18-500 Final Project Report: 12/14/2021

TracKat

Authors: MeeDm Bossard, Tarush Govil, Lucas Moiseyev

Electrical and Computer Engineering, Carnegie Mellon University

Abstract**—**TracKat is a novel pet tracking and health diagnostic system. The project utilizes a modular system of custom-made Bluetooth Low Energy (BLE) sensor tags to monitor pet house cat eating habits and transmit the collected data to a central hub device running a local web application. The web app stores and dynamically plots food and water intake over time, thus enabling pet owners to track the information. If the system catches any sudden changes in routine, it alerts the user with an email explaining which cat had what issue and recommends a check up with a veterinarian. TracKat is sure to be an excellent tool for cat enthusiasts and vets alike.

Index Terms**—**Bluetooth Low Energy (BLE), Cats, Load Cell, Printed Circuit Board (PCB), Radio-Frequency Identification (RFID), Raspberry Pi (RPI), Web Application

I. INTRODUCTION

Unlike many other pets, house cats instinctively hide signs of weakness or sickness, and can follow strange behavior routines leaving their owners confused as to how to best care for them. One tell-tale sign that a cat is sick or injured is a change in its normal eating habits, but such behavior patterns can be difficult for owners to track manually. The problem is further confounded if a household has multiple cats that eat freely throughout the day, as it is extremely difficult to determine their individual food and water consumption.

TracKat enables cat owners to track and maintain their cat's health by sensing sudden changes in food and water consumption levels per cat. The system features a modular network of sensor tags communicating with a small hub computer. For the device to adequately inform users of their cats' health, it is built to accurately detect and display cat feeding and drinking within a maximum window of 10% false positives and 10% false negatives per day. To enable food bowl placement anywhere within a user's home (unrestricted by wall outlets), the sensor tags are designed to run off local battery power. To ensure the system is user friendly, the tags are designed with strict power rationing to last nearly 8 weeks on a single coin cell battery charge.

While there are devices on the market that perform similar actions, there is a distinct lack of devices that allow users to monitor their cat's eating and drinking habits over time. For instance, there are devices that first detect a cat's microchip number and then allow specific cats to get to their food. On the flip side, there are autonomous feeders which dispense food on a regular schedule. Both of these devices do not allow for the user to be able to track eating and drinking for multiple cats in a user friendly way.

II. DESIGN REQUIREMENTS

Establishing specific design requirements helped to ensure that TracKat would be able to successfully track fast changes and indicate cat health in multi-cat households.

The most important requirement for TracKat is its ability to **accurately detect a specific cat's eating and drinking habits**. The system should not only track these habits, but also be able to catch fast changes in consumption. False positives within 10% translate to having roughly 1 "extra" value for every 10 real times a cat goes to eat or drink. Likewise, false negatives within 10% mean the system can, on average, fail to catch up to one eating or drinking event per 10 actual events. This way the owner can still get a great sense of their cat's health.

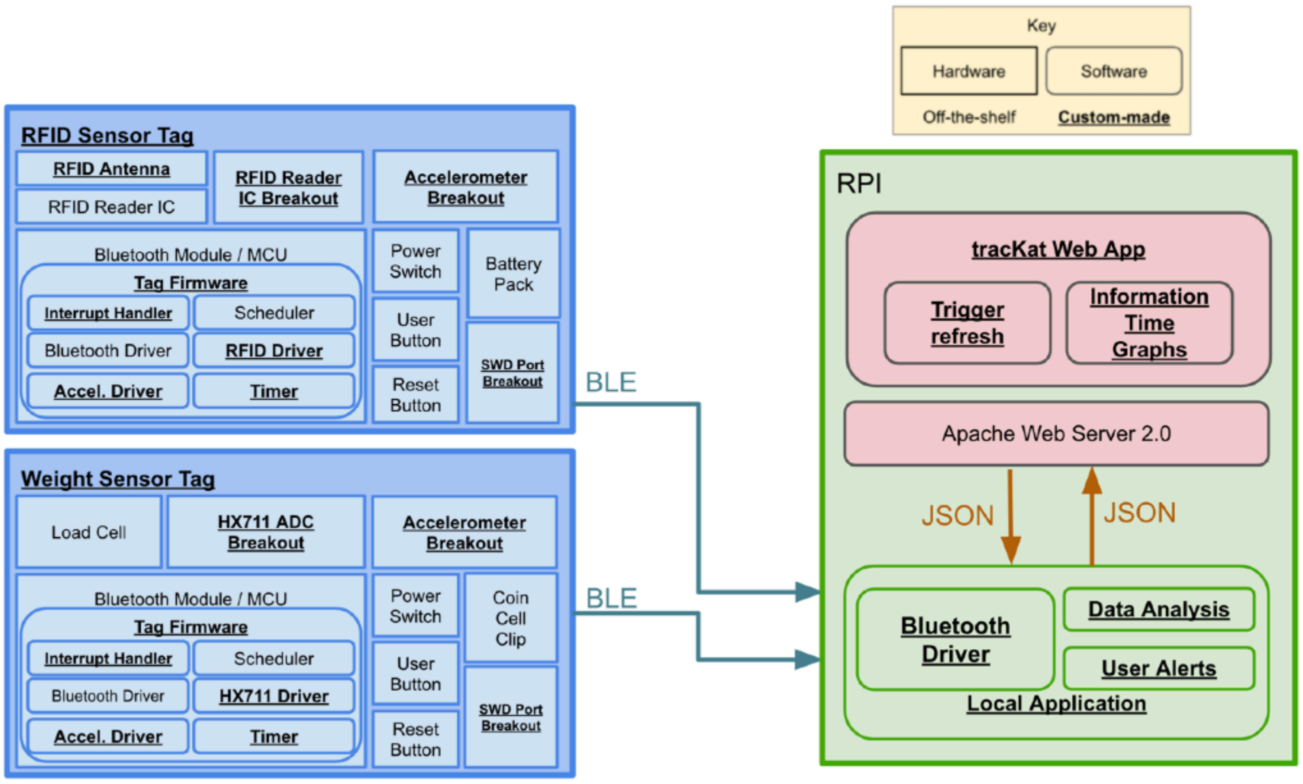
Another main aspect of the project is its ability to support **modular configurability**. As previously mentioned, other devices on the market are not suitable for multi-cat households when it comes to regulating food intake. TracKat must be configurable by the user for different numbers of cats and bowls. This feature is tested by varying configurations and verifying that the system works for all cases.

In order to allow the most accurate information as possible about the cat's consumption, the **weight sensing granularity** must be able to detect the smallest size that a cat could eat. This would be one kernel, which weighs roughly 0.1 grams. Hence the weight sensing granularity must be around 0.1 grams at the least to account for the minimum weight the cat could consume. An average lick of water for a cat is 3/100 of a teaspoon[1]. At 5 grams per teaspoon, this translates to 0.15 grams of water. Thus the 0.1 gram requirement is sufficient to also account for minimum water intake. To test this goal, one kernel is taken out of a test food bowl, and the system is checked for an accurate response.

Changing out batteries frequently can get tiresome and redundant for a user. Hence, **maximizing the battery life** of the components of the device is an important factor to consider in easing the use of the system. This project considers replacing batteries at a rate of once every 8 weeks as fairly reasonable for most electronic devices. This design requirement is tested by running the device through Dialog Systems' Power Profiling tool.

Lastly, being able to refer back to the cat's information for an extended period of time can be extremely valuable for the user to understand their cat's long term consumption routines. However, any longer than a year as a period of **time to display the data** won't be as useful for the owner, since fast changes in their habits are more likely to be of interest. Hence the data should be available to the user for at least 1 year. This design goal is tested by calculating the memory consumed by each data point, multiplying this by the absolute extreme maximum number of possible data points in an entire year, and then comparing this result against the available storage size of the central hub.

III. Architecture and/or Principle of Operation



1. System Diagram - RFID would eventually be scrapped to descope

A. Primary Components

1) Sensor Tags

The sensor tags are TracKat's connection to the physical world. Simplicity, modularity, and ruggedness are key factors that drive the design of the electronics, firmware, and mechanics.

The tags feature custom-designed PCB's built around the DA14531 - a bare-bones BLE module that supports the computing and peripheral capabilities required by the tags without being bloated with unnecessary extra features. The tags also include a coin cell battery clip, a power switch, a general purpose user button, a reset button, a six pronged pogo pin SWD port, a 10 pin adapter for the SWD port, and GPIO breakouts. This main PCB is designed to interact with custom made sensor breakout boards - specifically a load cell ADC/ Amplifier chip breakout board, an accelerometer board, and an RFID chip board.

The sensor tag firmware drives weight sensing capabilities by running a system timer that fire interrupts at constant intervals. The interrupts are handled with logic that automatically establishes a baseline empty bowl reading upon being connected to the hub computer, filters out invalid sensor readings (such as a cat pressing into the bowl as it eats), and saves timestamps of each eating event. The tag also automatically senses food refills meaning that users do not have to manually notify the system that they refilled a bowl.

The tag electronics are housed inside a custom designed, 3D printed enclosure that comes in two pieces. The bottom plate includes mounting brackets offset from the ground to keep the electronics up and away from a potentially messy floor. The top plate features a skirt that pushes any falling water or food debris out and away from the electronics, and a lip that securely holds onto a standard sized metal cat food/water bowl.

2) Central Hub

The Hub (an RPI 4 computer) serves as the connection between the physical sensors and the digital web app. It runs a process that updates all of the cat's information every 4 hours.

It first connects to the tags that the user indicates they would like to use via the provided bluetooth address. This address value is given by the web app at time of registration. It connects to the specified tags via bluetooth, then reads in characteristics, which are memory mapped io arrays or values. These arrays and values are little endian bytes, which are then converted into integer arrays and values. Of these arrays and values there are, an array of weights that are not in grams, an array of timestamps in seconds, a baseline value for conversion purposes, a microchip number array, and the number of points being sent. The hub then uses the baseline value sent over to convert the array of weights into grams. Additionally, the microchip array is then used to sort through the weight values, and separates each of the weight values and time values according to the microchip number, meaning you then know which cat has which value of weight and time stamp.

It then does computation on these arrays to get some more usable information. Using these arrays, it does calculations to determine the consumption. It does this by subtracting adjacent weight values. Additionally it finds the same information in terms of volume. While the volume was not implemented in the web app in the end, they are still internally there. All of these values are then stored in the user's dedicated folder, inside of a .json file, where the filename is the cat.

After this process has been completed for all the users, effectively reconnecting to the tags in their house, reading in values, then organizing them, analysis is performed. The hub performs statistical analysis on the feeding data by calculating a running average of earlier eating and drinking events. If a recent weight value falls outside of above two standard deviations of the mean, the hub alerts the user by sending an email warning them of potential health issues. To keep track of eating or drinking too little, the same analysis is done on the time array. This is done because there is thresholding of the values done in the tags that does not enable the hub to use the consumption arrays to determine whether cats have only eaten a little. Therefore tracking how much time has passed in between eating or drinking times can tell more about how little a cat may have consumed.

3) Web App

The Web Application, deployed on the Raspberry Pi (http://172.26.161.0), is the interface between the TracKat system and the user. It features a registration page for new users to enter their contact information, add new cats to track, and select what the system is tracking (food, water, or both). Existing users can login into their accounts once they have fully registered.

