UCSB
CS190B - F23

Final Project

Team Members:

First and last names	**Perm Number**
Jai Uparkar	3932183
Kelly Yan	3971017
Brandon Lee	8753261

Table of Contents:

Report

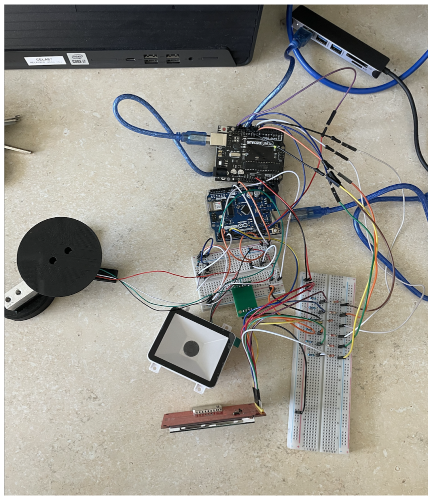
Problem Overview

As college students, maintaining a healthy lifestyle is challenging since many of us are busy with classes, social activities, lack nutritional awareness, and the process of tracking daily caloric and nutritional intake is often too tedious to track. As a result, we find ourselves, like many college students making suboptimal decisions about meals leading to health issues and nutritional deficiency. Current methods of tracking nutrition include apps like MyFitnessPal which recently have included a paywall and other methods which include manual entry. These results are often unreliable and are error prone.

To solve this problem, we developed the NutriScanner! Our solution is a user friendly nutrition scanner involving a weight scale, barcode reader, and an LED display. Our solution instantly provides the user precise nutritional information for the exact amount of food they are consuming. By using API calls to gather nutritional information from the barcode scanner, we are one step closer to solving an important problem plaguing college students and empowering them to make healthier choices.

The importance of our project is demonstrated by several key aspects: precision, convenience, educational, and customization.

Precision: The weight scale and scanner work together to provide precise information regarding calories and macros regarding a certain food item. This rules out the guessworking connected to portion sizes and nutritional value, especially protein and carbohydrates.
Convenience: Our solution streamlines the process of googling an item's nutritional and scrolling through several pages to by providing instantaneous feedback. The convenience of this solution allows them to make faster and educated decisions regularly.
Educational: By providing accurate nutritional information, users can make informed decisions about their health and fitness goals. Having accurate and instantaneous information gives individuals the power to make healthier choices and learn more about their food choices.



In short, our smart food scale with a built-in barcode scanner interfaces with an LCD screen to log food items without having to use a phone. The steps for using this smart scaler involves scanning an item, weighting it, and confirming the measurement which automatically logs the nutritional facts on the screen.

Methods & Solution

I. Hardware Setup

The barcode scanner is implemented with the Mairkt Embedded QR Code Barcode Scanning Module. This scanner uses UART to communicate, we found that this module has a USB and TTL interface, allowing it to interact with a computer and a microcontroller. Reviews and articles on this module indicate that it has the best sensitivity and highest accuracy. Specifically this module is capable of reading 1D/2D codes on various surfaces and uses image recognition algorithms to identify and read the codes. The module also has a 4-wire terminal for interfacing with the Arduino so configuring the hardware was simple. We used the Software Serial library to read the data from the food label.

To display is implemented with the TFT IlI9341 LCD Display, which uses the SPI communication protocol. The LCD display is required to communicate the nutritional information about their food to the users. An LCD Display uses liquid crystals to manipulate light, allowing it to display graphical elements. In order to display the output from the barcode scanner, we used the Adafruit ILI934 library. We decided to use this display because it is compatible with the architecture on all Arduino boards. This made testing and debugging our project easier since we had 2 different Arduinos during development. The library allows us to easily display graphical elements with customization of the text size, brightness, and color. The library's documentation and online resources proved helpful when we experimented with functions to optimize for user experience.

The weight scale uses the HX711 ADC Module Weighing Sensor with a load cell. All digital weight scales use a load cell which converts force to a measurable voltage that users can interpret. The widespread adoption of load cells is due to its accurate readings. The HX711 amplifier allows us to read the load cell and interface with the Arduino, which performs an A2D conversion that can be processed by software. The header pins needed to be soldered to connect to the Arduino. To read the measurements from the load cell, we needed to use the HX711_ADC library. Before we started, we needed to calibrate the load cell using the library's setCalFactor function. The library's documentation allowed us to remedy the fluctuations in readings.

Two Arduinos were used for our project. The first board was an arduino uno wifi rev2, and this handled communication with our flask server, barcode scanner, and load cell. The second board was an arduino uno, and this interfaced with our LCD display. The board also received data to be shown on the display from the first arduino. While this wasn't the initial solution we intended for, we chose this route because the wifi arduino board had no library support for our LCD display; we tried multiple LCD displays, all of which claimed could be interfaced with any arduino board, but did not work on the wifi arduino board. Our solution to this was to forward text data from the wifi arduino board to the receiver board (arduino uno) via I2C.

