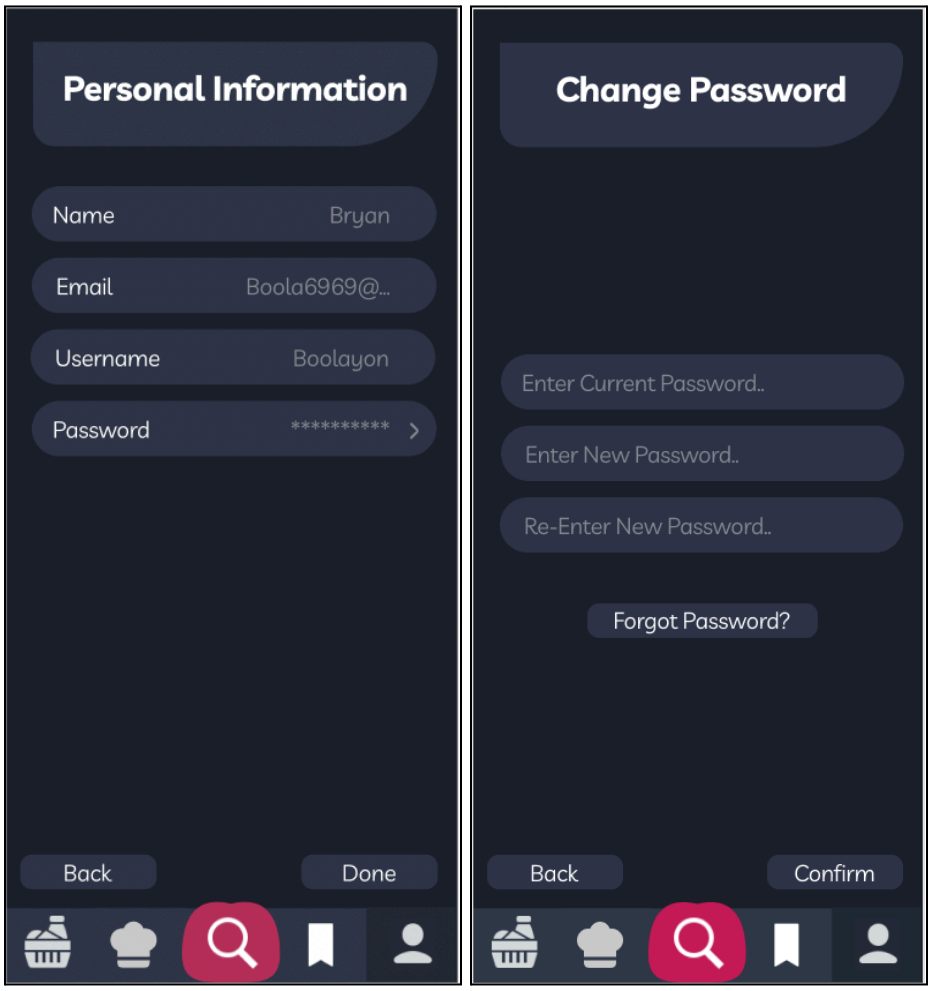


Figure 8.18. Personal Information Page

Figure 8.18 covers all personal information details that are all customizable by the user. The default is set through a series of questions that set up the user's page and devices. Personal information particularly pertains to a name, a username, and the email address specified during the registration process. Each of these fields can be changed directly except the user's password, which routes to a separate page within the app.

In order to ensure users' privacy, sensitive information such as the user's password is censored until specifically interacted. The reset password page has an additional option, where if the user forgot their password, the "Forgot Password?" button would route them to the send recovery email page where they can attempt to reset their password.

Additional features such as detection of existing usernames, or password format is also implemented. Ideally, the text boxes should have a built-in function where they read the user's input in real time and display its validity on the spot. Otherwise, the default implementation for this is a pop up notifying that the username or password is invalid.

8.3.4.3 General Settings Page

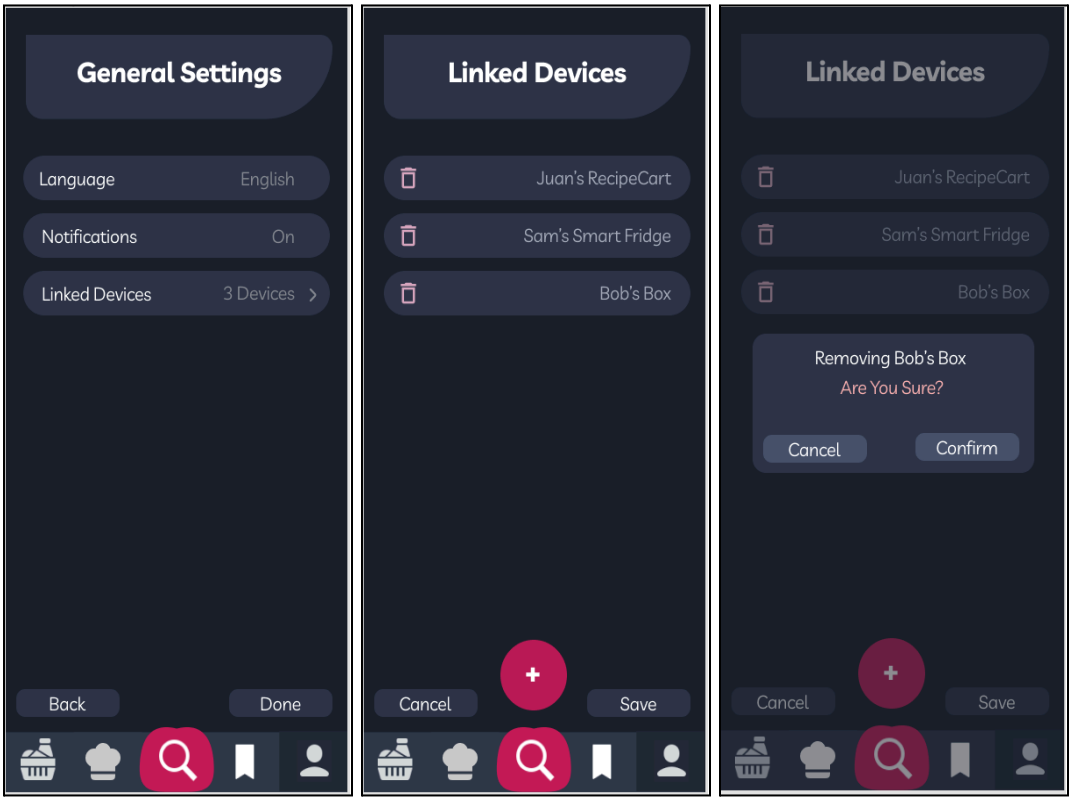


Figure 8.19. General Settings and Device Linking Figmas

Figure 8.19 demonstrates the general settings and device linking page. In the general settings page, English is set as default and the current only language available on the app. The notification settings dictates the application's permission to send information regarding updates, changes in recipe or user information. The device-link displays the number of smart storage devices that are currently connected to the app in order to track inventory.

The Device Link page, redirected from General Settings, lists out devices the app is using to track inventory. Devices are added using the "+" button at the bottom of the page, and are connected through Bluetooth, assuming most smart inventory uses Bluetooth to connect with respective applications. Each item listed includes an option to remove. After the user finishes modifying their device links, they can either save their options, or ignore their saves by tapping on the cancel button. When removing, a warning popup window appears to make sure the user wants to disconnect the device. When confirmed the device is removed from the list and the user is redirected back to the device listing layout. Otherwise the user can tap cancel or anywhere outside the screen to cancel their device removal process.

8.3.5 Food Preferences System

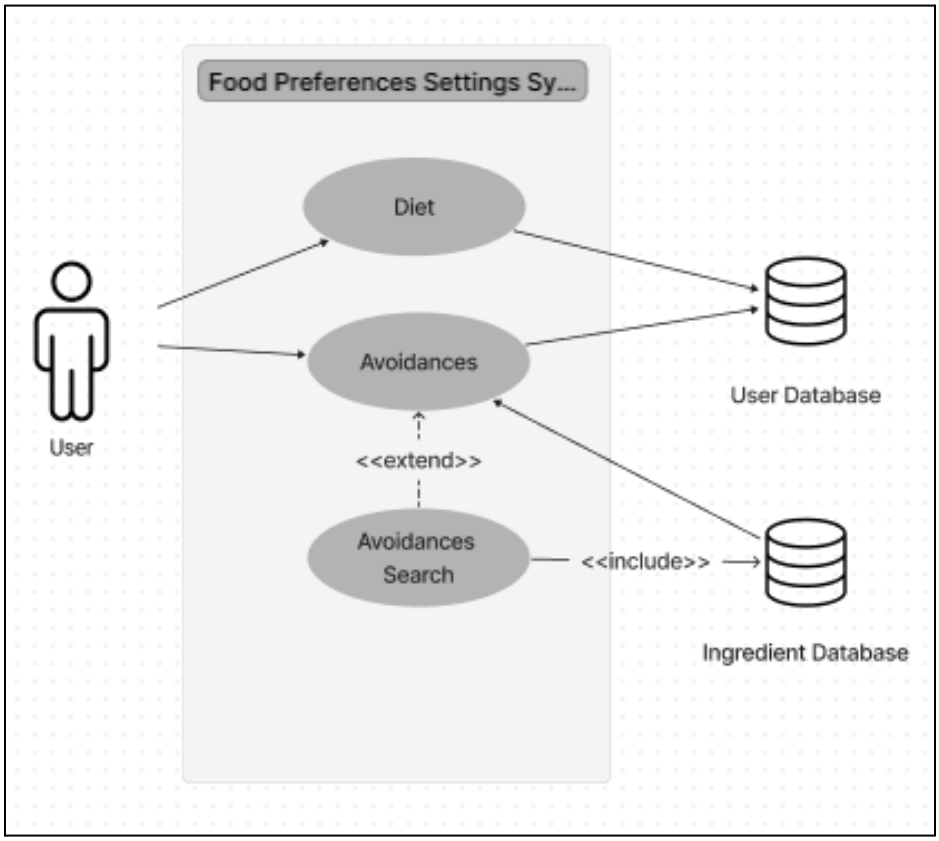


Figure 8.20. Food Preferences system use case diagram

Figure 8.20 depicts the user's access to the food preferences settings. The main function of this system is to allow the user to filter out specific approaches to the recipe that they want to see in the output. This includes customizing diet features such as recipes narrowed down to specific diet style such as vegan or gluten free, recipes narrowed down by specific regions such as mediterranean or asian, or recipes that users can make within a specified range of cooking experience.

Another function that the system allows the user to access is the ability to filter out food that they don't want in their diet. This creates the avoidance subsystem where they can search up food that they want to avoid, which the query will reach the ingredient database where it will pass specific information and images of food they don't want and return as a detailed list item which they can add to their list of avoidances.

Overall, the food preferences system allows the user to narrow down their recipe recommendation and generation system in order to cater to a specific range of the food they want and not want to see or make in their app. Both the diet and avoidance option that the user specified in their settings, will then be passed as a list in the user database where it will be passed as a set of white and blacklist for the recipe recommendation/generation system (see Figure 8.9).