They are then redirected to their custom profile page, from which they can access unique pages for each of the cats they had registered. On the cat's profile page, they can view the graphs of each cat they have entered into the system. They are able to view either the food graph, water graph, or both, depending on the option they chose at the time of registration. The graphs displayed changes dynamically based on the system configuration. Users can also click the "refresh" button on their profile page to manually update the graphs with the latest collected data from the sensor tags instead of waiting for the pre-scheduled automatic updates (once every 4 hours). Along with updating the graphs with the latest information, we perform data analysis on the food consumption and water consumption levels of the cat and determine if an unusual event has occurred, based off of which we send a custom message to the user's email address informing them about the issue.

The Web Application, along with storing data in its SQlite database, writes data to a global JSON file to communicate with the other software being written on the hub. This is done at the time of registration and examples of data stored include the user's email address (stored to inform the user of unusual eating/drinking activity), each cats' microchip number (for uniquely identifying which cat comes to the bowl), and the user's username (for correctly identifying the user for the cat, which is especially essential when a user may have multiple cats). In order to ensure full modularity of the system, at the time of registration, the web app also creates directories based on the user's username, and subdirectories (based on the cat's name) for the multiple cats the user registers. These subdirectories then contain JSON files which the web application reads from to display the data in a graphical manner on the web page.

B. System Connections

1) Tags → Hub

The RPI receives packets of information through BLE 5.0 using the bluepy python library. It first scans and connects to the correct bluetooth device, then gets the information by looking at the characteristic values. In order to conserve sensor tag battery life, bluetooth transmissions are scheduled to occur once every four hours. This is an acceptable time frame because, while TracKat is technically a real-time system, the results it produces (food consumption graphed over time) is not data that is highly time critical. The user also has an option to manually request an immediate, unscheduled update with the "refresh" button on their profile on the webapp.

The sensor tags store equal size arrays of valid food/water weights, valid eating event timestamps, and microchip ID numbers. Each array's values correspond to the other values based on element index. For example, the weight of food at weight array index 3 was detected at timestamp array index 3 and was eaten by a cat with the ID stored at microchip array index 3. The tags dump this data to the hub each time they establish a connection along with the baseline value, array length, and maximum value (updated upon food refill), and then return to low power operation upon bluetooth disconnect. The hub software then interprets these arrays and sorts them based on microchip ID into the appropriate cat profiles.

2) RPi ⟷ Web App

After the RPI receives data via BLE, it interprets the sensor data as explained in the hub section. It then packages the data in a standardized JSON file for each cat which can be easily read by the web application, which goes on to display the packaged data in a graphical manner.

When a user is registering information about themselves and their cats, the web app updates a separate JSON file with that information and stores it in a modular manner for organized, accessible format for the RPI. The RPI then uses this information to perform calculations such as finding the volume, knowing which cat has which microchip number, and storing other general information about the cats and users.

IV. Design Trade Studies

A. System Architecture: Single Device vs. Distributed System

An early trade study the team performed concerned the overall configuration of the system. The question to answer was whether to create a single, internet connected device versus splitting it into a distributed bluetooth network of multiple sensor tags with a single, internet connected central hub. Ultimately, the latter approach was chosen for its greater ease of use and implementation. It would be far simpler to support modular configurations (changing numbers of bowls) by simply connecting more individual tags to a network rather than designing mechanical and electrical hardware to connect/disconnect different bowls as a single device. The distributed approach also offered more flexibility in system configuration as the sensor tag bowls could be placed anywhere within a home versus a single, modular device having all bowls in one place.

B. Power

Another key design trade study concerned the power sources for the different components of the system. In order to allow users ultimate flexibility in terms of bowl placement, the sensor tags were designed to operate from battery power. Rechargeable LiPo batteries were ruled out because of concerns about user safety (designing a tool for animals with a potentially explosive component would not be ideal). A rough estimate of the power consumption of the chosen BLE module intermittently running a load cell ADC suggested that power consumption of well under 150mAh over the course of 8 weeks was achievable based on sensor sampling rates and bluetooth communication rates. The CR2032 coin cell was chosen over AA batteries because of its smaller form factor and because it provided sufficient power based on consumption estimates (220mAh per coin cell).

Being able to connect to the tags wirelessly, the hub computer could simply be placed near a wall outlet independent of tag location, so the chosen power system for the RPI was simply an off-the-shelf wall outlet adapter.

C. BLE Modules:

The primary drivers of the BLE module trade study were power consumption specifications, simplicity of implementation, and availability. The DA14531 chip was chosen for several reasons. Most importantly, it supported BLE 5.1 so it had the capabilities required by the project. When comparing to other chips on the market, we found that it contained every peripheral we needed without being bloated with unnecessary “extras", had excellent and extensive documentation, and was readily available in stock. It also boasted extremely low power consumption due to using an Arm Cortex M0+ processor in concert with what is currently the least power-hungry bluetooth transceiver on the market. The DA14531 consumed 270nA in hibernation, 3.5mA for BLE TX and 2.2mA for BLE RX. In general it had all the capabilities we needed while keeping power consumption extremely low.

D. Deployment on Raspberry Pi vs. Cloud

Initially at the time of the design review, we were planning on having the web server be deployed on the cloud, either through an Amazon EC2 or S3 instance. However, after discussions with the Professors and the TAs during the interim demo, we realized a much better and logical solution would be to have it deployed on the Raspberry Pi. It supports full functionality that the cloud would have provided and allows us to access the web application from anywhere (as demonstrated in the demo where we had the server running on one laptop and the web application being accessed from another). The files we were creating and maintaining had no reason to exist on the cloud and are all worked through locally on the hub.

E. Django vs. Flask

We were pretty certain that we wanted to proceed with a Python based web application. Given that, Django and Flask are the two most popular frameworks as of 2021.One of the reasons why Django turned out to be an obvious choice was its built in object-relational mapping system, which would make querying from the database much easier. Another reason was that Django is much stronger than Flask in terms of security, offering protection against cross-site scripting (XSS), cross-site request forgery (CSRF), and sql injection attacks. The final nail in the coffin was Django’s built-in authentication system, making it easier to verify users wishing to access information about their cat.

F. Information Display on Graphs

All the information that our graphs would need was being stored in arrays being populated by the RPI. One thing to consider, however, was what would be the most user-friendly way to display the data in a manner that would allow the user to easily understand their cats’ habits.

We figured that it would be best to have separate charts for the food and water consumption of the cats since the amount they would be consuming would vary quite a bit for both of them, hence resulting in quite an uneven distribution of the y-axis if we would have chosen to have them both on the same graph. We were also initially planning on displaying the volume of consumption, however we quickly realized that the graphs for those would turn out the same since it would just be a shift on the axis by a constant factor, hence it wouldn’t provide any additional useful information to the user.

G. Bluetooth vs. WiFi

To determine how we would have the sensor tags communicate with the RPI, we primarily focused on power, flexibility, and range. In terms of power consumption Bluetooth (especially the BLE 5.0 specification) completely outclassed WiFi, with the former having example use cases of battery powered devices running for approximately two years on a single charge, as opposed to WiFi needing to be hooked up to an outlet. Regarding flexibility, bluetooth allowed us to be much more adaptable with the number of bowls we used. When looking at cost however, implementing WiFi would have been cheaper with some boards and modules coming in at under \$5, whereas implementing bluetooth required approximately \$5-15 per tag and another \$60 for the two evaluation boards we used to develop on. Given our relatively large budget limit however, cost was not a major factor that affected our choice. WiFi would have much better range by spanning an entire home versus BLE 5.0 only reaching roughly 240 meters outside and around 40 meters indoors. Still, the extreme difference in power consumption swung the decision towards using bluetooth.

H. Bluetooth 4 vs. Bluetooth 5

When looking at speed, we saw that Bluetooth 5 had twice that of Bluetooth 4 due to the much larger bandwidth provided in the newer version. In regards to range, we saw that Bluetooth 5 could serve up to 4 times the distance to which it should allow connectivity. When considering power, we found that Bluetooth 5 tended to use up less than the older standard, thus allowing the sensor tags to keep running for a longer period of time. Finally when comparing support for IoT devices, Bluetooth 5 came out on top primarily because of its increased range and speed. Using Bluetooth 5 over Bluetooth 4 was an easy decision. This primarily drove the decision to use the RPI 4 as a hub instead of the RPI 3, as the 4 supported Bluetooth 5.0 whereas the 3 did not.

I. Weight Sensors: Load Cell vs. Force Sensitive Resistor

When taking into consideration factors like accuracy, hysteresis, and granularity, the load cell came out on top as compared to the force sensitive resistor, as it purportedly had a much smaller error rate of +/- 5%. Load cells are also more rigid, meaning they would survive better over many uses by potentially mischievous cats.

V. System Description

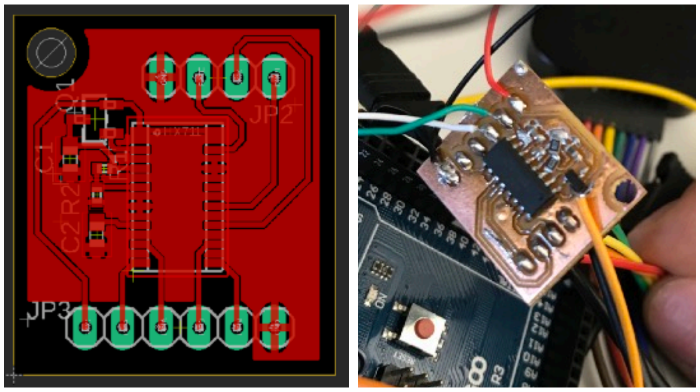
A.Sensor Tags

1)Electronics

Perhaps the most important aspect of the sensor tags was the electrical hardware. The tag PCB's were designed from scratch and manufactured in house using a Bantam PCB mill. The initial iteration of the tag schematic was designed around using a DA14531 chip because the BLE module version of it was not in stock. Thus all supporting components needed to be spec'd, researched, and purchased.

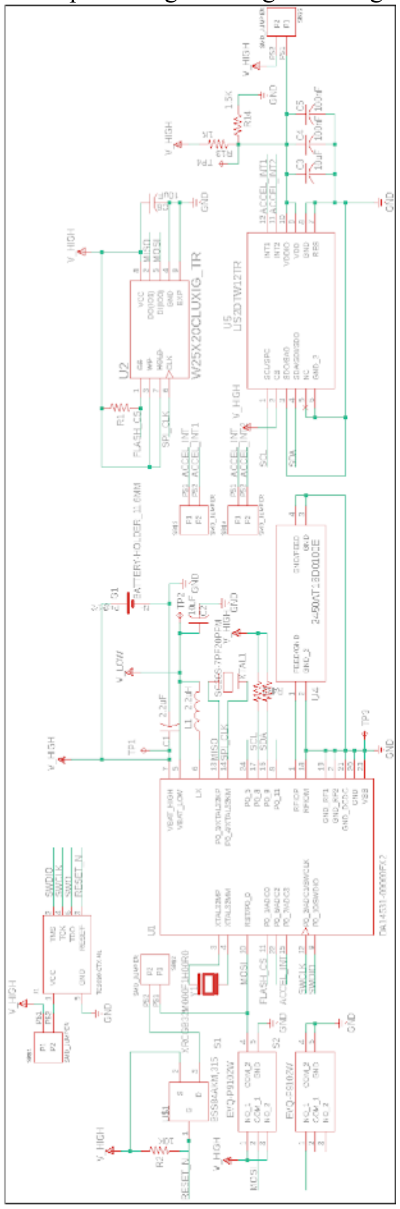
This version of the board would feature the DA14531 chip, a surface mount external antenna, 2MB SPI Flash memory chip, an external crystal oscillator, an accelerometer connected via I2C lines, an SWD port, a user button, a reset button, and various other support circuitry components. The layout for this design proved to be far more difficult than anticipated, as proper RF design principles had to be adhered to, such as using ground stitching and fencing vias and calculating trace widths to support the external antenna.