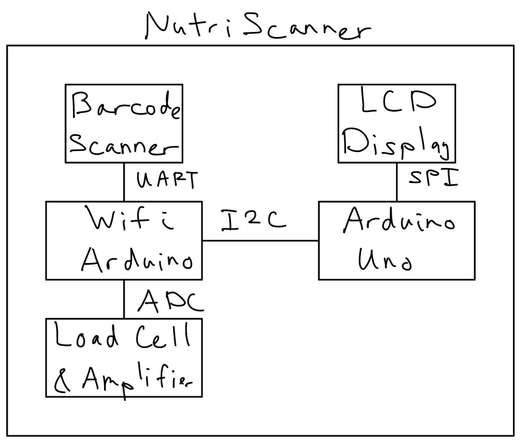


Figure 1: Block Diagram of NutriScanner Device.

II. Software

On the arduino side, software included code for peripheral communication, wifi connection, and flask server interaction. On the server side, software included code for setting up the server and handling the api request from our arduino. We used arduino IDE (C-based) to program our hardware, and python to implement the flask server. Relevant arduino libraries include driver code for the HX711, LCD display, UART communication, and I2C communication.

Let's define the arduino wifi rev2 as the master board and the arduino uno as the receiver board. For the master board, code consisted of driver code for the barcode scanner and load cell/amplifier, functions to connect to WiFi and our flask server (e.g. make port connection, make api call), and a finite state machine to correctly respond to inputs from the user. For the receiver board, code consisted of driver code for the LCD display, functions to draw text to the display, and setup code to receive data from the arduino wifi board. Note that we forwarded data (e.g. display text, nutrition facts, etc.) from the master board to the receiver board via I2C.

The finite state machine (FSM) consists of three states — scan, weigh, and display — and was implemented using switch case statements. The FSM initially starts at the scan state, and asks the user to scan an item via the lcd display. The FSM transitions to the weight state once a barcode is read (i.e. the barcode buffer contains data and is read from the arduino). Additionally, we used UART to handle data communication between the barcode and the arduino. Once the FSM is in the weigh state, the device asks the user to weigh an item and press the confirm button on the device. When in this state, the user can weigh items, and we also implemented a tare functionality (resetting the weight to zero when an object is on the scale). However, pressing the tare button did not change the current state of the FSM. When the confirm button is pressed, the FSM transitions to the display state, which calls our api (we used open food facts) and displays the calculated nutrition for that food item and weight. We did this by calculating the ratio between the recorded weight and the serving size (which we got from our api) and multiplying this number by the food item's calories, fat, carbohydrates, and protein information from the api (the api's nutrition facts are based on the serving size). We then set a delay before transitioning back to the scan state.

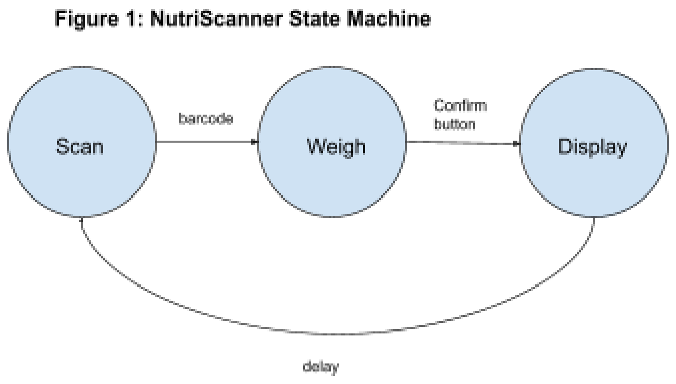


Figure 2: FSM bubble chart for NutriScanner Device

V. IoT

To call our API, we used a flask server as a middleman. The arduino would make a HTTP request to our flask server with a specified endpoint. That specified endpoint would then call the Food Facts API and would retrieve a JSON object from the API. The flask server would then parse the JSON into a shorter response for the arduino (as initially our arduino couldn't handle the original JSON response). Finally, the flask server would return the parsed JSON object to the Arduino.

While this solution may seem convoluted, we ran into issues calling the API directly from our Arduino. The main issue revolved around the fact that our API used an HTTPS endpoint. Since Arduino's are resource-constrained, we couldn't handle HTTPS endpoints from our Arduino; we could only handle HTTP requests. The only way to bypass this was to buy a more powerful WiFi module, but due to the interest of money and time, we decided to not buy an external WiFi module and instead went with using a flask server to handle the API.