8.3.5.1 Food Preferences Page

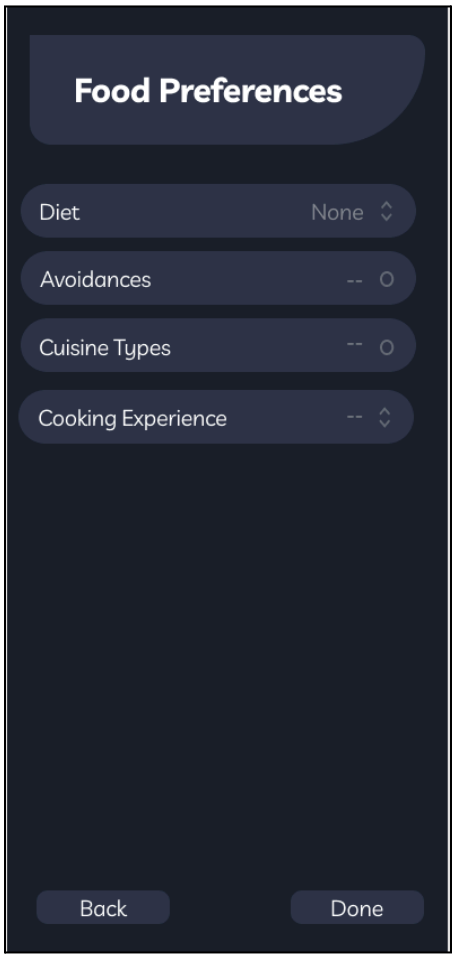


Figure 8.21. Food Preferences Page Figma

The food preferences page would be redirected from the signup page either after the user successfully signs up (see Figure 8.7), or when the user wants to change the preferences in their existing account by going through the Settings page (see Figure 8.17). The current preferences shown are language, diet, avoidances, cuisine types, and cooking experience. The app's only supported language is currently English, so it remains as the default option. However, it remains as an option because the app wants to expand to all types of audiences.

The diet and cooking experiences are drop-down formats where the user can choose one option of diet or cooking experience that fits the user best so that the app can filter out certain recipes that contain their diet that is within the difficulty of their comfort. The diet and avoidances settings are checklist format where users can select multiple options to filter out ingredients that they don't want in their food and different types of Cuisines that users would like to see in their recipe recommendations. The back button would either link users back to the login page if they access this page through the sign up page or the user settings page if they access from there.

8.3.5.2 Diet Selection Dropdown

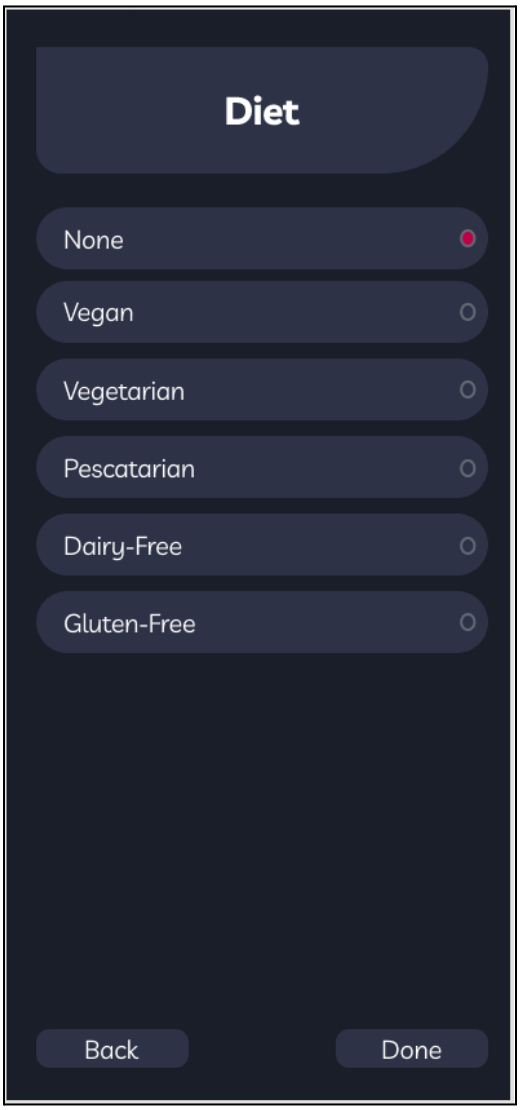


Figure 8.22. Diet Selection Dropdown Figma

Figure 8.22 shows a sampled Diet Selection Dropdown page where users can choose one diet option to follow. The default option for diet will always be none, assuming that the user is comfortable with all types of cuisines. If the user is committed to a specific diet, especially detailed and sensitive diets such as vegan, it is necessary for them to access this page and change it so that the system can cater to their needs. The suggestion algorithm will be changed accordingly based on the option chosen. The back and done button both redirect the user back to the Profile's preference page, where the back button will unsave the current session that the user entered and potentially edited, and the done button will go back to the preference settings while saving what the user specified in their last access to the diet list dropdown.

8.3.5.3 Avoidance Selection Checklist and Search Modal

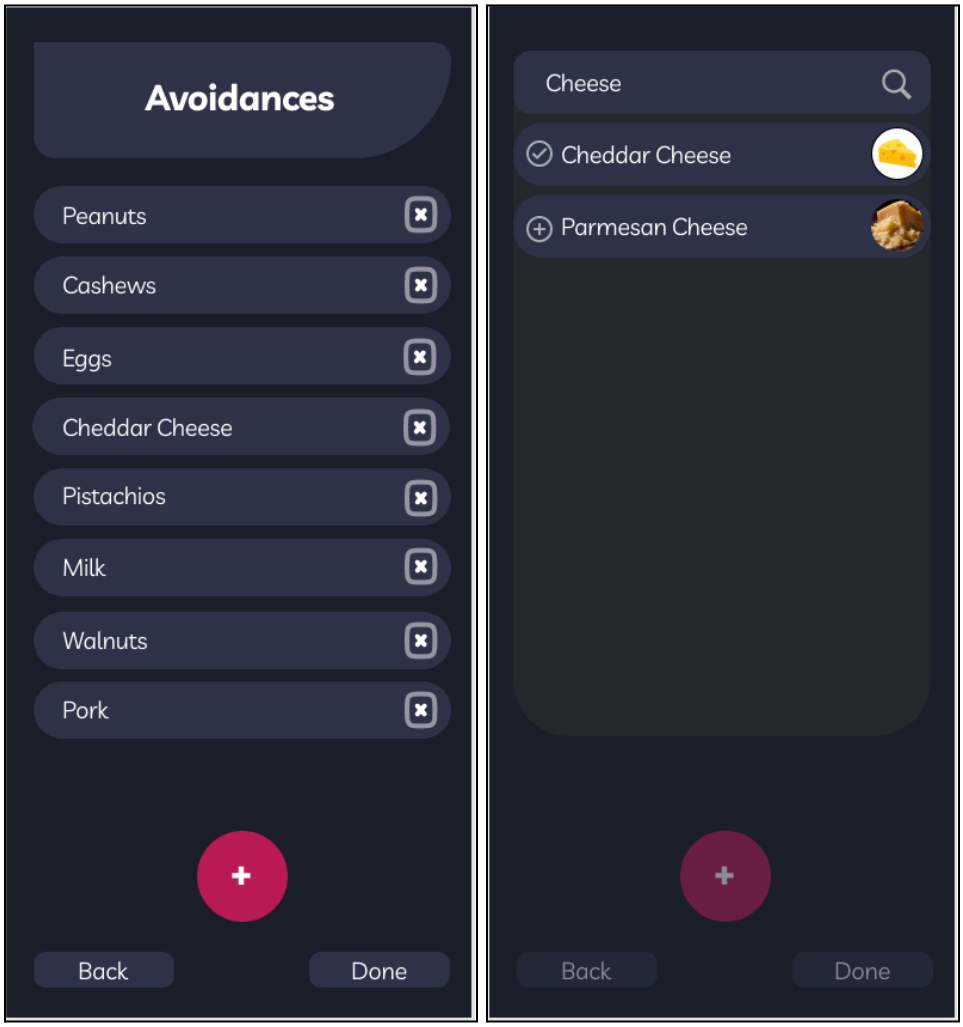


Figure 8.23. Avoidance Selection Checklist and Search Modal Figmas

In Figure 8.23, the diagram on the left depicts the edit panel with a dropdown list of the user's avoidances. The user may remove previously selected avoidances by tapping on the associated "x" buttons. The back and done buttons both redirect the user back to the Profile Preferences page.

The right figure shows the pop up modal upon pressing the "+" button. Users may leverage the search functionalities to select and add ingredients to add to avoidances. Items appearing in the search are expected to support multiselect. Upon pressing "Done", the selected ingredients would be displayed in the list shown in the right diagram.

9.0 Implementation and Prototyping

This section discusses the methods involved in implementing and prototyping the RecipeCart. Particularly, we describe the procedures in installing the hardware and constructing the software architecture.

9.1 Hardware Setup and Implementation

The purpose of the RecipeCart is to demonstrate the flexibility and compactness of the ingredient detection solution. All the hardware components are thus assembled onto a portable container. With the SBC functioning as the heart of the architecture, the camera and loading platform are placed in close proximity to the Jetson Nano to minimize broadcast delay and wire entanglement.

Aside from mounting the Wi-Fi module and the Raspberry Pi camera, the hardware setup and prototyping primarily focuses on hooking up the digital scale to the HX711 load cell amplifier and integrating the weight system with the Jetson Nano.

9.1.1 Initializing the Jetson Nano and Raspberry Pi Camera Setup

Following the setup instructions outlined on the Jetson Nano Developer Kit webpage, we begin by loading the microSD card into the Jetson Nano and formatting the storage space before installing the default Linux operating system image. We then connect the Jetson Nano to a computer display via the DisplayPort and a keyboard and mouse through the USB ports. This final step allows us to complete the Jetson Nano setup via a GUI.

To enable wireless connectivity, we proceed to unlatch the Jetson Nano from its carrier board and insert the AC8265 NIC into its slot. We then unlatch the CSI slot and insert the band from the Raspberry Pi camera, ensuring it is firmly clipped into the slot.

We also need to install OpenCV with CUDA compatibility on the Jetson Nano to allow the barcode detection software to take advantage of the onboard Maxwell GPU. Jtop is installed to monitor Jetson-related system statistics and to confirm successful configuration of the OpenCV library [9.1].

9.1.2 Prototyping the Weight System

In building the weight scale system, we solder the scale's data wires to the HX711's E+, E-, A+, and A- gates. The VCC and GND gates are latched onto the Jetson Nano to provide power to the board. Finally, we connect the CLK and DATA gates to the Jetson Nano's pins 7 and 11, respectively, to enable data serialization.

Provided a successful linking process between the Jetson Nano and the HX711, we next configure the Jetson Nano with a custom driver to control the HX711 and interpolate the

received data. We refer to a HX711 Python library on GitHub that builds on an existing HX711 codebase originally developed for the Raspberry Pi mini-computer [9.2].

We then modify the source code to reflect updates in the *libgpiod* Python library used to control the Jetson's GPIO pins. We also adjust the reference unit according to the sensitivity of the kitchen scale—in our particular case, we found 320 to be the best estimated reference unit.