Luckily, the module version of the DA14531 became available part way through the semester. This module enclosed the DA14531 chip, the SPI flash memory, and the crystal oscillator into a single package that could be surface mounted to a PCB. Most importantly, the module had a PCB antenna built in, meaning that complicated RF design could be avoided for the rest of the PCB layout. The weight tag board was re-designed around this module and included the I2C accelerometer, the SWD port, a coin cell battery clip, a power switch, a user button, and a reset button. This board also included the HX711 chip - an amplifier and 24 bit ADC used to interface with the load cell sensor.

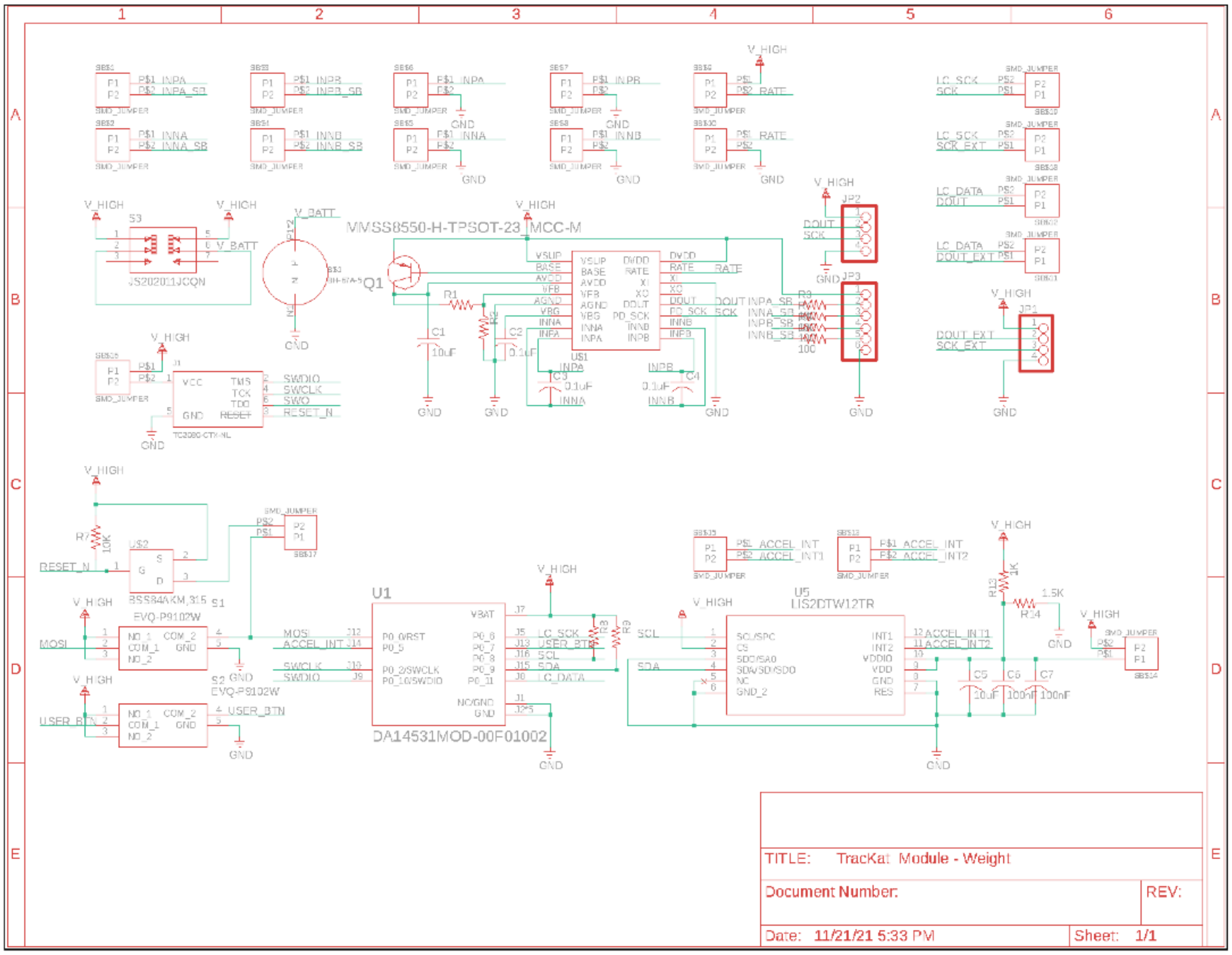


2. Custom made HX711 breakout board used to verify the design

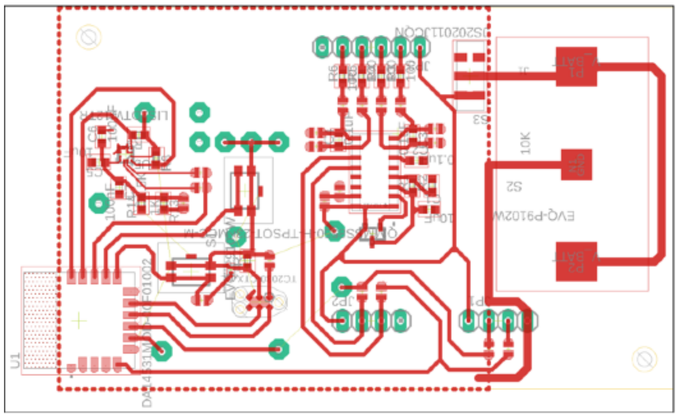
Prior to fabrication, an HX711 breakout board was built to verify the design. Likewise, accelerometer and RFID breakout boards were designed, etched, and populated. RFID sensing was later scrapped to descope the project and shift greater focus towards perfecting the weight sensing application.



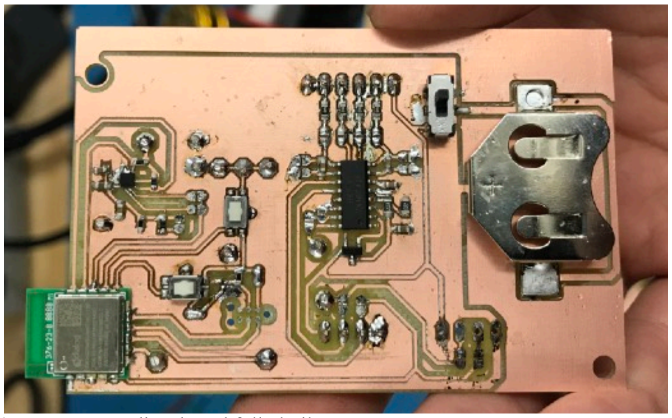
3. The initial tag schematic was overly complicated



4. The intermediate tag design was built around the DA14531 module rather than the chip, making it far easier to fabricate



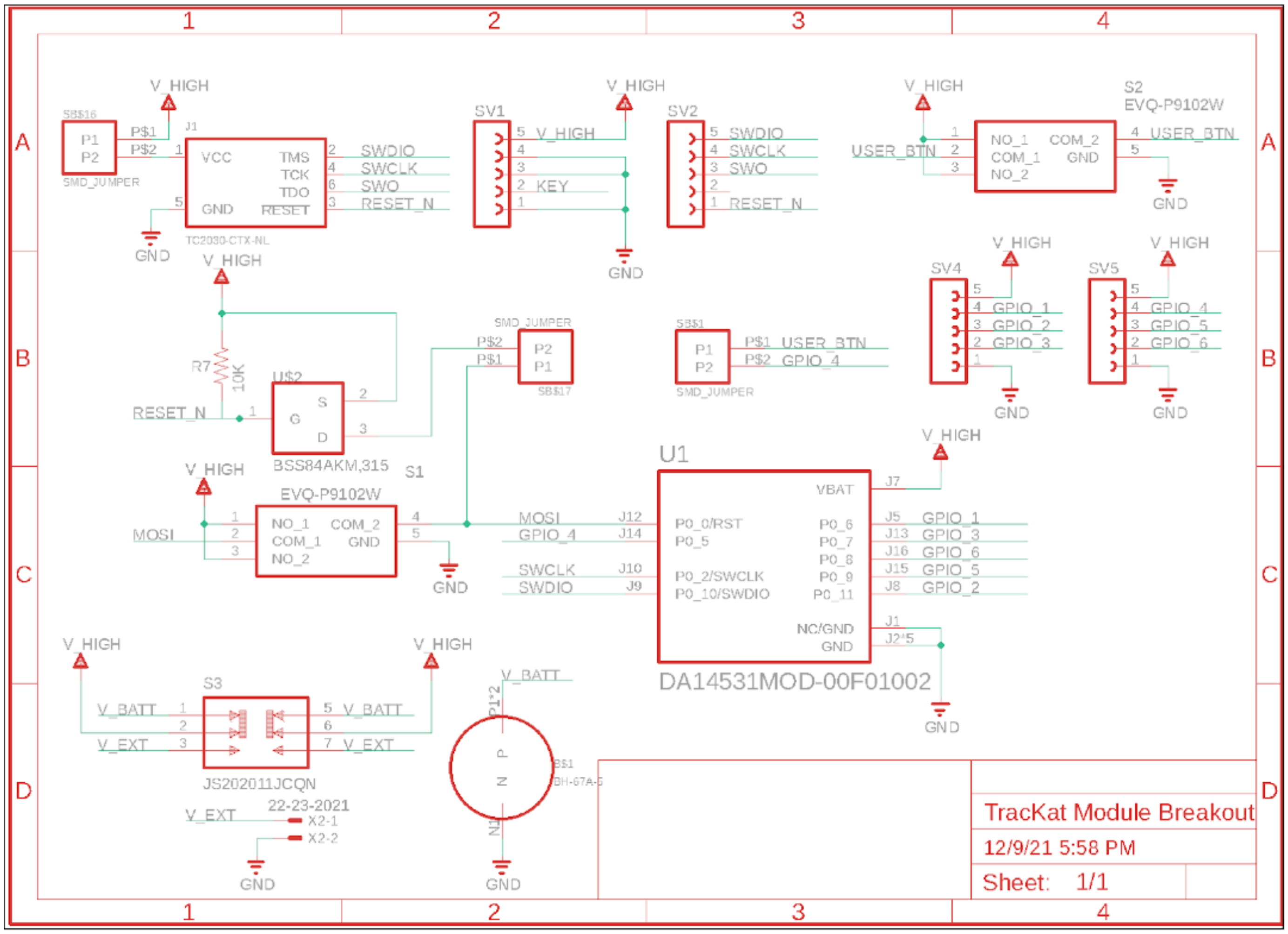
5. Intermediate board layout



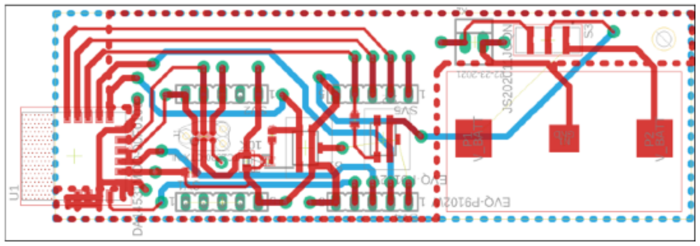
6. Intermediate board fully built out

This board was designed to be programmed with a six-pinned SWD connector that would press directly onto the copper contacts. This connection worked, but unfortunately we soon realized that the CMSIS-DAP board used to flash the module had problems saving new code into the SPI flash memory. This iteration of the board thus could only be run tethered to a computer from a debug port.