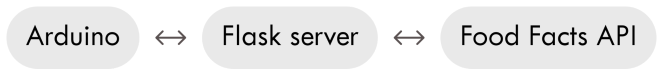


Figure 3: Flow diagram between Arduino, flask server, and Food Facts API

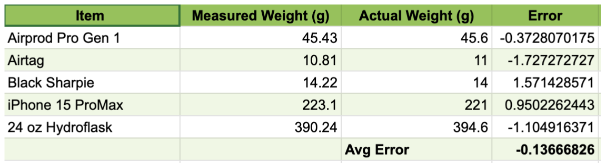
Results & Findings

I. NutriScanner Accuracy

A central objective of the NutroScanner is to deliver reliable and accurate measurements. To ensure this goal, we performed a series of tests to evaluate the accuracy of the scale. For the sake of testing, we used common items with a known manufacturing weight to validate our load cell readings. The results were fairly accurate with fluctuations in weight due to some calibration oversights and faulty connections. We superglued the 3D components to the load cell before measuring the weight so our calibration weight was discovered with some trial and error. The wires for the load cell are also fragile and lack pins making the connection to the breadboard loose at times.

We performed a comparative analysis between the object's known weight and load cell reading. We then calculated the average error of all the observations and determined them to be within a relatively insignificant margin. In addition, we evaluated the precision of the weight scale reading my weighing the same item multiple times to see how much results fluctuated. We calculated the average and standard deviation to further evaluate the scale's reliability.

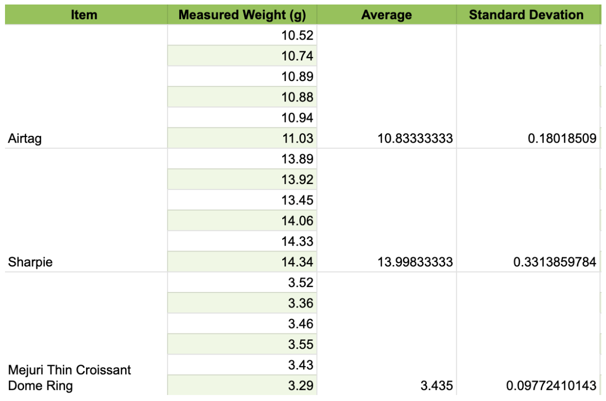
Table 1: Weight Scale Accuracy and Error



Item	Measured Weight (g)	Actual Weight (g)	Error
Airprod Pro Gen 1	45.43	45.6	-0.3728070175
Airtag	10.81	11	-1.727272727
Black Sharpie	14.22	14	1.571428571
iPhone 15 ProMax	223.1	221	0.9502262443
24 oz Hydroflask	390.24	394.6	-1.104916371
		Avg Error	**-0.13666826**

Caption: Each item was weighed once on the load cell and compared to the known weight.

Table 2: Weight Scale Precision



Item	Measured Weight (g)	Average	Standard Devation
	10.52		
	10.74		
	10.89		
	10.88		
	10.94		
Airtag	11.03	10.83333333	0.18018509
	13.89		
	13.92		
	13.45		
	14.06		
	14.33		
Sharpie	14.34	13.99833333	0.3313859784
	3.52		
	3.36		
	3.46		
	3.55		
	3.43		
Mejuri Thin Croissant Dome Ring	3.29	3.435	0.09772410143

Caption: Each item was weighed 6 times to calculate the precision of the load cell and how reliable the readings were.

Our figures and experimentation reveals the reliability and quality of our product's ability to correctly measure ingredients for nutritional purposes. While the load cell readings are not always consistent, it provides enough accuracy and similarity in the context of the problem.

II. Barcode Scanner Recognition

Our project's functionality highly depends on the barcode scanner's ability to identify and read codes. Although we didn't design the image recognition logic, we still wanted to evaluate the scanner's accuracy and demonstrate the input and output functionality. The barcode scanner worked with 100 % accuracy since it correctly identified all the digits. All in all the Mairkt Embedded QR Code Barcode Scanning Module was very easy to use and performed as expected.

Table 3: Barcode Scanner Input and Output Response

Food Item	Scanner Output	Correct?
Trader Joe's Trader Giotto's Balsamic Glaze	009789270000	Y
Trader Joe's Pumpkin Bread and Muffin Mix	767872611092	Y
Trader Joe's Everything But The Bagel	653476917305	Y

Trader Joe's Traditional Tunisian Harissa	703255363707	Y

III. Arduino Rev 2 Wifi Shortcomings
Not only was the wifi module on this board not powerful enough to handle our API (which was called using an HTTPS endpoint), but also it had no support for our LCD display library. Furthermore, when our initial LCD display could not interface with the Rev 2, we tried another LCD display with another library to see if another driver would fix it. However, both LCD display libraries refused to work with the Rev 2. We noticed on several Arduino forums that many users have experienced similar library support issues for the Rev 2, so we instead hooked up an Arduino Uno Rev3 to our system, which would interface with our display. As mentioned in the Software section, we sent LCD display data from our Rev 2 to our Rev 3 via I2C.