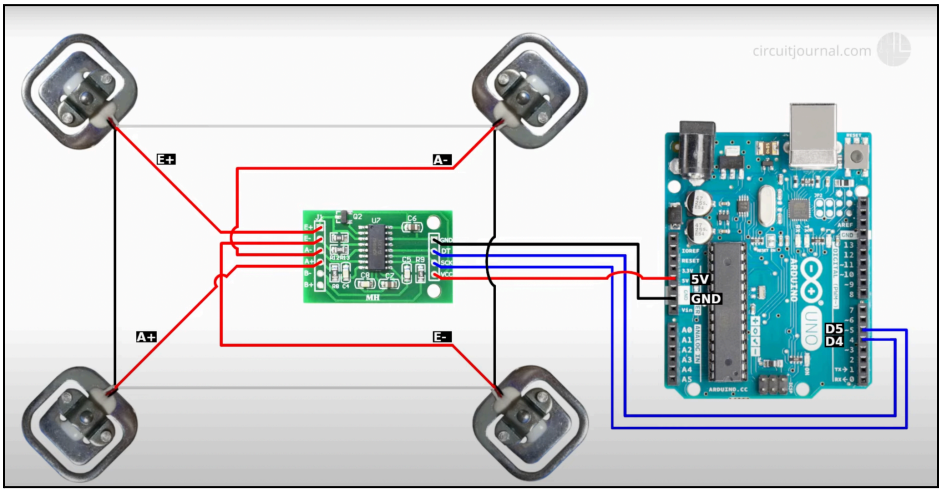


Figure 9.1. HX711 fully connected to load cells and example SBC [9.3]

9.2 Software Implementation

The software implementation is divided into six main sections: the IoT, the databases, the ingredient classification software, the backend serverless Lambda functions, the recipe search system, and the Flutter-based mobile frontend.

9.2.1 Configuring AWS IoT Core and MQTT Protocol on Jetson Nano

The Systems Manager agent is first installed onto the Jetson Nano via AWS Greengrass and given the necessary IAM roles and permissions to interact with the System Manager mother console. The AWS IoT Greengrass token exchange role for the IoT Thing (i.e. the Jetson Nano IoT device as seen by AWS) must also be updated to reflect the new System Manager permissions. Certificates and keys for AWS communication are placed within the same directory as the barcode detection script to allow the Jetson Nano to forward messages to the MQTT broker in AWS IoT Core.

We install the AWS IoT Python SDK which includes methods for publishing and subscribing to MQTT messages brokered by AWS. In a Python file called *publish.py*, we define the IoT endpoint, client ID, and credentials needed to connect to the MQTT server in AWS IoT Core. We sectionalize the publishing topic header as *{deviceID}/{cognitoID}*, where the device ID refers to the Jetson Nano's instance ID as

perceived in AWS and the Cognito ID refers to the ID of the user who initiated the barcode detection request.

In a separate file called *barcode.py*, we first retrieve the process ID (PID) for the barcode detection script and write it to a file for later reference when terminating the program. We specify the camera capture quality to be 1280p x 720 to obtain a 60 fps frame rate without overly diminishing the video quality and view. We then obtain a connection ID to the Raspberry Pi camera via the gstreamer pipeline and forward it to OpenCV's video capture object. Barcodes are detected and decoded every 15 frames to save compute power and minimize duplications. The detected barcode is packaged inside a JSON object and published to the specified MQTT topic in AWS. Small time buffers are inserted throughout the barcode detection loop to prevent overlap and data duplication.

Throughout the barcode detection process, the script plays sounds through the *pygame* Python library to indicate whether the barcode detection system has booted, has successfully or unsuccessfully scanned and published the barcode message, and has terminated. Sounds are provided by the team members themselves.

9.2.2 Deploying and Populating Weaviate in Kubernetes

For scalable deployment, the Weaviate Database is hosted as a pod in a kubernetes cluster formed from AWS EC2 instances. We use KOPS to instantiate the kubernetes cluster because it provides high-quality and easy-to-manage clusters with ample supporting documentation. Weaviate requires that kubernetes be of version 1.23 and above. At the moment of this writing, KOPS requires that kubernetes be of version 1.24 or below for deployment on AWS due to a DNS failure to resolve bug. For that reason we run this script on kubernetes 1.23. Once a cluster has successfully formed, Weaviate can be deployed as a helm repository, and its configuration can be altered with a value.yaml file. The *kubectl get services* command can be executed to retrieve the hosting DNS or cluster IP. Using this address, we can input datasets into our Weaviate database by crawling image or recipe datasets, generating schemas and embeddings, and adding the data in batches. Weaviate offers a Python client that abstracts away much of the graphql-style underlying queries, which greatly simplified the process of cleaning and inputting the object data.

9.2.3 Deploying Meta DINOv2 through AWS Lambda

The Meta DINOv2 code is packaged as a Docker container with all its dependencies prior to loading it into AWS Lambda due to the sheer size of the PyTorch library required. The code itself includes a process for cleaning and formatting the input image before running it through the DINOv2 model to convert the image bytes into vector embeddings.

The Lambda then calls the Weaviate client to fetch the vector embeddings and their associated labels with a certain similarity threshold to the input image vector. The classification is then performed based on the K-Nearest-Neighbors algorithm. The topmost classification label is returned to the mobile frontend for user confirmation.

9.2.4 Deploying and Configuring Backend Lambda Functions

There are a total of six Lambda functions callable from the Amplify backend server.

The first Lambda is known as a "post confirmation trigger" Lambda and, as the name implies, it executes the Lambda upon the user successfully verifying their email during the registration process. This Lambda initializes an empty user settings object corresponding to the registered user and injects it into the DynamoDB through a GraphQL API endpoint. Each user can only have one settings instance which can only be created upon user registration, hence the purpose of this Lambda.

The "Start-Stop" Lambda's primary function is to initiate and terminate the barcode detection session on the Jetson Nano via AWS Systems Manager. The Lambda first fetches the Jetson Nano's managed instance ID from the SSM client before sending the designated command. The "Start" command initiates a virtual Python runtime environment and proceeds to run the barcode detection script. The "Stop" command simply kills the barcode detection script via its process ID as determined within the barcode detection Python program.

```
# get barcode Python script directory from environment variable
startScript = '''
cd {}
. barcode_venv/bin/activate
nohup python barcode3.py {} >/dev/null 2>&1 &
'''.format(os.environ['BARCODE_DIRECTORY'], cognitoID)

stopScript = """
cd {}
kill `cat app.pid`
""".format(os.environ['BARCODE_DIRECTORY'])
```

Figure 9.2. Start and Stop Shell Command Scripts for Barcode Detection

The "ProcessBarcode" Lambda retrieves the product information associated with the queried barcode. It first queries DynamoDB to check if the barcode record already exists within the local database. If no record is found, the Lambda attempts to fetch the relevant product information from the World Open Food Facts API. The resulting JSON is parsed through for the ingredient name and quantity. If no name is found, the Lambda returns "Unknown" as the ingredient name—unknown ingredients cannot be subsequently added to the user's inventory. With the ingredient name, the Lambda then queries the Weaviate vector database's ingredient table to obtain a list of related names that may facilitate name conformity when later searching for recipes based on ingredients. The ingredient metadata is finally returned to the mobile frontend.

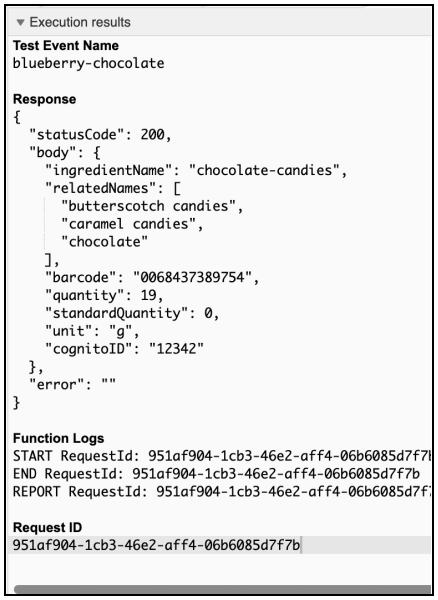


Figure 9.3. Sample JSON Response from ProcessBarcode Lambda

The "DINOv2 Weaviate" Lambda is the containerized function from Section 9.2.3.

The "UpdateRecipeRating" Lambda's functionality is relatively straightforward: it updates the recipe ratings in the Weaviate database's recipe table to reflect the user updates made on the mobile frontend.

Finally, the "RecipeSearchWeaviate" Lambda handles the recommendation system and recipe search feature of the RecipeCart. It creates a mask of "blacklisted" ingredients to represent the avoidances and the user's diet type and applies it as a hard, post-filter on the returned recipes. Recipes are retrieved from Weaviate based on their ingredient list's similarity to the concatenated list of related ingredient names (obtained from applying a union between all the inventory ingredients' *relatedNames* fields). The resulting list of recipes is then sorted based on ingredient similarity and average ratings.

The Amplify backend server must have IAM permissions to execute all six of these Lambdas, which must be specified in the override.ts file within the awscloudformation folder [9.4]. The file is revealed after running the *amplify override project* command.

9.2.5 Building a Recipe Search with Recommendations

The recipe recommendation system is divided into a filtering algorithm and a sorting algorithm. The filtering process is performed based on the user's diet type and ingredient avoidances—information unique to each user stored in the settings table. The sorting algorithm employs the recipes' average ratings as well as the ingredients currently held in the user's inventory.

The ideal recommendation system takes into account the recency of the recipes' popularity metrics by tracking the time period in which certain recipes gain traction. Such

additional fields would allow us to sort recipes based on metrics such as ”most viewed within this month” or ”most likes within the past week.”

In light of the time constraints and project complications, we are implementing a simplified version of this content-based recommendation system that only looks at the global metrics such as recipe ratings.

9.2.6 Developing a Flutter Mobile Frontend with AWS Amplify

In developing the Flutter-based frontend, we closely follow the steps outlined in the AWS guide to building Flutter apps with AWS Amplify [9.5]. We wrap the app with the Amplify Cognito Authenticator plug-in which is a pre-configured, out-of-the-box authentication widget. We then initialize the GraphQL API layer and specify the GraphQL model schemas.

The Flutter-based API layer is separated into three levels as governed by Flutter Riverpod best practices: an API service layer containing the actual API logics and calls, a data layer containing the API repository, and a controller layer with the API provider controller to expose the API functions to the frontend user interface.

The user interface is meanwhile composed of four main pages controlled through a navigation bar and a router provider.

The inventory page is the landing page and displays to the user all the ingredients currently held in their inventory from which the user may search through via the search anchor widget. Pressing the ‘plus’ icon initiates the ingredient detection process and connects the user to the Jetson Nano. The scanned barcodes are brokered by an MQTT client and directly streamed via a StreamBuilder widget to the frontend. Incoming barcode messages are passed through the “ProcessBarcode” Lambda to return the appropriate ingredient information for user display. Users may also press the camera button to pop up their mobile camera and instead take a picture of the food item in question. The events are again streamed through a FutureBuilder as we invoke the “DINOv2 Weaviate” Lambda. Detected ingredients are displayed to the user for confirmation before being added to their inventory.