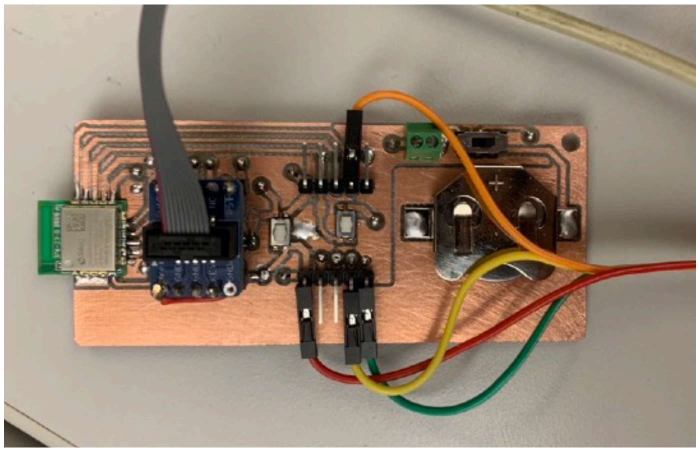
This version of the sensor tag board was etched on a single layer PCB, making layout another serious challenge. The next, and final iteration of the board would be made with two layers. The next iteration would also solve the flashing problem by include a new 10 pin port adapter for a new JLink flash board, which used JTAG directly rather than going through a CMSIS-DAP.



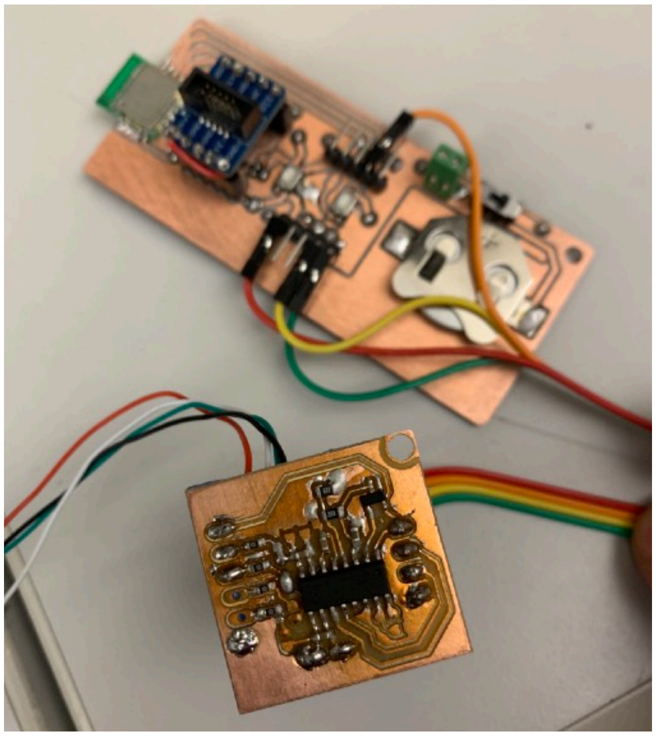
7. Final board schematic



8. Final board layout



9. Final board with flash board ribbon cable plugged in



10. Final board connected to an improved iteration of the external HX711 chip breakout board

The final version of the sensor tag board was simplified with the removal of the built-in accelerometer and load cell ADC in favor of simply breaking out all of the GPIO lines and then attaching the requisite sensors from external breakout boards. This was done to allow for maximum design flexibility and ease of implementation.

2)Firmware:

The firmware for the sensor tag boards was built with Dialog System's SDK 6. The SDK features a pre-made kernel and bluetooth handling, which developers can tap into by writing custom callback functions that are triggered by various events. This custom code is referred to as the "user space". The sensor tag user space handles configuring GPIO, running a custom BLE profile, driving the HX711 Amplifier/ADC chip to read load cell sensor values, catching valid "eating events", and keeping time of each valid eating instance. An I2C accelerometer driver and RFID driver were partially implemented but ultimately left out due to RFID sensing being removed from the scope of the hardware portion. To simulate microchip ID's (to allow the rest of the project pipeline to test monitoring multiple cats), the general purpose user button of the custom sensor tag board is used to manually select between two different microchip ID's - if an eating event happens when the button is pressed it is logged under one cat's profile, while being logged as the other cat if the event happens while the button is released. A second version of this test simply tied what would be the user button input to an open GPIO line that had no internal pullup or pulldown resistors - this effectively made each eating event fall under a random choice of the two cat ID's thus simulating the randomness of real world multi-cat operation.

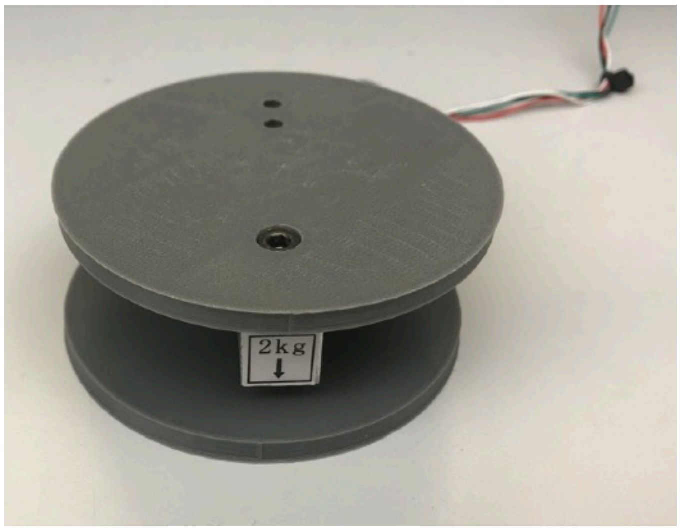
The HX711 is driven by running a systick timer that fires interrupts every 20µs. A custom interrupt handler runs logic to decide how each interrupt affects the rest of the system. Upon the first bluetooth connection being established, the timer is initialized and starts ticking. The load cell is read by waiting for the data output line of the HX711 to signal that a value is ready to be sent, and then pulsing the clock line 25 times to read the 24 bit ADC value from the chip's internal shift register. Upon a successful read, the load cell is put to sleep to conserve power until the time comes to read it again.

Upon initial connection, the sensor tag immediately runs a calibration routine to tare itself and reports this baseline value as a characteristic in the custom profile read by the hub computer. It then reads the load cell once every 5 seconds with the rest of the time spent in low power sleep mode operation. The driver maintains a maximum value that is updated upon detection of a large weight being held constant for an extended period of time. This way, users never have to manually inform the system every time they fill a bowl. An eating event is considered valid only if it decreases the weight in the bowl by an amount greater than the threshold value. This thresholding is necessary to ensure that spurious vibrations and other sensor noise are not counted as eating events.

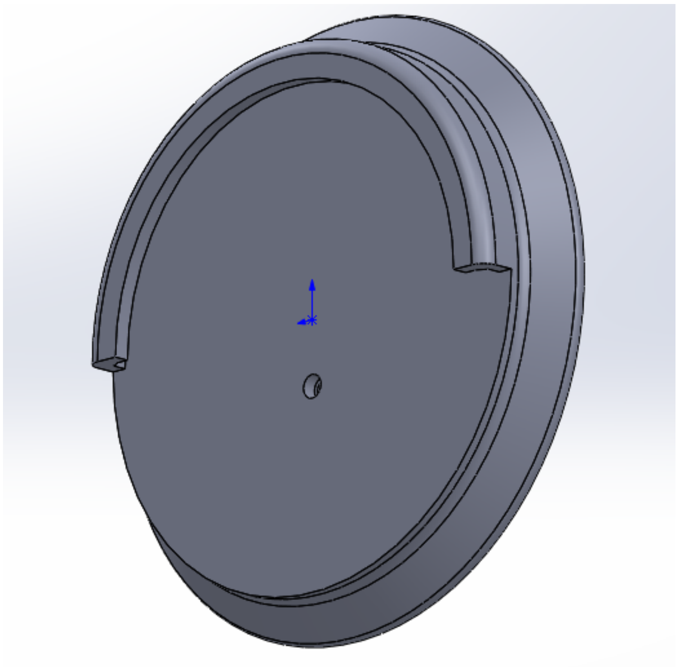
When an eating event is captured, the value is saved as an element in the weight array, the timestamp of the event is converted into seconds and saved as an element in the timestamp array, and the microchip of the cat at the time of the event is saved as an element of the microchip array. These three arrays must be the same size as elements of the same index in each array correspond to one another - cat ID at element "x" ate food amount at element "x" at timestamp element "x". When the hub preforms a reading of the sensor tag, the arrays are dumped and then erased to conserve the limited local memory.

3)Mechanical Enclosure:

The most critical part of the mechanical design was ensuring that the sensor tag electronics would be protected from various debris falling from the food and water bowls. As such, the top plate of the weight tag was designed with an umbrella-like outer skirt to push any messy food a cat might spill away from the hardware underneath. The top plate was also designed with a semi-circular lip on top to firmly secure the food bowl. The press fit was tight enough to grip the bowl without being a nuisance to remove it.