IV. User Feedback
Although we didn't have time to perform a rigorous and complete user test, we wanted to understand the user experience. The most obvious and significant feedback we received was that the user flow wasn't intuitive. There are no labels on the buttons indicating which are for tarring and confirming. Also the configuration of our board forces the user to maneuver around multiple wires to press the button. Unless instructed, the flow of use wasn't clear which reveals some lack of intuitive and intentional design on our part. As a proof of concept, our project archives the overall goal of tracking nutrition but lacks sufficient design for user experience.

Challenges

We ran into several challenges when completing this problem, with a majority of the problems being the physical components and our API call. Since our project had many moving parts, it took a lot of time and research to identify what barcode scanner, load cell, and LED screen to buy. Shipping these items took longer than expected which delayed the software development process. Additionally, since two project members didn't have much experience with hardware, there was a learning curve in configuring the breadboard and the wiring of the components.

I. Weight Scale
The first major challenge we ran into was creating our weight scale with our load cell. Without a plate and a stand, the load cell will not have enough space to compress and thus make an accurate reading. As a result, we couldn't continue our implementation process on the softwareside because of faulty readings and object's size limitations. As a result, we decided to 3D print the plate and stand to make the weight scale. The YouTube tutorial we were following to implement the weight scale included the STL files for both components. We then converted the STL files into GCode to print the parts. Once we printed the parts we discovered that the screw holes for the components and the weight scale did not match so we needed to super glue them together.

II. Load Cell Amplifier

The next hardware issue we ran into was with the width of the amplifier for the load cell. The amplifier just had holes in them and there wasn't any room for the wires on the breadboard to be connected to it. Since we needed both components connected to have a functional weight scale, we needed to solder male header pins into the amplifier. Since no one had experience in this, Brandon took on the responsibility to complete this with materials from his Capstone lab. This process took a while since we needed someone with expertise to explain this to us and materials to perform it. In addition, the load cell wires were extremely thin, making it difficult to have a secure connection with the Arduino.

III. API Call
The next issue we ran into was the API call. Upon scanning the barcode, we had to make an API request (with an HTTPS endpoint) to get the nutritional information about the food item. However, as mentioned before, working with an HTTPS-based API through Arduino is difficult because it is resource constrained; the Rev 2's built-in wifi module was not powerful enough to handle our API request. As a result, this limited the number of APIs we could work with (could only work with API's with HTTP endpoints).

To remedy this, we first tried to see if the Open Food Facts API, which uses an HTTPS endpoint, could be called using an HTTP endpoint. We used Postman to test this theory out, and it seemed that we could make an HTTP call to our API. However, when we tried making the API call on our arduino, we received a 301 status code, which indicates a redirect to another URL. In other words, the arduino thought that we were providing an incorrect URL, and outputted a redirect URL in the headers. However, this redirect URL was an HTTPS endpoint. This did not make sense to us as we could make HTTP requests to our API on Postman, but not on the Arduino. We went to multiple office hours to troubleshoot this issue, but we still weren't able to find a solution. The TA's and us theorized that Postman does additional processing with HTTP requests, whereas the Arduino doesn't. This would explain the difference in responses and status codes between Postman and the Arduino.

Knowing that it was infeasible to directly call our API from our Arduino, we set up an HTTP endpoint on a flask server that the Arduino would call. When the Arduino calls the flask server, the flask server would then call our Open Food Facts API, retrieve the response body, parse the response into relevant information we needed, and return that back to the arduino. This allowed us to use our API by calling an HTTP endpoint.

IV. Tare Button Functionality
We ran into an issue with calibrating and implementing the tare button functionality for the weight scale. The weight readings on the load cell were highly fluctuating and not tarring properly. The lack of documentation and specific libraries required halted our progress for a while but with time we were able to troubleshoot this issue. The fluctuations in readings were likely caused by the sensitivity of the load cell.

IV. Arduino Wifi Rev 2 vs Arduino Uno Rev 3
We developed our software functionality on different Arduinos efficiency since there were many moving parts. After each component was functioning separately, we configured everything to the same Arduino. To our dismay we discovered that the driver for the display doesn't work for the other Arduino. This

created a challenge for us as we moved to the final part of the project. As mentioned previously, the Arduino Wifi Rev 2's built-in wifi module is not strong enough to handle an API request and has issues with connecting to certain LCDs.