The search page returns all relevant recipes based on the current ingredient inventory by default. However, users may specify what specific recipe they would like to explore, and the search algorithm will return an aggregated list of recipes sorted by user preferences and ingredient availability. Recipes can be individually viewed, saved, and rated.

The saved recipes page simply displays all the user’s saved recipes for quick look up and future reference. The list of saved recipes are obtained from a series of pointer references stored in each user’s settings table.

The settings page currently only contains methods for the user to modify their dietary preferences and avoidances as well as a logout button.

10.0 System Testing and Results

Throughout the development process of the RecipeCart, it is crucial to regularly test components to ensure proper functionality and compatibility with the current system and adherence to the engineering specifications specified in Table 2.1 and the house of quality in Figure 2.7. This section is dedicated to outlining specific testing methods that may be employed for each key component of the project.

10.1 Hardware Specific Testing

Having bypassed the requirements to design a custom PCB, the RecipeCart's hardware testing reduces down to simply testing the functionalities of the Jetson Nano, the Raspberry Pi camera, and the weight scale system.

10.1.1 Jetson Nano Network Connectivity Testing

After performing the basic setup instructions, the Jetson Nano Linux-based operating system should be accessible through a GUI or the command line. The Jetson Nano's basic functionalities can be tested by connecting the SBC to a monitor via the HDMI cable or to a computer screen via the microUSB cable. Successful powering of the Jetson Nano is characterized by a green LED located near the microUSB slot on the carrier board.

To test for reliable network connectivity, the Jetson Nano can attempt to access the Internet through its default operating system's browser. For the purpose of the RecipeCart, a consistent and stable network connection can be observed by loading a script onto the Jetson Nano to send a set of empty packets to the backend server. The response time should be recorded and compared to the specified broadcast delay.

10.1.2 Raspberry Pi Camera Integration Testing

Upon inserting the Raspberry Pi camera into the Jetson Nano's CSI lane, sample computer vision programs can be found on various publicly available repositories and loaded onto the Jetson Nano to test the camera's quality and auto-focus accuracy. Writing a simple program to capture pictures and package them as requests to the server may also be relevant to testing the constraints of the image-based ingredient classification pipeline. Taken pictures should be checked for the appropriate data format and size. The quality of the produced images is directly related to the potential ingredient classification accuracy design specification.

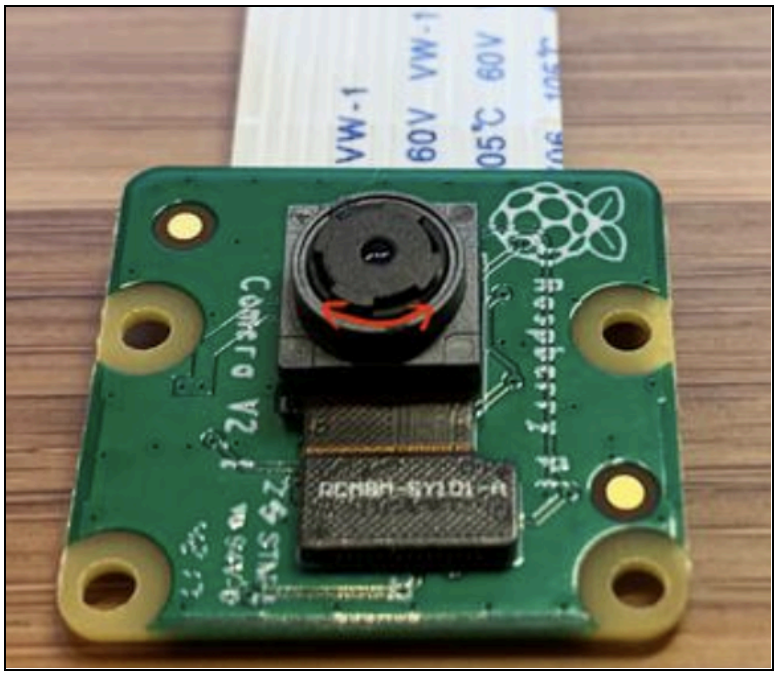


Figure 10.1. Rotating the Outer Lens to Adjust Focus

The Raspberry Pi Module 2 camera's focus is manually adjusted to a depth of field of roughly 10 cm away from the camera lens to capture clear frames of the barcodes up close. The manual adjustment entails rotating the camera's outer lens with a pair of tweezers.

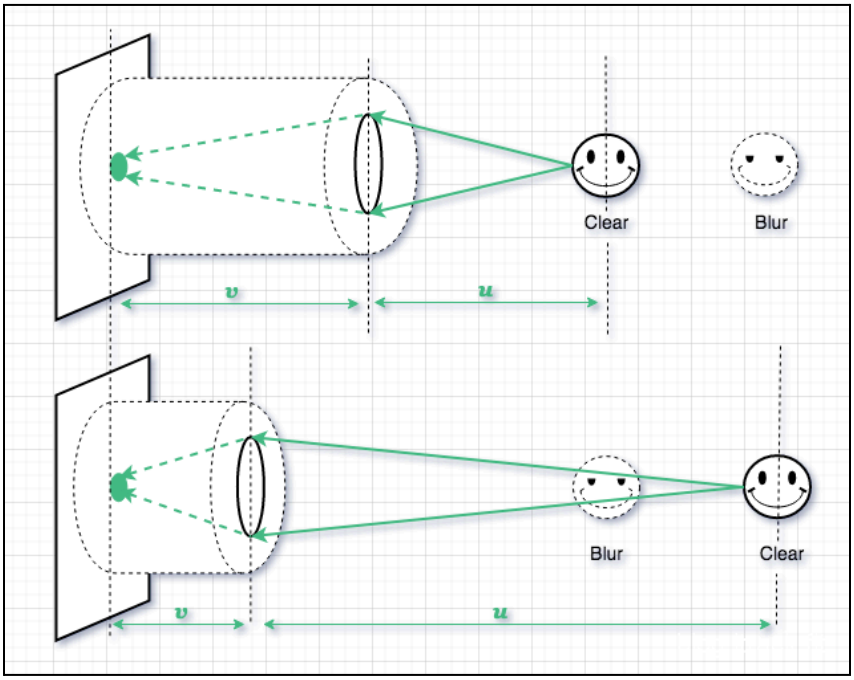


Figure 10.2. Figure Lens Diagram

10.1.3 HX711 and Weight Sensor Integration Testing

In testing the RecipeCart's weighting system, we first observe the HX711's connectivity with the digital kitchen scale, which can be evaluated through a continuity check using a digital multimeter. The next step is to check the HX711's connection with the Jetson Nano, which can be done by setting the Jetson Nano to listen to the serial data port linked to the HX711. We can then devise a program to continuously fetch the data from the HX711. The received data is expected to be uncalibrated and unscaled, requiring some form of data cleaning and reformatting to empirical or metric units.

Testing of the entire weight scale system can be done manually by individually placing a selection of food ingredients on the bathroom scale, checking its output, and comparing that quantity with the values retrieved by the program on the Jetson Nano. The results from the HX711-amplified data are expected to be slightly different from the digital bathroom scale due to the inevitable voltage fluctuations relating to the HX711.

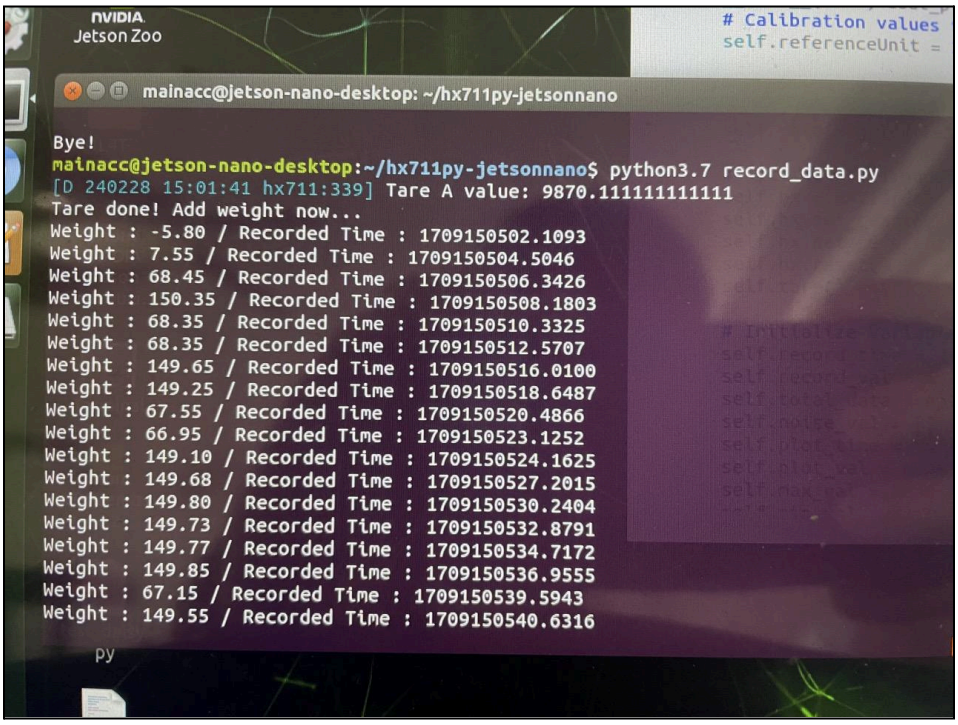


Figure 10.3. Weight Values Retrieved from HX711

Despite having calibrated the HX711 Python-based driver program, the resulting weight values still experience significant fluctuations, and the program occasionally erroneously outputs weight values in the negatives. Furthermore, the computed weights are not adequately precise, often exceeding the tolerable margin of error of 20 grams. Implementing a more precise weight system either requires a significantly better digital scale or a less disturbed HX711 integration, both conditions of which are not achievable within the resources and scope of this project. Consequently, we decided to omit the weight system from the current development of the RecipeCart.

10.2 Software Specific Testing

In this section, we detail how we intend to evaluate its adherence to design constraints, its connectivity, and its performance. These tests can range in complexity from simply testing connectivity to gathering complex performance metrics to evaluate viability and scalability.

First, all software must be tested for adherence to the design constraints established in Table 2.1. Object classification time and accuracy are heavily dependent on the classification model's quality. To test the classification model's quality, there are a number of potentially useful metrics to evaluate its accuracy aside from the typical accuracy measure; these include but are not limited to the AUC curve, the precision metric, the recall metric, the f-1 score (a hybrid between recall and precision), and the classification rate. The recipe recommendation time and minimum recipe recommendation can be evaluated by performing stress tests on the recommendation system by making some increasingly complex or bizarre queries. Furthermore, the recommendation quality can be tested by gathering user satisfaction with the overall product and presentation of the software.

10.2.1 World Open Food Facts Database Connectivity Testing

Access to the barcode database and speed of queries is directly related to two major design constraints of RecipeCart. First, the time required to query from an external database will likely constitute the bulk of the barcode-based classification time. And the associated connectivity of the Barcode database directly corresponds to the viability of barcode-based classification, which has extremely high accuracy. The product metadata in World Open Food Facts Database can be tested by compiling a list of barcodes and querying them to check for data conformity and consistency. The returned information should include quantity information, nutritional information, and a proper, relatively standardized name for the ingredient, of which we mainly need the name of ingredients and their respective quantity.