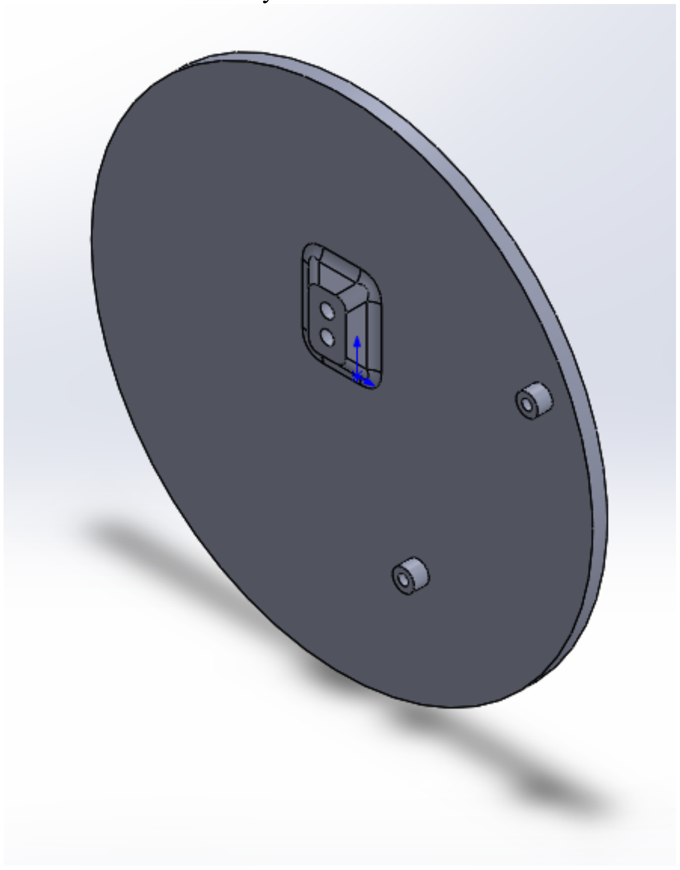


11. Custom designed test hardware was made first to ensure dimensions and fittings were correct

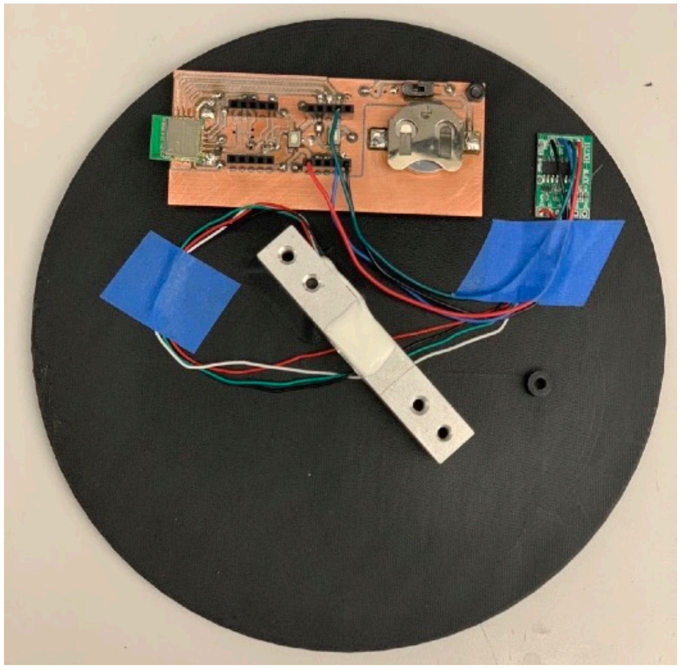


12. Top plate of the weight tag enclosure

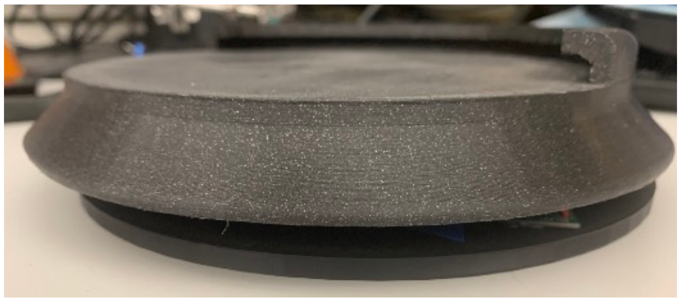
The bottom plate of the tag included standoffs to secure the custom tag PCB and keep it further offset from the floor which could also become messy.



13. Bottom plate of the weight tag enclosure



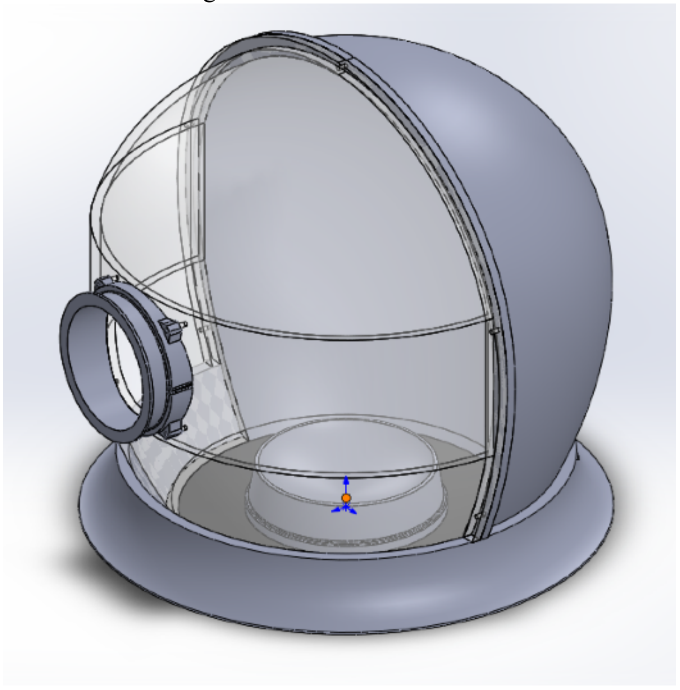
14. Final board and sensor attached to bottom plate



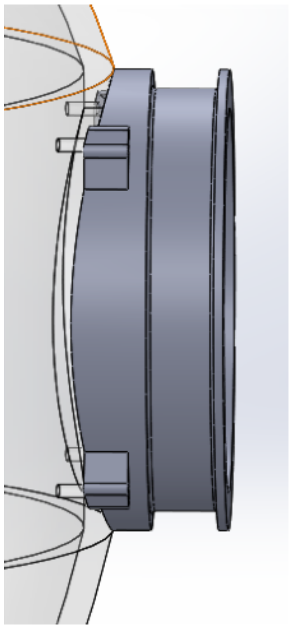
15. Side profile of fully assembled tag with electronics housed inside

16. Fully assembled tag loaded with food

Prior to descoping the project away from RFID, a mock enclosure was designed and the antenna coil frame was built.



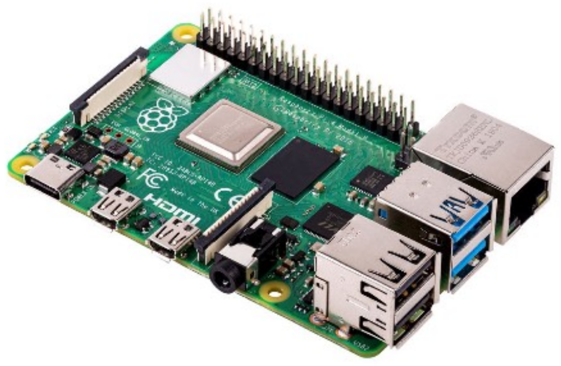
17. Proposed RFID bowl enclosure



18. Side profile of RFID antenna coil frame

19. Custom made RFID antenna coil wrapped around custom designed and 3D printed frame

B.Hub



20. Raspberry Pi 4 as a Hub

As explained in previous sections, the hub has a bluetooth driver which communicates with the tags, reads in the values, then organizes it, and also has a portion that does data analysis, and communicates with the web application via JSON files. All of the code written that will be detailed in the later sections are then copied over and used for the refresh button on the web app's end.

As a general reminder, the hub communicates with the web app using .json files. There is one main file that includes information that the user entered during registration. For example, the user's email, name, and other information about the user, and each cat's name, microchip number, weight, and other information about their cat. We also have a file system, where there is a folder for each user, of which files are named after each cat. Inside these files are the data that is collected from the user. The hub fills the files, and the web app uses these values to display.

All of the following code was done for each user. Since each user inputs the bluetooth addresses for the tags they have "bought", every 4 hours we are able to connect to all the user's tags and gather values, then do the following procedure:

1)Bluepy

To create the bluetooth driver, the bluepy module was used. This is a module that helps you use bluetooth low energy. It allows you to connect to a peripheral device that has bluetooth low energy enabled, then read from, and write from the device. While using it, there were lots of snags in the beginning. The documentation is not extensive, and it was very confusing to work with. For instance, when we started using it to connect to the peripherals, it would refuse to find and connect to any peripheral device. We later found out it was searching for all address types and had to do lots of google searching to figure out what was wrong. Once we were searching for only random addresses, we were able to connect but this took weeks to figure out. Otherwise, it was quite confusing as you have to decide which service, then which characteristic you want to read from, and then read. While we had documentation online that had examples, we carefully went through what each was doing and used all of these functions.

In general, we used this module to connect to, and then read the values for each user. Since the user also says whether they want to use either the food, water, or microchip values, we have also written custom code to ensure the following procedures only happen to either the food and/or water bowls. Additionally, if the user doesn't want to use the microchip tag, the web app inputs 0 as the microchip number. If this is seen, the hub assumes there is only one cat for that user and only updates that file.

2)Converting from Little Endian Arrays and Values

Once we are able to then read via bluetooth, we then have to convert the little endian arrays and values into readable information. First we wrote custom code to look through the array and index into each 4 bytes, and using struct.unpack(), turn it into the int version. We would do this for all three arrays, the microchip array, weight array, and the timestamp array. We also read in the baseline value and convert it. While the microchip array and time stamp arrays are all at the right units, we had to convert the weight array into grams using the baseline value and another static ratio value. We then had all three arrays ready to go.

3)Organizing Using Microchip Number

We then wrote custom code to organize the information. First we found all the unique microchip numbers that were in the microchip array. For each unique number, we were then able to find all elements in it that matched that. We then were able to find the index of where that was, and index into both the weight and time stamp values and add it to a new list. Therefore, each unique microchip number had a weight and time stamp list.

4)Analysis on Arrays

The following process is done for each individual cat in the user's folder. For the weight values, we just run it through a for loop to find the difference between adjacent indices. Additionally, it looks at the last value from the weights already stored in the .json file for the cat and makes sure that this is also accounted for. It also calculates the food and water consumption and raw weights in terms of volume as well. Then the .json file for that cat is updated.

We hit some bugs in the beginning because fixing edge cases can sometimes be hard. We ended up using dummy arrays and testing out all assortments of arrays that could come in. From being empty, to being 1 element long, etc. to make sure it worked for all types of data coming in.

5)Statistical Analysis

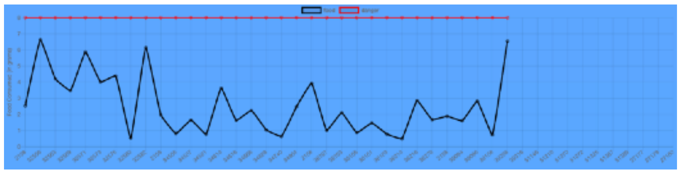
After the process explained above is done for all users and therefore all the cats, data analysis is performed on all of them in order to figure out if there was an unusual eating event. To do this, we determine values called danger levels, where if a data point goes over or under this value there is a problem. We use the idea that 95% of data falls within 2 standard deviations of the mean. We use up till the last 3 values to determine the original mean and standard deviation. Then we look at the last 3 values to see if they fall outside of this "danger level". While we landed on 3 for the demo, this is a hard coded value that would change depending on how often the cat would eat every 4 hours.

To achieve this, we used a python module called statistics to find the mean and standard deviation of the beginning data points inside of the consumed data. Then we were able to use these values to figure out what the danger level would be. This danger level would then get put as a static value into the cat's file to then be displayed on the web app.

Since the values coming from the tags are only saved if the amount consumed is over a certain threshold, there was no way we could use the weight values to then figure out if they were eating too little. Because then we would not be able to catch the smaller eating events. Hence for overeating or over drinking we use the weight values, but for under drinking and undereating, we use the timestamp values to see if there was too much time between eating events.

If this data analysis is done and there is an unusual event, then it will use code written by Tarush that sends an email. It will send a custom message that shows the user's name, cat's name, the type of event, and a suggested course of action depending on the event.

C.Web App



21. Screenshot of Web Application

1)Registration and Login

For building the web application, we used the Django framework and have the web application deployed on the RPI, allowing for accessibility on different devices. When the user first visits the page, they are met with a login page which asks for the user's username and password. For authentication, Django provides us with a user authentication system which will handle user accounts, permissions, and groups. If the user does not have an account, we provide a link which directs them to a registration page. Over here, they can create their account as well as store basic information about the cats they wish to register. They also input whether they wish to keep track of food and/or water intake as well as the bluetooth addresses for the bowls. Both the login and registration pages on the web application are created using Django's forms class.