Figure 10.4. Sample Product Information Retrieved from WOFF

10.2.2 OpenCV and Pyzbar Barcode Detection Testing

The barcode detection classification is heavily dependent on the object detection mechanism; if the object detection returns a noisy output, it may result in failing to decode the barcode which will incur classification time and accuracy costs. The combination of OpenCV and Pyzbar can be tested by compiling a set of ingredients with barcodes, showing them to the camera, and observing their detection speed and decoding accuracy.

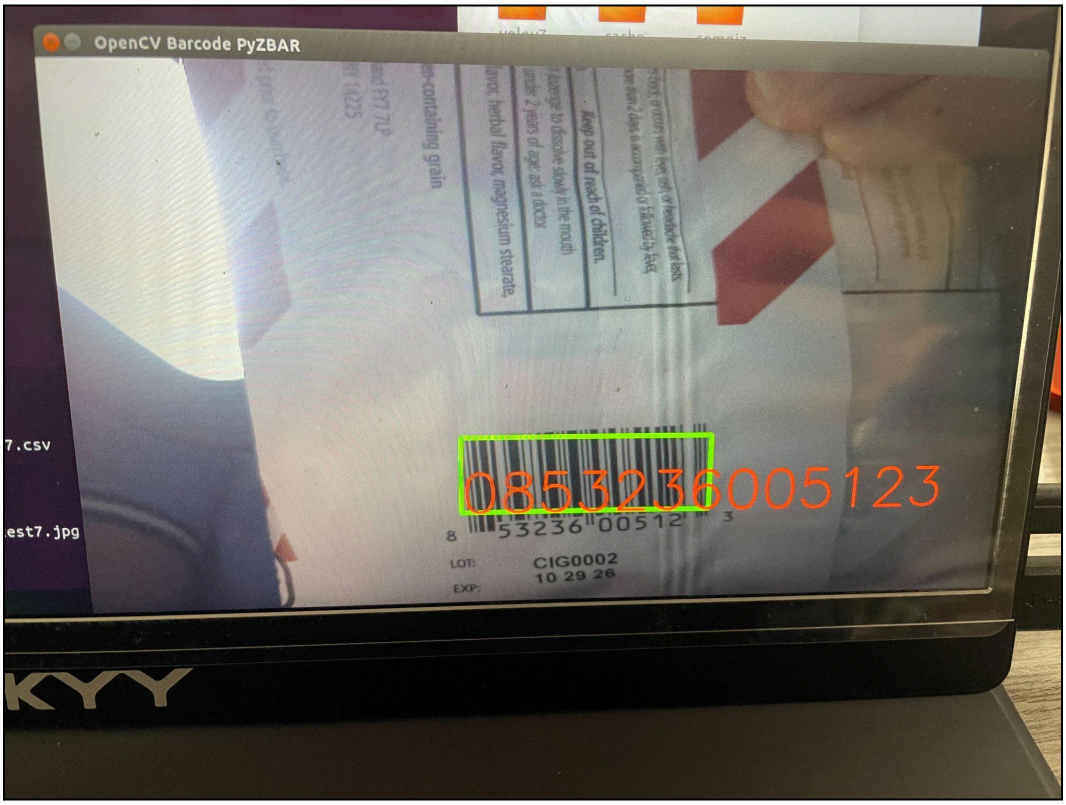


Figure 10.5. Pyzbar successfully decoding the barcode of a bag of cough drops

10.2.3 Meta DINOv2 Object Classification Testing

Image-based classification accuracy and time is heavily dependent on the performance and complexity of the image classifier. For RecipeCart's image classification system, the performance of DINOv2 can be tested by compiling a set of ingredients that are visually distinct, showing them to the camera, and observing the detection speed and embedding quality. High quality embeddings should result in better classification rates as the same ingredients should map to similar regions in space.

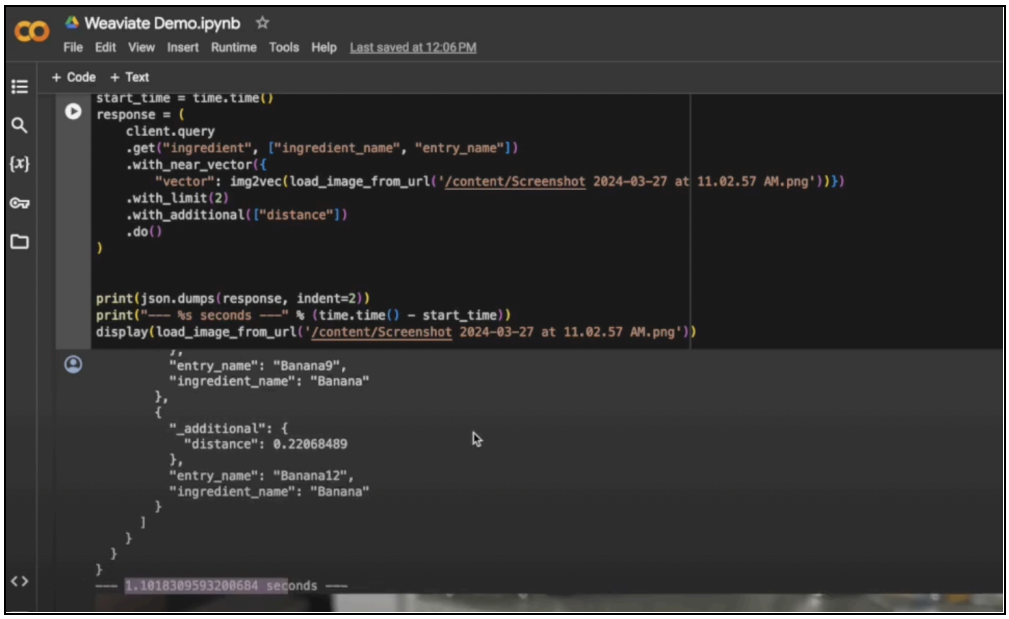


Figure 10.6. Image Classified as 'Banana' by Meta DINOv2

We see in the above screenshot that the banana is correctly identified in roughly 1.1 seconds, which easily satisfies the design constraint of having ingredients classified in under 5 seconds.

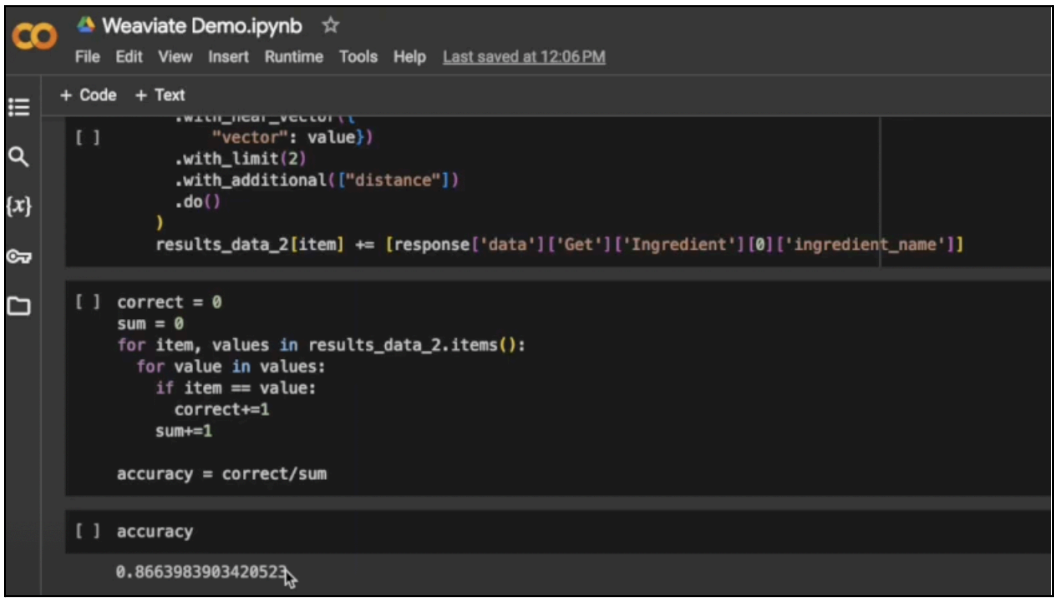


Figure 10.7. Total Accuracy of Image-Based Classification Pipeline

The above screenshot shows a total accuracy of 86.6 % for the image-based classification pipeline leveraging Meta DINOv2 to reverse search the vector embeddings stored in the Weaviate vector database. This testing result affirms the 80 % or above classification accuracy design specification.

10.2.4 AWS Backend Connectivity

AWS is the central component for all API calls and Database queries. All API calls for the RecipeCart will have to pass through the AWS backend. As such, it is important to ensure that the cloud platform properly connects to all other backend components. Testing revolves around ensuring component compatibility and communication of data.

The Weaviate Vector Database in the Kubernetes cluster was successfully tested for connectivity with the Weaviate client in the containerized Lambda. The image pre-processing and vector embedding through Meta Dinov2 was externally tested in a Jupyter notebook on Google Colab before being packaged into a Docker container and loaded as a Lambda function. The entire object classification pipeline by means of vector embedding reverse search works as a singular microservice and should be easily integrated into the rest of the RecipeCart system.

The object classification pipeline satisfies two of the three main engineering specifications by consistently achieving a classification accuracy of 85% and boasting a classification time of under 2 seconds.

10.2.5 User Settings Table GraphQL API Testing

The user settings table will contain the bulk of user-generated data; these include but are not limited to usernames, profile settings, avoidances, saved recipes, and diet type. No user operations would function properly if the server is not properly connected to the user database. We test the settings table by making a series of queries and ensuring the appropriate output through the API.

Figure 10.8. Sample GraphQL Query on the Settings Table

The above query is performed through AWS AppSync, Amazon's serverless GraphQL solution. The query is run on a secondary index table in DynamoDB and returns the fields specified in the 'items' object.

10.2.6 Ingredient Table GraphQL API Testing

An ingredient database is necessary for enabling ingredient search, which will prove useful for setting up avoidances or managing the ingredient inventory. Since users directly interact with the ingredient database often, its information needs to be readily available and quickly accessible. To test this, we can evaluate time it takes to return an ingredient id and other relevant information when ingredients are queried from the database.

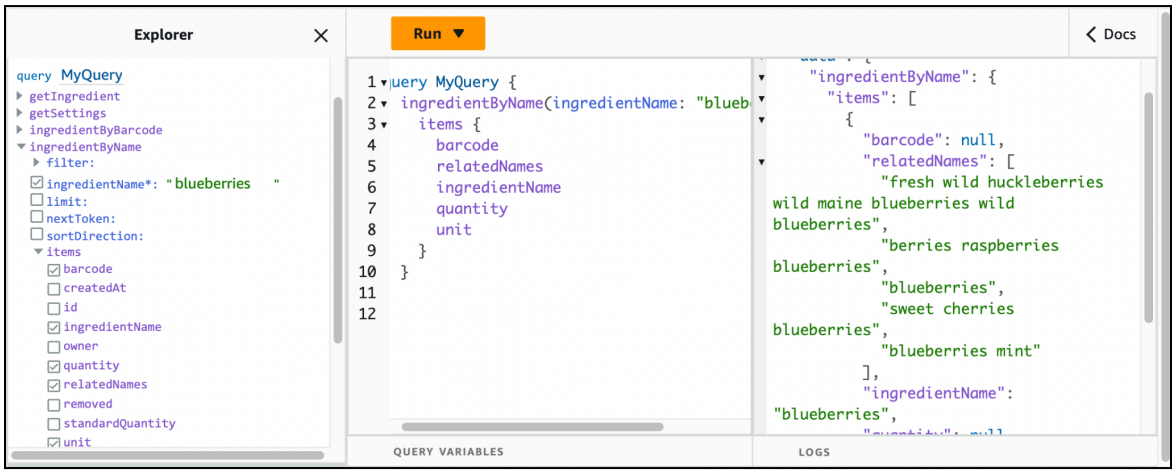


Figure 10.9. Sample GraphQL Query on the Ingredients Table

The above query is also performed through AWS AppSync and also queries on a secondary index table of the original Ingredients table.

10.2.7 Recipe Vector Database API Testing

Immediate access to the recipe database which is constantly subject to change is crucial to the effectiveness of recipe recommendation. Assuming extreme scales of recipe data, the recipe database is expected to perform filtering in a sufficiently efficient way such that queries do not take too long. When recipes are queried from the database, it should return a recipe name, a set of ingredients, cooking instructions, and an associated rating.

```
{
  "data": {
    "Get": {
      "Recipe": [
        {
          "_additional": {
            "id": "03e50dbe-9b7a-44a4-85b3-5521adf67b73"
          },
          "ingredients_sliced": [
            "1 daikon radish, peeled or zucchini",
            "4 carrots, peeled",
            "4 cups mung bean sprouts",
            "4 scallions, thinly sliced",
            "12 oz organic, GMO-free firm tofu, cut in quarters",
            "1 small handful cilantro leaves, chopped, plus extra for serving",
            "2 tablespoons black sesame seeds, plus extra for serving",
            "4 slices of lime, to serve",
            "1/2 cup peanut butter (adjust to allergies by choosing a different seed or nut butter)",
            "4 tablespoons lime juice",
            "2 tablespoons clear honey, preferably unheated",
            "2 tablespoons organic, GMO-free tamari or soy sauce",
            "1 pinch ground cayenne pepper or more to taste",
            "2 teaspoons grated fresh ginger",
            "About 3 tablespoons water",
            "to thin"
          ],
          "instructions": "Use a julienne peeler, mandoline or spiralizer (or even a potato peeler) to c
          "rating": [
            0,
            0,
            0,
            0,
            0
          ],
```


Figure 10.10. Sample Recipe Vector Stored in Weaviate

10.2.8 Recipe Recommendation System Testing

The recipe recommendation quality and efficiency is heavily influenced by the design of the recommendation system, which itself can be influenced by a number of factors including algorithmic complexity, embedding quality, or storage medium. To test the recommendation system, we isolate the different factors of the recommendation system and focus on the most important factor. The most important component of the recommendation system can be determined by checking for the hyperparameters that accelerate the filtering process while ensuring that recommendations still adhere to user personal preferences and dietary constraints. Since user engagement is a subjective metric, as long as the suggested recipes do not fall into the pitfall of recommending the same recipe repeatedly and does not recommend recipes violating dietary constraints, the recommendation system will be considered viable.

```
response = (
    client.query
    .get("Recipe", ["title", "instructions"])
    .with_near_text({
      "concepts": ["noodles"],
      "moveAwayFrom": {
      "concepts": ["asian"],
      "force": 0.0
      },
      "moveTo": {
        "concepts": ['french'],
        "force": 0.99
      }
    })
    .with_limit(10)
    .do()
)

print(json.dumps(response, indent=2))
```

```
{
  "data": {
    "Get": {
      "Recipe": [
        {
          "instructions": "Cook noodles according to package directions. During the la
          "title": "Ramen Noodle Bowl with Escarole and Spicy Tofu Crumbles"
        },
        {
          "instructions": "Cook noodles according to package directions. Drain, transf
          "title": "Cold Rice Noodle Salad With Chicken, Herbs, and Cucumbers"
        },
        {
          "instructions": "In a large pot of boiling water, cook the s\u014dmen noodle
          "title": "Summertime S\u014dmen Noodles"
```

Figure 10.11. Sample Recipe Recommendations

The above query mimics a recipe search where the user wants recipes involving noodles cooked in an Asian fashion. The topmost outputs reflect this search request.

10.2.9 Mobile Frontend Unit Testing

Before beginning tests for UI/UX, it is important to ensure that each button or action works as intended. First, we navigate the different tabs to ensure that each button or object is displayed as intended. Next, we explore every button and interactables and trace the input and outputs to ensure that they match their intended use.

10.2.10 Mobile UI/UX Testing

Lying at the interface between users and the server is the Mobile's UI/UX design. If the UI/UX is not sufficiently intuitive or is unattractive, it may discourage users from wanting to interact and use RecipeCart. Testing the UI/UX for user satisfaction is crucial to developing a user-friendly application. To that end, we gather a focus group to explore the menu options of RecipeCart and gather their feedback for further improvement to the UI design.

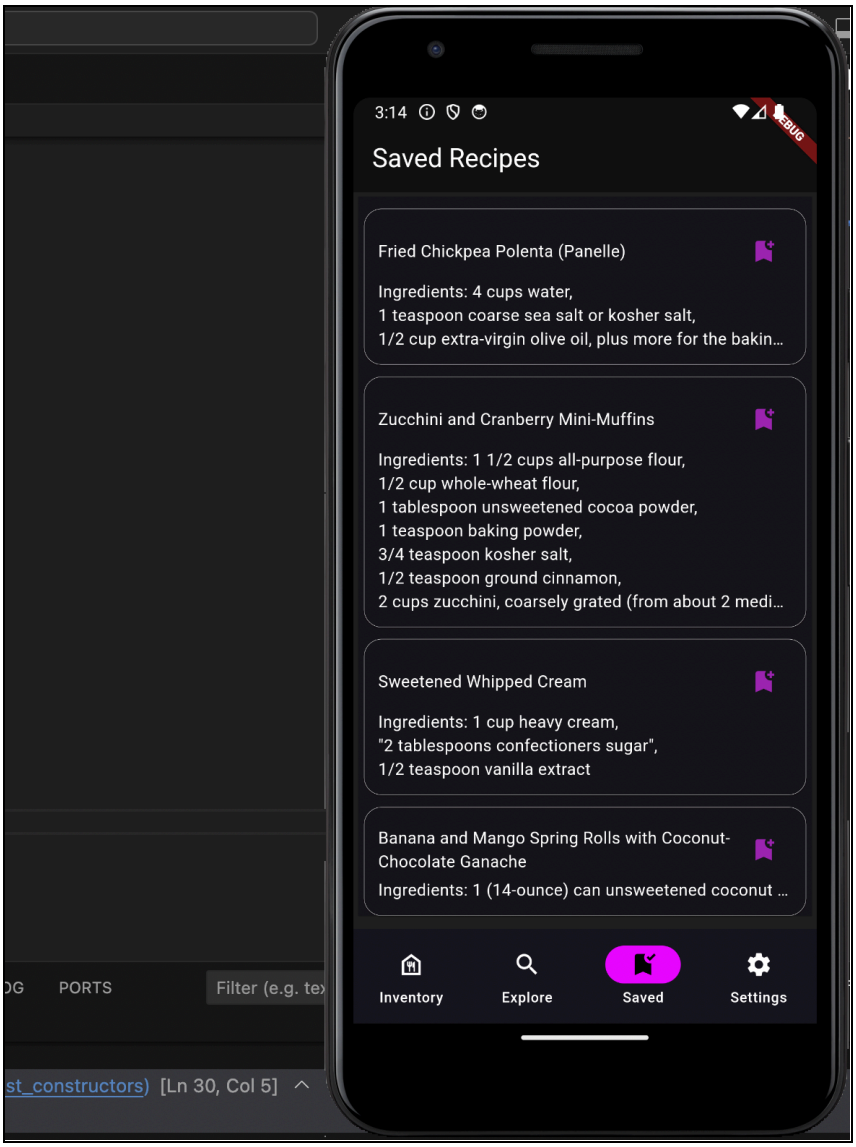


Figure 10.12. Saved Recipes Page Layout

The above screenshot shows the current layout for the Saved Recipes Page of the RecipeCart mobile app.

11.0 Administrative Content

This administrative section summarizes the final overall bill of materials and discusses budgeting practices, the project timeline throughout Senior Design I and II, and project roles and team dynamics.

11.1 Final Overall Bill of Materials and Budgeting

The below table summarizes the overall cost associated with developing the RecipeCart. The hardware design consists of the majority of the capital expenditures, while the software design is governed by a pay-as-you-go basis where the cost racks up depending on the compute usage and frequency.

Part Name	**Source**	**Quantity**	**Price**
Rubbermaid White 11.4-Quart Dishpan	Amazon.com	1	$12.95
Jetson Nano Developer Kit	Amazon.com	1	$149.00
Wireless-AC8265 NIC WiFi and Bluetooth 4.2 Module	Amazon.com	1	$24.00
SanDisk 64GB MicroSDXC with Memory Card Adapter	Amazon.com	1	$11.70
5V/4A DC Jetson Nano Power Jack	Amazon.com	1	$17.00
Raspberry Pi Camera Module 2	Amazon.com	1	$25.00
HX711 Load Cell Amplifier	Amazon.com	6	$12.00
GGQ Digital Kitchen Scale	Amazon.com	1	$17.00
Jump Wire Kit	Home	1	$0.00
Basic Soldering Kit	Home	1	$0.00
	Hardware Subtotal:		$268.65
AWS Elastic Compute Cloud (EC2)	Amazon Web Services	—	$74.65
AWS Virtual Private Cloud (VPC)	Amazon Web Services	—	$17.84
AWS Other Services	Amazon Web Services	—	$8.99

Software Subtotal:	$101.48
GRAND TOTAL:	**$370.13**

Table 11.1. Final Overall Bill of Materials

While we initially estimated the software costs to hover around $14-20 per month, the cost of maintaining and deploying an instance of the Weaviate vector database significantly escalated the costs associated with AWS. In fact the majority of the software-related expenses is attributed to running the EC2 instances and the VPC needed for the Weaviate database deployed through AWS Elastic Kubernetes Services. The software expenses reflect AWS usage for only the last two months of the project, when most AWS resources saw deployment. The expenses thus average approximately $70 per month, primarily to run and support the Weaviate database.

Provided with an existing and less costly vector database, the total software development cost of this project should total to no more than $15 per month. The software expenses were not expected to exceed $20 per month. Nonetheless, with the total cost of the project amounting to approximately $370, we satisfied our initial team budget of $400.

This project had no sponsors and funding came directly from the individual team members. The final cost of the project is equally split three ways among the team members.

11.2 Project Timeline and Work Distribution

The below table describes the timeline for the development of the RecipeCart. Much of Senior Design I was dedicated to planning, researching, and designing the RecipeCart. Although the initial goal was to spend approximately a week for each chapter of the report, certain chapters required additional research and consultation. Moreover, it was crucial to perform a thorough research into the software implementations and architectures to establish solid groundworks for the subsequent system design sections. Consequently, the latter half of the final report was drafted in the span of the last four weeks under vigorous time constraints.