2)Storage of Data in Database and Files

This data is then stored and managed in an SQlite database where we have two separate classes, the cat owner class for each user and a cat class, for each cat registered for each user. Along with this, the web application is responsible for creating directories based on the user's username and subdirectories based on the cat's name. Each inner cat directory then consists of a JSON file which contains arrays of data (ex. time arrays, food and water consumed arrays, updated danger levels) from which the charts read information from and display.

We also store this data in a separate JSON file to allow easy accessibility of this information for the other software that is written on the hub. This allows for easy integration between the two subsystems and avoids requiring us to always have to access our database constantly for reading and writing.

3)Bluetooth Integration

One of the pain points we faced with regards to allowing full integration from the hardware with the web app was getting the bluetooth module to work with the virtual environment running on the Raspberry Pi. Due to my lack of experience with this module and with importing external packages, a lot of time was spent on researching, debugging, and ensuring correct installation. Ultimately the solution ended up being a quick fix, but a good amount of time was spent in figuring out the issue.

This however, was a big accomplishment and really improved the functionality of the web application. Originally, the graphs were only being updated at a fixed interval of time. However, now with the bluetooth working from the server, the user, from their home profile page, could request for the most immediate information from the bluetooth tags with the click of a button which would write to the json file and update the graphs accordingly.

4)Charts JS Graphs

The cat's page comprises the graphs tracking their food and water consumption. This was achieved using the Charts JS library, where we use line graphs to display how the consumption amount changes over time as well as the danger level determined by statistical analysis being conducted on the hub. The Y-axis of the graph displays the food/water consumption and the X-axis of the graph consists of the timestamps at which the cat ate/drank a minimum threshold amount of food/water to trigger the tags.

5)Notifications

When we do detect an unusual event occurring with the cat's consumption, we wanted to immediately inform the owner about this. To do so, we would send an email to the owner, which we had stored at the time of registration. We use Python's smtplib (SMTP protocol client) library, where we would create a client session, ensure our connection is secure, and send an email to the user from a global email we create for the web application. The email message is customized based on the cat eating or drinking less than it usually does or much larger than it does.

6)Raspberry Pi Deployment

Initially, we were running our application on Django's local web server and ensuring it would work as intended. Later on to allow for global access, we deployed it fully on the Raspberry Pi on it's IP address, such that we could have the server running on one end and could access it on other devices. This required ensuring that the hub had full access to all of the static files (ex. CSS) and ensuring that it could perform reads and writes from the database as well.

VI.Test and Validation

A.Accurate Cat Feeding and Drinking Detection

I. Weight Tag Value Test

Reading #	**Weight Values and Errors**			
	Sensor Reading (g)	***Actual Weight (g)***	***Raw Error***	***% Error***
1	4.61	5.67	-1.05	18.51
2	2.15	3.17	-1.02	32.16
3	3.46	4.54	-1.08	23.79
4	4.25	4.66	-0.39	8.37
5	17.49	16.22	+1.27	7.83
6	11.30	11.86	-0.56	4.72
7	15.33	16.57	-1.24	7.48
8	1.85	1.88	-0.03	1.59
9	5.30	4.84	+0.46	9.50
10	1.85	2.76	-0.91	32.97
Totals:	72.17	67.59	8.01	
Average Error: **14.69%**				
Raw Absolute Window Error (absolute value of distance from actual weight): **11.10%**				
Raw Value Error (% off from actual weight totaled across 10 readings): **6.35%**				

Cat feeding and drinking detection was tested by grabbing ten varying handfuls of food back-to-back from the food bowl, and comparing the sensor reading against the actual weight of each handful of food (as reported by a highly accurate jewelry scale). Technically, the requirements laid out as project goals were to keep false positives and false negatives within 10%. In this sense, the system performed nearly flawlessly as it did not fail to report a single food event. The sensor did however detect an extra handful of food that was not taken at the end of the test session. Thus the system had a **false negatives rate of 0% and false positives of 10%**.

More interesting than that was the error rate. The average sensor **error per reading came out to 14.69%**. Seeing as the TracKat system is designed to keep rolling averages of sensed values over long periods of time, any single erroneous value is less important than error spread across an average of readings. The total raw value error of the system across ten readings (taking the absolute value of each raw difference and summing together) came to 8.01, or just **11.10%** off from the actual weight. The total grams the system was off across all ten readings came out to just **6.35%**.

These results were extremely encouraging, but could likely further be improved in the future with more sophisticated calibration routines.

B.Modular Configuration

For the web application, we wanted to ensure that its functionality was fully modular. We conducted multiple tests in order to ensure that the web app could handle any combination of users, cats, and food and water consumption. This was first tested at the time of registration, by letting the user register one cat or multiple cats, and having the optionality of tracking only food or water (as opposed to both). We also made sure that, based on what users input, the web app only displayed graphs for what the user wished to track (so having only the food/water graph show up as opposed to both) and that in the case of an unusual event an email was sent to the correct user with accurate information on what the detected issue was. These features were all successfully implemented so this requirement was fully met.

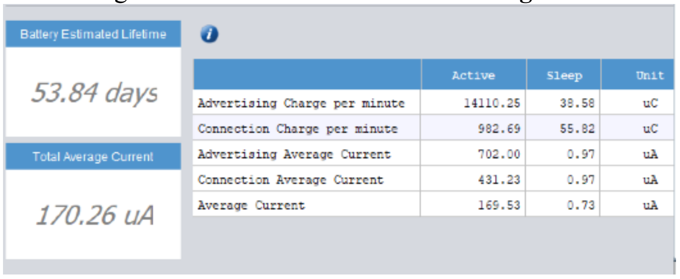
C.Weight Sensing Granularity

Weight sensing granularity was derived theoretically by taking the total weight of the load cells used and dividing this value by the 24 bits the HX711 was capable of outputting. The sensor tags' 1kg load cells would thus theoretically be capable of sensing differences of roughly 1000 / (2 ^ 24) = 0.0000596 grams. This theoretical value, of course is the highly idealized version that ignores sensor offsets and threshold values. Sensor offset (the reading at which 0 actually started) was determined experimentally by simply running the sensor calibration on zero weight (but including the weight of the empty top plate and empty bowl, as this would factor into what the "0" food value was). Afterwards, a known weight was placed on the tag and the weight in grams was divided by the difference in raw values to derive the finest possible granularity achievable. This value came out to roughly 0.0008g per bit when using 2kg load cells. The final tags, however, were switched to using 1kg load cells, as the maximum possible weight of food that could fit in the bowl plus the weight of the bowl and top plate came out to less than 420 grams. The new theoretical value came out to roughly 0.00048g/b.

This value was further cut down by the necessary thresholding, which cut out readings that differed less than a certain amount from the previous valid reading. After several tests, a raw value of 3000 was set as the best threshold in terms of balancing not missing valid readings while also not generating false positives. 3000b * 0.00048g/b gave a final granularity of **1.44 grams**, or roughly 5 kernels of kibble. This was much higher than the desired granularity of 0.1 grams, but still sufficient to detect most reasonable average eating sessions. It should be noted that lower threshold values produced much better granularity results but at the cost of worse false positives. A threshold of 250b produced granularity of **0.12 grams**.

D. Long Battery Life

Battery life was calculated using Dialog Systems' Power Profiling tool. This software allowed us to enter values for how much current external components used, how long they were on, how often BLE transmissions would take place, what percent of time would be spent in Sleep Mode, and other such factors. It then calculated power consumption and output a resulting battery life estimate given a specific size power bank. Given the single coin cell capacity of 220mAh, we derived a battery life estimate of **53.84 days** - just under our 56 day target. This requirement as successfully met considering the result was **within 4% of the target**.



	Active	Sleep	Unit
Advertising Charge per minute	14110.25	38.58	uC
Connection Charge per minute	982.69	55.82	uC
Advertising Average Current	702.00	0.97	uA
Connection Average Current	431.23	0.97	uA
Average Current	169.53	0.73	uA

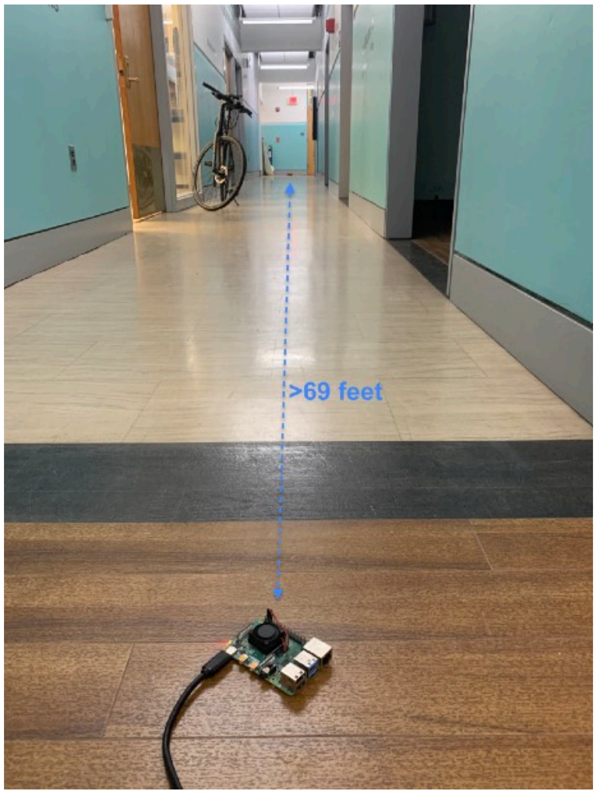
22. Power Profiler results

E.Displayed Data Time Range

Data time range was based entirely on the storage capacity of the RPI 4 hub computer. Given a microSD card size of 128GB, there was roughly 97GB of available storage. To ensure no edge cases could crop up, the absolute worst case scenario was used - assuming every single sensor reading resulted in a valid reading. This would mean a new reading every 5 seconds for an entire year - 6311520 in total. This value multiplied by the 12 bytes of information per read (weight integer, timestamp integer, and microchip integer), gave a total year's maximum data use of 75738240 bytes, just 0.076GB. This requirement was firmly knocked out of the park, with just **0.00781%** memory used for a whole year under extreme worst case conditions.

F.Reach Across Average American Apartment

Another goal of the project was to be able to have sensor tag bowls be placeable anywhere within a user's home. To test this we set a goal of reaching across an average sized American apartment (~950sqft). Assuming the hub would be placed in the middle of the diagonal of the theoretical room, this meant a target of supporting BLE communication across a 25 foot distance from tag to hub. This requirement was tested by simply placing the hub at the end of a long hallway and walking back with the tag until a communication error was encountered. We reached the full length of the ECE wing hallway without running into any communication errors and called it a day. This means that the supported communication distance achieved was **>69 feet!**



23. Hallway distance test

VII. Project Management

A. Schedule

Schedule is included in Appendix. Page 17

B. Team Member Responsibilities

We split responsibilities for each team member based on their strengths. Lucas worked primarily on the hardware side. He developed all of the embedded electronics, firmware, and mechanical parts for the sensor tags. MeeDm worked on the hub, specifically, establishing communication between the BLE module and RPI hub, decoding the packets of information sent from the sensors, doing statistical analysis, and communicating with the web app via JSON files. Tarush worked on building and deploying the web application, storing and managing data for users and their cats, connecting to the bluetooth tags from the web app, displaying graphical data for each individual cat, and communicating and sending information from the web app to the RPI.