To alleviate the workload in Senior Design II and provide as much flexibility as possible for the implementation phase, parts acquisition and prototype testing occurred throughout late November and early December.

Certain entries in the table below are color-coded to show the work distribution throughout Senior Design I. The same color codes are applied to tasks in Senior Design II to clarify the work distribution in the implementation phase of the project. Multicolored tasks are jointly handled by multiple team members.

Project Timeline	**Start Date**	**End Date**

Brainstorm, pitch ideas, and form groups	21-Aug-2023	27-Aug-2023
Flesh out ideas and do preliminary research	28-Aug-2023	03-Sep-2023
Invite ECE professors and faculty to join the senior design reviewer committee	01-Sep-2023	15-Sep-2023
Draft Chapter 2: Project Description Draft Chapter 11: Administrative Content	04-Sep-2023	15-Sep-2023
Design the website and upload the 10-page D&C	17-Sep-2023	06-Oct-2023
Meet with Dr. Wei and review the 10-page D&C	19-Sep-2023	22-Sep-2023
Draft Chapter 3: Hardware Parts Comparisons	25-Sep-2023	08-Oct-2023
Draft Chapter 4: Software Comparisons	02-Oct-2023	29-Oct-2023
Draft Chapter 5: Standards and Design Constraints Draft Chapter 6: ChatGPT Applications and Limitations	25-Oct-2023	03-Nov-2023
Finalize and upload the 60-page D&C to the website	04-Nov-2023	05-Nov-2023
Meet with Dr. Wei and review the 60-page D&C	06-Nov-2023	08-Nov-2023
Draft Chapter 7: Hardware Design Draft Chapter 8: Software Design	06-Nov-2023	26-Nov-2023
Draft Chapter 9: Implementation and Prototyping	13-Nov-2023	29-Nov-2023
Gather all hardware parts and perform quality testing	20-Nov-2023	09-Dec-2023
Draft Chapter 10: System Testing Draft Chapter 1: Executive Summary Draft Chapter 12: Project Conclusion Create end-to-end image forwarding prototype	25-Nov-2023	29-Nov-2023
Upload roughly completed draft to website for review Upload video demo for hardware components to website	29-Nov-2023	30-Nov-2023
Edit as necessary and submit the Senior Design I Report	01-Dec-2023	03-Dec-2023
Upload finalized Senior Design I Report to website	04-Dec-2023	05-Dec-2023
END OF SENIOR DESIGN I		
Acquire permissions for necessary software, including pre-trained ML models, APIs, and database access	06-Dec-2023	10-Dec-2023

Organize and further develop senior design website	13-Dec-2023	07-Jan-2024
Set up backend server via AWS Amplify Set up user, ingredient, and recipe databases Set up GitHub repository	15-Dec-2023	21-Dec-2023
Integrate Raspberry Pi Camera with Jetson Nano Load barcode detection software onto Jetson Nano Design image-based classification pipeline	21-Dec-2023	24-Dec-2024
Connect kitchen scale to HX711 Integrate HX711 with Jetson Nano Develop ingredient processing script	08-Jan-2024	14-Jan-2024
Assemble hardware architecture Integrate Meta DINOv2 with backend	15-Jan-2024	31-Jan-2024
Connect hardware architecture to backend	01-Feb-2024	17-Feb-2024
Create APIs for backend	18-Feb-2024	25-Feb-2024
Develop recipe recommendation system	26-Feb-2024	04-Mar-2024
Develop mobile frontend and revise backend	05-Mar-2024	14-Apr-2024
Debug and test system to comply with specifications	04-Apr-2024	14-Apr-2024
Finalize report, website, presentation, and demo	15-Apr-2024	22-Apr-2024
END OF SENIOR DESIGN II		

Table 11.2. Color-Coded Project Task Timeline

In the time period between Senior Design I and Senior Design II, we took early action familiarizing ourselves with AWS resources, Amplify, and the backend setup process. We also set up the user, ingredient, and recipe data tables and connected it to the Amplify backend. We also loaded the Jetson Nano with the barcode detection software to test its efficiency and compatibility with the Raspberry Pi Module 2 camera.

11.3 Additional Project Roles

The development of the RecipeCart is broken down into four phases: hardware architecture assembly, backend setup and configuration, mobile frontend, and machine learning. With our team's expertise primarily leaning on the software side and the RecipeCart's hardware architecture being relatively simple, we intend to assign exclusively two team members to hardware development. The work distribution and responsibilities for the software development are more intertwined due to the complexity of the project. In addition to the color-coded responsibilities in Table 11.2, we assign

Darren Ha with managing the project and ensuring that adequate progress is made in the development and testing of the RecipeCart.

12.0 Project Conclusion

Many existing smart inventory systems are implemented in different settings to help users conveniently track their food storage. One example of this is Cust2Cart, a smart-cart system that helps users keep track of what they buy in real time. Another example is the Samsung Smart Fridge, one part of the Samsung's smart house system that lets the user keep track of items they have in their household through phone or computer apps. Samsung also implements the Samsung Food application, which is a software that allows users to explore personalized recipe recommendations. All of these products have a main downside. Smart inventory systems, as convenient as they can be, are all very expensive. Inspired by this approach, we aim to find a cheap and effective alternative to combine the advantages of having smart inventories and the recipe system to create a product that allows us to get access to recipes straight from items detected in our inventory.

We attempt to address the weaknesses of existing zero-food-waste recipe recommenders by drawing on the demand for user-aligned tailored recipe generation instead of simple recipe selection. We introduce the RecipeCart, a mobile application with support for hardware ingredient detection. By focusing our efforts and choice of components toward minimizing the computational and physical load of our hardware and maximizing the scalability of our software, we can be confident that the RecipeCart can be a commercially viable melding between existing smart home or shopping technologies and AI-augmented ingredient-efficient recipe generation and recommendation.

While the RecipeCart presented in this work is minimal, we can envision a number of ways users and companies can leverage this technology:

Companies like Samsung can integrate this technology with their smart fridges to track ingredients actively in the fridge. Alternatively, the RecipeCart's hardware can be cheaply extended to tracking other ingredients that are not stored in the fridge such as the pantry without much additional cost.

Grocery stores can use the RecipeCart to encourage users to explore a wider variety of ingredients to incorporate into their diet, advertise food products, or inspire users to purchase additional ingredients by showing them appetizing recipe recommendations based on their current cart.

Additionally, the RecipeCart's hardware can be a viable method of automating checkout systems and hastening the pipeline for food purchase.

With these benefits in mind, we infer that companies are likely to be incentivized to purchase and integrate the RecipeCart into their services. We further speculate that this effect may alleviate the burden of purchasing physical hardware on users and encourage a larger user base, thereby generating sufficient data to further refine the program.

We aim to soft-launch the RecipeCart beta version at a local scale, with a maximum of 200 users concurrently to collect feedback. Even though the application aims at users that own smart inventory devices, due to the additional feature that allows users to manually add ingredients into their inventory, the app can be initialized with no hardware constraints. At the end of Senior Design II, the app is estimated to have gone through enough testing and optimization stages for a stable official launch. Our RecipeCart aims to handle at least 1000 concurrent users, and servers stay hosted for at least 1 year. For companies that express interest in the ideas of the RecipeCart, we note that some of the current components of the RecipeCart such as the databases are subject to non-commercial use only. Though efforts to change out these components are not likely to impede the overall program, the interested companies will need to provide for these replacements.

Acknowledgements

We would like to thank Dr. Enxia Zhang for her involvement in aiding the team in the later stages of the project's development. We would also like to thank Dr. Tanvir Ahmed and Professor Arup Guha for serving as reviewers for our project.

We would finally like to acknowledge Dr. Lei Wei for his assistance and understanding with the RecipeCart team. His constructive inputs provided the team with an insightful approach with handling certain engineering specifications.

Appendix A – References

Lenovo. (2022). Revolutionizing the shopping experience. *Lenovo.Com*. Lenovo. Retrieved September 15, 2023, from https://www.lenovo.com/content/dam/lenovo/dcg/global/en/customer-stories/case-study-cust2mate.pdf.

Samsung. (2023). *Samsung Family HubTM: Samsung Us: Undefined undefined*. Samsung us. Retrieved September 15, 2023, from https://www.samsung.com/us/explore/family-hub-refrigerator/overview/.

Slater, M., & Kreizman, J. (2022). (rep.). *A2Z Smart Technologies Initiation of Coverage*. Valore Research & Consulting. Retrieved September 15, 2023, from https://a2zas.com/wp-content/uploads/2022/06/AZ_Valore_Initiation_April_2022_ENG_DRAFT_compressed.pdf.

[2.4] https://samsungfood.com/about/#:~:text=The%20Whisk%20app%20officially%20rebranded,press%2C%20users%2C%20and%20creators.

[2.5] https://app.samsungfood.com/recipes

[3.1] https://www.techtarget.com/iotagenda/definition/microcontroller#:~:text=A%20microcontroller%20is%20a%20compact,peripherals%20on%20a%20single%20chip.

[3.2] https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf

[3.3] https://www.seeedstudio.com/blog/2019/10/24/microcontrollers-for-machine-learning-and-ai/

[3.4] https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[3.5] https://coral.ai/products/dev-board/#tech-specs

[3.6] https://developer.nvidia.com/embedded/jetson-nano

[3.7] https://forums.developer.nvidia.com/t/best-cameras-for-jetson/49494

[3.8] https://www.raspberrypi.com/documentation/accessories/camera.html#camera-module-3

[3.9] https://boredconsultant.com/2023/02/19/Raspi-Camera-Module-v3-vs-HQ-Camera/

[3.10] https://www.logitech.com/en-us/products/webcams/c920s-pro-hd-webcam.960-001257.html

[3.11] https://drive.google.com/file/d/10IgEGNXSWZNjBNJv240IYPmdfYQsQpE6/view

[3.12] https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all#installing-the-hx711-arduino-library-and-examples

[3.13] https://community.particle.io/t/strange-behavior-of-load-cell-hx711/49566/2

[3.14] https://www.digitalscalesblog.com/connecting-scale-raspberry-pi/

[3.15] https://laumas-us.com/product/tle-load-cell-amplifier/

[3.16] https://www.seeedstudio.com/ZKETECH-EBD-A20H-AC-Electronic-Load-Battery-Capacity-Tester-Powe-Supply-Tester-30V-20A-200W-p-4521.html

[3.17] https://www.findthisbest.com/best-lab-power-supplies

[3.18] https://www.digikey.com/en/products/detail/xp-power/VCS50US12/4488662

[4.1] https://www.dynamsoft.com/barcode-reader/docs/core/introduction/

[4.2] https://github.com/nrl-ai/daisykit

[4.3] https://www.themarketingtechnologist.co/building-a-recommendation-engine-for-geeksetting-up-the-prerequisites-13/

[4.4] https://neptune.ai/blog/understanding-few-shot-learning-in-computer-vision

[4.5] https://cloud.google.com/vision?hl=en

[4.6] https://aws.amazon.com/rekognition/

[4.7] https://webuters.medium.com/google-cloud-vision-vs-amazon-rekoginition-which-is-better-c7cb4780d82a

[4.8] https://ai.meta.com/blog/dino-v2-computer-vision-self-supervised-learning/

[4.9] https://developers.google.com/machine-learning/recommendation/content-based/basics#:

~:text=Content-based%20filtering%20uses%20item,previous%20actions%20or%20explicit%20feedback.