C. Budget

Budget is included in Appendix. Page 16.

1) AWS Credits

Although we requested AWS credits for our project, we did not end up using them since as mentioned in previous sections, we ended up deploying our web application on the RPI instead. However, we would still like to thank Amazon for providing us with these credits for our project.

D. Risk Management

One of the greatest risks the team encountered was supply chain problems. Specifically, these issues impacted chip selection as well as board fabrication. The HX711 chip was entirely out of stock, which forced us to desolder the chips from existing breakout boards we could find around campus. The TMS3705 RFID chip we had designed around was in stock for a short duration but ran out before our order could be placed (the day after the order form was submitted). We re-spec'd the RFID portion of the design around an existing, in-stock chip, but ultimately abandoned RFID sensing as it was out of scope for a single semester project. PCB fabrication also would have taken too long to allow for adequate prototyping, so we opted to manufacture them in house with a Bantam PCB mill.

VIII. Ethical Issues

If TracKat is being relied upon to give an accurate picture of a cat's health and fails, there can be two cases. In the first, the user may be notified of a potential problem where there is no problem. This would not have too devastating an impact, but eventually could become annoying if it continued to happen. The second case is that the user may not be notified of a problem when there is in fact a problem with the cat. Since the user is relying on the device to monitor the condition of the cat, this false negative case could lead to the cat's health declining. This could possibly result in illness, injury, or even death of the cat. In order to avoid this, a disclaimer could be added to the web application to inform the user to not solely rely on the device to monitor their cat. TracKat performs within the bounds of 10% false positives and 10% false negatives to further mitigate this issue.

Lastly, some parts of the device, with enough wear and tear, could be ripped off and become a choking hazard. This could affect the cats or small children in the household. Care will have to be given to package the device when handing it to a user so that they know this can happen and to use caution if they have a particularly destructive pet or child.

IX. Related Work

There are other products on the market that use the concept of having an RFID tag to distinguish between an owner's different pets. Products like the SureFeed Microchip Small Cat & Dog Feeder and PetLibro Automatic Cat & Dog Feeder store sealed food and have lids which only open for the designated pet. You can also choose to allot specific timings and frequency of when you want to be feeding your pet. Our project takes this in a different direction by allowing owners to develop a better understanding of what their cats' eating and drinking habits on a regular basis. TracKat immediately detects and notifies the owner of sudden changes as opposed to the current products more so functioning as black boxes that performing the mundane tasks necessary for feeding pets. TracKat thus removes the burden from the owners of having to regularly check up on their pet's well-being.

X. Summary

The system was able to perform most of the key components and functions that we set out to build. However, it did not do some. It was able to accurately detect how much food and water was consumed and display it on the web app which was the main goal of the project. We held on to the goal of implementing RFID tags until quite late in the semester, but ultimately had to descope and drop RFID altogether. Still, the hub and web app software was designed to work with microchip array values that would have been sent from the RFID tags, had they been implemented. Using user-provided zip codes to recommend nearby veterinarians was also removed from the scope of the project. Overall, TracKat met nearly all of its stated goals.

A.Future work

There is plenty of work ahead of this project, and we hope to take it much further. First and foremost would be implementing the RFID sensing hardware we initially wanted to build. After that, more carried sensor tags could be developed such as infrared sensor tags and vibration sensor tags. The hub software could be expanded to account for more varied behavior patterns and seek out potential health issues based on cat weight, age, and type. The web app could also be expanded with more user options and controls for expanding or zooming into particular sections of time.

B.Lessons Learned

The most important lesson we learned was how important communication between team members is. The more communication there was between parts, the faster they got integrated. It was necessary for integration, but additionally to keep everyone on the same page for everything. Even if another team member didn't fully understand a part, having a general idea was so important for all of the reports, presentations, demos, but most importantly, integration and to respect one another's anxiety and stress levels.

Another lesson we learned was how to take into consideration other's working schedules. Some of us work better at the last minute and others of us worked better earlier. In order to make sure we all worked together, we ended up dividing up tasks so that the ones that work differently were able to do things at their own pace. Even if it did cause some stress, it ended up making it so that everyone was allowed a chance to do their best.

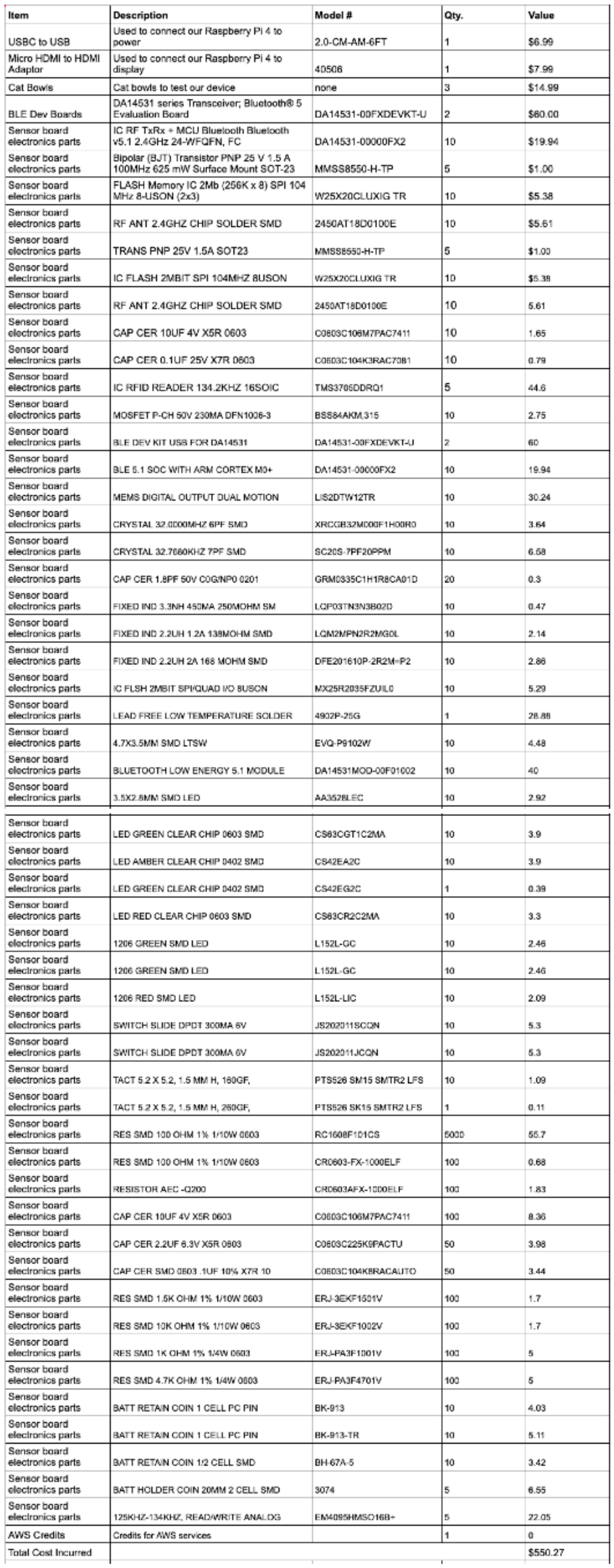
Accurately assessing how long the project would take is also important. We found that the team members that were better at doing this were able to get a more finished product. This is because knowing how long something will take to complete lets you determine the scope of your area properly. That way you can finish the simpler area and build on it if need be.

Glossary of Acronyms

ADC - Analog-Digital Converter
BLE - Bluetooth Low Energy
CMSIS-DAP - Cortex Microcontroller Software Interface Standard Debug Access Port
EC2 - Elastic Compute Cloud
GPIO - general purpose input/output
IC - Integrated Circuit
JSON - Javascript Object Notation
JTAG - Joint Test Action Group
PCB - Printed Circuit Board
RPI - Raspberry Pi
SDK - Software Development Kit
SWD - Software Debug