[4.10] https://developers.google.com/machine-learning/recommendation/collaborative/basics

[4.11] https://www.scientificamerican.com/article/how-recommendation-algorithms-work-and-why-they-may-miss-the-mark/#:~:text=Most%20recommendation%20algorithms%20now%20use,of%20release%20and%20other%20attributes.

[4.12] https://recipegpt.art

[4.13] https://krntneja.github.io/files/recipe-generation-paper-iccc23.pdf

[4.14] https://nestify.io/blog/exploring-digitalocean/

[4.15] https://docs.digitalocean.com/products/app-platform/how-to/create-apps/

[4.16] https://www.digitalocean.com/products/app-platform

[4.17] https://www.geeksforgeeks.org/firebase-introduction/

[4.18] https://blog.back4app.com/firebase/#Firebase_Advantages

[4.19] https://firebase.google.com/docs/functions

[4.20] https://blog.back4app.com/aws-amplify-vs-lambda/#:~:text=AWS%20Amplify%20allows%20developers%20to,intelligence%20and%20machine%20learning%20actions

[4.21] https://aws.amazon.com/amplify/

[4.22] https://learn.microsoft.com/en-us/azure/developer/mobile-apps/azure-mobile-apps/overview

[4.23] https://learn.microsoft.com/en-us/azure/developer/mobile-apps/serverless-compute

[4.24] https://azure.microsoft.com/en-us/products/category/databases

[4.25] https://www.c-metric.com/blog/cloud-pricing-comparison-of-aws-vs-azure-vs-google-cloud/#:~:text=From%20this%20comparison%2C%20it's%20very,is%20double%20the%20rest%20two.

[4.26] https://upcdatabase.org/api-pricing

[4.27] https://go-upc.com/plans/bulk-lookups

[4.28] https://www.upcitemdb.com/

[4.29] https://world.openfoodfacts.org/

[4.30] https://docs.pinecone.io/docs/overview

[4.31] https://weaviate.io/developers/weaviate

[4.32] https://blueclawdb.com/mysql/advantages-disadvantages-mysql/

[4.33] https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb

[4.34]https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin#:~:text=Xamarin%20support%20will%20end%20on,about%20upgrading%20Xamarin%20projects%20to%20.

[4.35]https://waverleysoftware.com/blog/why-use-flutter-pros-and-cons/#:~:text=Flutter%20allows%20developers%20to%20build,application%20development%20are%20much%20lower.

[4.36]https://reactnative.dev/docs/components-and-apis#android-components-and-apis

[4.37]https://pagepro.co/blog/react-native-pros-and-cons/

[5.1] https://ieeexplore.ieee.org/document/6783681

[5.2] https://www.pelco.com/blog/onvif-guide

[5.3] https://ieeexplore.ieee.org/document/10011140

[5.4] https://ieeexplore.ieee.org/document/9382202

[5.5] https://ieeexplore.ieee.org/document/9456823

[5.6] https://ieeexplore.ieee.org/document/9586768

[5.7] https://aws.amazon.com/what-is/osi-model/#:~:text=The%20Open%20Systems%20Interconnection%20(OSI)%20model%20was%20developed%20by%20the,IEC%207498%2D1%3A1994.

[5.8] https://www.ieee802.org/

[5.9] https://ieeexplore.ieee.org/document/6522432

[5.10] https://www.mipi.org/specifications/csi-2

[5.11] https://www.mipi.org/specifications/camera-command-set

[5.12] https://www.mipi.org/specifications/mipi-cse

[5.13] https://ieeexplore.ieee.org/document/9082285

[5.14] https://ieeexplore.ieee.org/document/9968219

[5.15] https://ieeexplore.ieee.org/document/7076554

[5.16] https://www.iso.org/standard/63712.html

[5.17] https://www.softkraft.co/software-development-standards/#iso-12207-software-life-cycle-processes

[5.18] https://www.iso.org/standard/79428.html

[5.19] https://www.iso.org/standard/64901.html

[5.20] https://www.opslevel.com/resources/standards-in-software-development-and-9-best-practices

[5.21] https://www.redhat.com/en/topics/devops/what-is-ci-cd

[5.22] https://asana.com/resources/agile-methodology'

[5.23] https://ieeexplore.ieee.org/document/9062658

[5.24] https://ieeexplore.ieee.org/document/10278119

[6.1] https://medium.com/@rushdauwaiz/exploration-of-chatgpt-and-the-future-of-generation-ai-1-1767fef6e137

[6.2] https://eliiza.com.au/wp-content/uploads/2023/03/ChatGPT-decoded.pdf

[6.3] https://lifearchitect.ai/chatgpt/

[6.4] https://www.nature.com/articles/s41746-023-00819-6

[6.5] https://www.thepromptmarket.ai/prompt-engineering-info/

[6.6] https://openai.com/gpt-4

[6.7] https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/

[6.8] https://blog.google/technology/ai/bard-google-ai-search-updates/

[6.9] https://ai.meta.com/blog/large-language-model-llama-meta-ai/

[6.10] https://voyager.minedojo.org

[7.1]

https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/SP-09732-001_v1.1.pdf?9hrfJFagMDDd1xM4VWp-zeUQyafc0sF3xS8wUndjQcPDdXB4M_qKH_drB8GCy0XJOgVWcXJE2lB1xaDjpmhO-n96CevdCPa-IfmFwQXtzJ9Z38edJqIHLwtLQ6VP4sHv3gkbnru3DG99qeT0n7YMrjz1a2YWPcqk-Prm7w==&t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9

[7.2]
https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/NV_Jetson_Nano_Developer_Kit_User_Guide.pdf?NIkhvD-_qGC7W_v3vPtsWcx5OQb6qa-HbWKmh_a_BgiFQGASX762fv6oJfwcvO0t1JNVTsD9WAu5EmKt1PDWCiFrq2hySa-6KAJLjF7gvkbcoraDa96pgUGBgxPjYM1RVbPuHdbB1BbOx4rAMjAgqE5DGKc7SKGrQzoNKpzjX02a_TJOLOQmWapeZn_4c7jKDdo=&t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9

[7.3]
https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano_DataSheet_DS09366001v1.1.pdf?vGbRB6Z_iQG5tH3AC1EKF0EgXyplrDIhUx04XLblbHXcBBFYNoKf8eVXCs4l_YCX3YVUob9fr6zmm9lGUybmASXXIyz-G8e8a34IRN9RQb3ZZvcAOZDZ05cLAXXMYdBJvTegZVkhGQ-E6-eZhKpUnSLPboteWmJfw5U_HotWxZnYWh2qT2VwW8UDFbQqjg==&t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9

[7.4]
https://datasheets.raspberrypi.com/camera/camera-module-2-mechanical-drawing.pdf

[7.5] https://cdn.sparkfun.com/assets/f/5/5/b/c/SparkFun_HX711_Load_Cell.pdf

[7.6] https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[8.1] https://computationalcreativity.net/iccc23/papers/ICCC-2023_paper_46.pdf

[8.2] https://www.kaggle.com/code/dskagglemt/grocery-dataset-product-image/input

[9.1] https://youtu.be/art0-99fFa8

[9.2] https://github.com/kempei/hx711py-jetsonnano

[9.3] https://youtu.be/LIuf2egMioA

[9.4] https://docs.amplify.aws/flutter/tools/cli/project/override-iam/

[9.5] https://aws.amazon.com/getting-started/hands-on/build-flutter-mobile-app-part-one/

Appendix B – Additional Resources

Hardware Implementation

Weight Scale System

Guides to Amplify Digital Scale Data with HX711 Load Cell Amplifier
https://www.youtube.com/watch?v=O9AiMON1420

https://www.youtube.com/watch?v=sxzoAGf1kOo

Python Library to Interface Jetson Nano with HX711
https://github.com/kempei/hx711py-jetsonnano/blob/master/hx711.py

Camera Integration

Raspberry Pi Camera Software Documentation
https://www.raspberrypi.com/documentation/computers/camera_software.html

Python Library to Interface Jetson Nano with CSI Camera
https://github.com/JetsonHacksNano/CSI-Camera

Jetson Nano Network Connectivity

Guide to Setup WiFi and Bluetooth on Jetson Nano
https://jetsonhacks.com/2019/04/08/jetson-nano-intel-wifi-and-bluetooth/

Software Implementation

AWS Amplify Guides and Documentation

AWS Amplify with Flutter
https://docs.amplify.aws/flutter/

Guide to Building a Flutter app with Amplify
https://aws.amazon.com/getting-started/hands-on/build-flutter-mobile-app-part-one/

https://aws.amazon.com/getting-started/hands-on/build-flutter-mobile-app-part-two/

Guide to Using Flutter Riverpod to expose API endpoints

https://github.com/bizz84/flutter-tips-and-tricks/blob/main/tips/0046-riverpod-difference-between-ref-watch-ref-read-ref-listen/index.md

Ingredient Detection Software

Pyzbar Documentation

https://pypi.org/project/pyzbar/

Meta DINOv2 Documentation

https://dinov2.metademolab.com

MQTT Client on Amplify Flutter

https://youtu.be/aY7i0xnQW54

https://github.com/0015/ThatProject/blob/master/ESP32_MQTT/2_Flutter_MQTT_Client_App/mqtt_esp32cam_viewer_full_version/lib/main.dart

https://github.com/shamblett/mqtt5_client/blob/master/example/mqtt5_server_client_secure.dart

AWS Lambda Resources

Publicly Available Lambda Layers

https://api.klayers.cloud/api/v2/p3.10/layers/latest/us-east-2/html

Datasets

Recipe 1M+ Dataset

http://pic2recipe.csail.mit.edu/

Kaggle Recipe 1M+ Data Subset

https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images/

World Open Food Facts Database

https://world.openfoodfacts.org/

Weight System Integration Resources

HX711 Python Library for Jetson Nano

https://github.com/kempei/hx711py-jetsonnano/blob/master/hx711.py

Wiring the HX711 to a Digital Scale

https://youtu.be/LIuf2egMioA

Recommendation System

https://github.com/recommenders-team/recommenders

Recipe Generation System

https://github.com/LARC-CMU-SMU/RecipeGPT-exp

Mobile Frontend Documentation

Figmas

https://www.figma.com/file/j5WkYkRPcjGrjh1nVyyQMI/Bryan-Ha's-team-library?type=design&node-id=0%3A1&mode=design&t=iFyBr295TpqymG9o-1

Use Case Diagrams

https://www.figma.com/file/0w96FYz8Fxrpe7IDH5k8Ag/Use-Case-Diagram-For-Mobile-InMart-(Community)?type=whiteboard&node-id=0%3A1&t=bi5moetRevvkyw1e-1

Senior Design Website

(You made need a VPN to connect to the UCF WiFi to view the website)

maverick.eecs.ucf.edu/seniordesign/fa2023sp2024/g35/

GitHub Repository

https://github.com/RecipeCart/RecipeCart