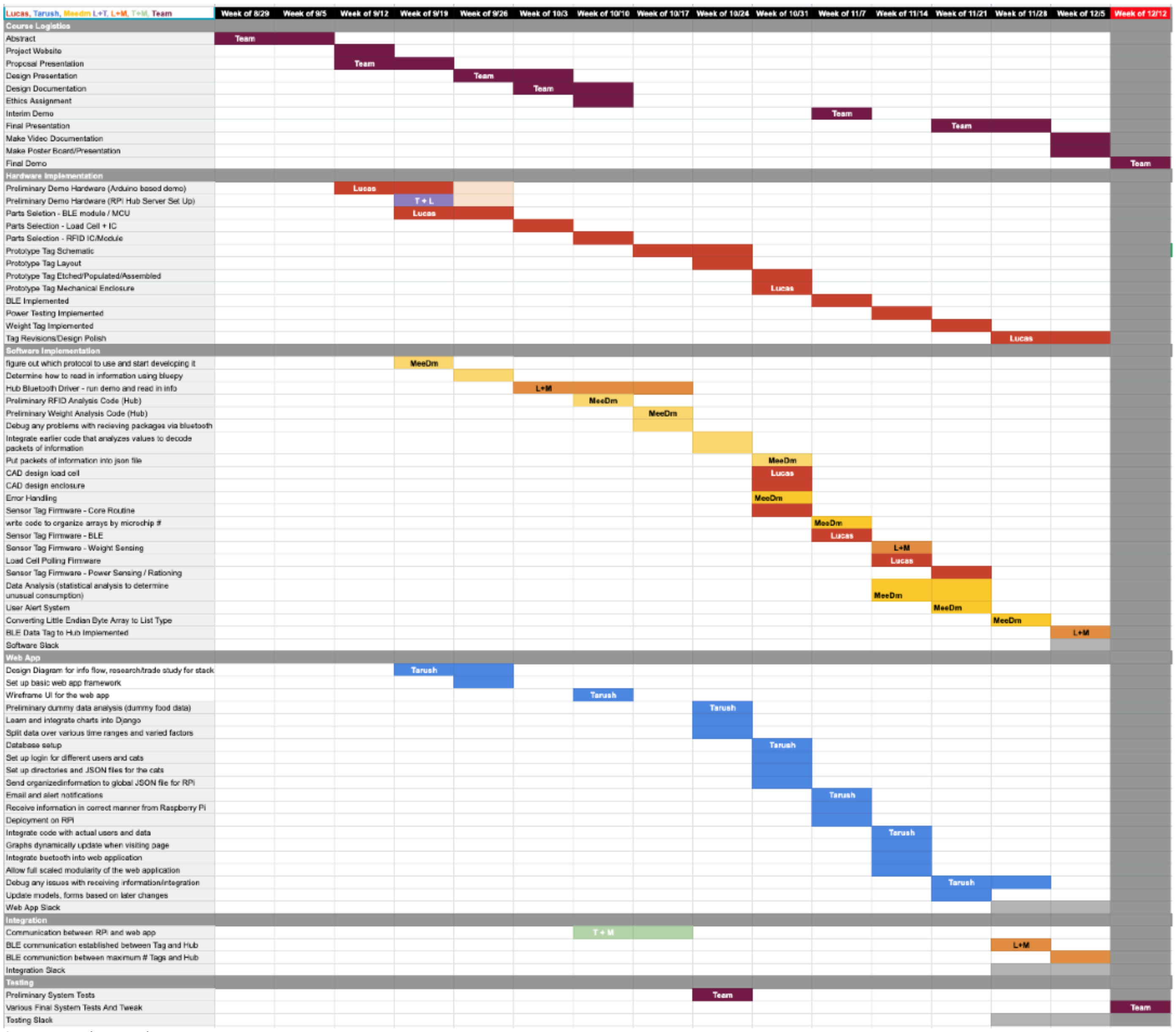
References

1. https://www.newportri.com/ZZ/sponsored/20200811/top-5-ways-to-encourage-your-cat-to-drink-more-water?template=ampart
2. User Authentication in Django https://docs.djangoproject.com/en/3.2/topics/auth/
3. SureFeed Microchip Small Dog & Cat Feeder https://www.chewy.com/surefeed-microchip-small-dog-cat/dp/157145?utm_source=google-product&utm_medium=cpc&utm_campaign=hg&utm_content=SureFeed&utm_term=&gclid=CjwKCAjwh5qLBhALEiwAioods4xazbHapEA-djE2OMsKp41S2hRiWLkrfjCjDPrCWgDGxfhLpycVnBoCkE8QAvD_BwE
4. Petlibro Automatic Dog & Cat Feeder https://www.chewy.com/petlibro-automatic-dog-cat-feeder/dp/303874?utm_source=google-product&utm_medium=cpc&utm_campaign=hg&utm_content=Petlibro&utm_term=&gclid=CjwKCAjwh5qLBhALEiwAioods2d54A1dJfONJXJw9mINDlFYEbLkd4qePIbqOQSyDtPekD1fiauxPhoCj5kQAvD_BwE
5. Picture for user's home page https://wallpapercave.com/wp/21MQXjm.jpg
6. Bootstrap forms https://bootsnipp.com/snippets/z8MPd, https://bootsnipp.com/snippets/bxzmb
7. Python Library for sending emails https://docs.python.org/3/library/smtplib.html
8. Dialog Systems DA14531 Resources: https://www.dialog-semiconductor.com/products/bluetooth-low-energy/da14530-and-da14531
9. RFID 101: System Frequency Ranges
10. BLE 5 vs Thread Reasoning: Range/power: IEEE802.15.4 (thread) vs. Bluetooth 5 - Nordic Q&A - Nordic DevZone
11. I made a REAL magic wand! (DA14531 project)
12. How to Design and Build Your Own Idea - Example: A Magic Wand (with Samson March) (DA14531 project)
13. Vibration Sensor Selection Guide from Cole-Parmer
14. RFID basics by Priority 1 Design
15. Nordic Semiconductor Infocenter
16. RFID Cat Door : 8 Steps
17. BLE and GATT for IoT: Getting Started with Bluetooth Low Energy and the Generic Attribute Profile Specification for IoT | Programmatic Ponderings
18. Home All Products TWN4 MultiTech Nano LF
19. Taidacent Long Range Iso11784/85 Fdx-b Em4305 Ear Tag Reader Rfid Module Ttl Uart 134.2khz Animal Rfid Reader Module - Buy 134khz Reader Rfid Module,Animal Rfid Reader Module,Rfid Module Product on Alibaba.com
20. TMS37157 data sheet, product information and support | TI.com
21. TMS3705 data sheet, product information and support | TI.com
22. https://www.priority1design.com.au/shopfront/index.php?main_page=product_info&cPath=1&products_id=3
23. RF430F5978 data sheet, product information and support | TI.com



Item	Description	Model #	Qty.	Value
USBC to USB	Used to connect our Raspberry Pi 4 to power	2.0-CM-AM-6FT	1	$6.99
Micro HDMI to HDMI Adaptor	Used to connect our Raspberry Pi 4 to display	40506	1	$7.99
Cat Bowls	Cat bowls to test our device	none	3	$14.99
BLE Dev Boards	DA14531 series Transceiver; Bluetooth® 5 Evaluation Board	DA14531-00FXDEVKT-U	2	$60.00
Sensor board electronics parts	IC RF TxRx + MCU Bluetooth Bluetooth v5.1 2.4GHz 24-WFQFN, FC	DA14531-00000FX2	10	$19.94
Sensor board electronics parts	Bipolar (BJT) Transistor PNP 25 V 1.5 A 100MHz 625 mW Surface Mount SOT-23	MMSS8550-H-TP	5	$1.00
Sensor board electronics parts	FLASH Memory IC 2Mb (256K x 8) SPI 104 MHz 8-USON (2x3)	W25X20CLUXIG TR	10	$5.38
Sensor board electronics parts	RF ANT 2.4GHZ CHIP SOLDER SMD	2450AT18D0100E	10	$5.61
Sensor board electronics parts	TRANS PNP 25V 1.5A SOT23	MMSS8550-H-TP	5	$1.00
Sensor board electronics parts	IC FLASH 2MBIT SPI 104MHZ 8USON	W25X20CLUXIG TR	10	$5.38
Sensor board electronics parts	RF ANT 2.4GHZ CHIP SOLDER SMD	2450AT18D0100E	10	5.61
Sensor board electronics parts	CAP CER 10UF 4V X5R 0603	C0603C106M7PAC7411	10	1.65
Sensor board electronics parts	CAP CER 0.1UF 25V X7R 0603	C0603C104K3RAC7081	10	0.79
Sensor board electronics parts	IC RFID READER 134.2KHZ 16SOIC	TMS3705DDRQ1	5	44.6
Sensor board electronics parts	MOSFET P-CH 50V 230MA DFN1006-3	BSS84AKM,315	10	2.75
Sensor board electronics parts	BLE DEV KIT USB FOR DA14531	DA14531-00FXDEVKT-U	2	60
Sensor board electronics parts	BLE 5.1 SOC WITH ARM CORTEX M0+	DA14531-00000FX2	10	19.94
Sensor board electronics parts	MEMS DIGITAL OUTPUT DUAL MOTION	LIS2DTW12TR	10	30.24
Sensor board electronics parts	CRYSTAL 32.0000MHZ 6PF SMD	XRCGB32M000F1H00R0	10	3.64
Sensor board electronics parts	CRYSTAL 32.7680KHZ 7PF SMD	SC20S-7PF20PPM	10	6.58
Sensor board electronics parts	CAP CER 1.8PF 50V C0G/NP0 0201	GRM0335C1H1R8CA01D	20	0.3
Sensor board electronics parts	FIXED IND 3.3NH 450MA 250MOHM SM	LQP03TN3N3B02D	10	0.47
Sensor board electronics parts	FIXED IND 2.2UH 1.2A 138MOHM SMD	LQM2MPN2R2MG0L	10	2.14
Sensor board electronics parts	FIXED IND 2.2UH 2A 168 MOHM SMD	DFE201610P-2R2M=P2	10	2.86
Sensor board electronics parts	IC FLSH 2MBIT SPI/QUAD I/O 8USON	MX25R2035FZUIL0	10	5.29
Sensor board electronics parts	LEAD FREE LOW TEMPERATURE SOLDER	4902P-25G	1	28.88
Sensor board electronics parts	4.7X3.5MM SMD LTSW	EVQ-P9102W	10	4.48
Sensor board electronics parts	BLUETOOTH LOW ENERGY 5.1 MODULE	DA14531MOD-00F01002	10	40
Sensor board electronics parts	3.5X2.8MM SMD LED	AA3528LEC	10	2.92
Sensor board electronics parts	LED GREEN CLEAR CHIP 0603 SMD	CS63CGT1C2MA	10	3.9
Sensor board electronics parts	LED AMBER CLEAR CHIP 0402 SMD	CS42EA2C	10	3.9
Sensor board electronics parts	LED GREEN CLEAR CHIP 0402 SMD	CS42EG2C	1	0.39
Sensor board electronics parts	LED RED CLEAR CHIP 0603 SMD	CS63CR2C2MA	10	3.3
Sensor board electronics parts	1206 GREEN SMD LED	L152L-GC	10	2.46
Sensor board electronics parts	1206 GREEN SMD LED	L152L-GC	10	2.46
Sensor board electronics parts	1206 RED SMD LED	L152L-LIC	10	2.09
Sensor board electronics parts	SWITCH SLIDE DPDT 300MA 6V	JS202011SCQN	10	5.3
Sensor board electronics parts	SWITCH SLIDE DPDT 300MA 6V	JS202011JCQN	10	5.3
Sensor board electronics parts	TACT 5.2 X 5.2, 1.5 MM H, 160GF,	PTS526 SM15 SMTR2 LFS	10	1.09
Sensor board electronics parts	TACT 5.2 X 5.2, 1.5 MM H, 260GF,	PTS526 SK15 SMTR2 LFS	1	0.11
Sensor board electronics parts	RES SMD 100 OHM 1% 1/10W 0603	RC1608F101CS	5000	55.7
Sensor board electronics parts	RES SMD 100 OHM 1% 1/10W 0603	CR0603-FX-1000ELF	100	0.68
Sensor board electronics parts	RESISTOR AEC -Q200	CR0603AFX-1000ELF	100	1.83
Sensor board electronics parts	CAP CER 10UF 4V X5R 0603	C0603C106M7PAC7411	100	8.36
Sensor board electronics parts	CAP CER 2.2UF 6.3V X5R 0603	C0603C225K9PACTU	50	3.98
Sensor board electronics parts	CAP CER SMD 0603 .1UF 10% X7R 10	C0603C104K8RACAUTO	50	3.44
Sensor board electronics parts	RES SMD 1.5K OHM 1% 1/10W 0603	ERJ-3EKF1501V	100	1.7
Sensor board electronics parts	RES SMD 10K OHM 1% 1/10W 0603	ERJ-3EKF1002V	100	1.7
Sensor board electronics parts	RES SMD 1K OHM 1% 1/4W 0603	ERJ-PA3F1001V	100	5
Sensor board electronics parts	RES SMD 4.7K OHM 1% 1/4W 0603	ERJ-PA3F4701V	100	5
Sensor board electronics parts	BATT RETAIN COIN 1 CELL PC PIN	BK-913	10	4.03
Sensor board electronics parts	BATT RETAIN COIN 1 CELL PC PIN	BK-913-TR	10	5.11
Sensor board electronics parts	BATT RETAIN COIN 1/2 CELL SMD	BH-67A-5	10	3.42
Sensor board electronics parts	BATT HOLDER COIN 20MM 2 CELL SMD	3074	5	6.55
Sensor board electronics parts	125KHZ-134KHZ, READ/WRITE ANALOG	EM4095HMSO16B+	5	22.05
AWS Credits	Credits for AWS services		1	0
Total Cost Incurred				$550.27

24. Project Budget



Lucas, Tarush, Meedm L+T, L+M, T+M, Team	Week of 8/29	Week of 9/5	Week of 9/12	Week of 9/19	Week of 9/26	Week of 10/3	Week of 10/10	Week of 10/17	Week of 10/24	Week of 10/31	Week of 11/7	Week of 11/14	Week of 11/21	Week of 11/28	Week of 12/5	Week of 12/12
Course Logistics																
Abstract	Team	🟪														
Project Website			🟪													
Proposal Presentation			Team	🟪												
Design Presentation					Team	🟪										
Design Documentation						Team	🟪									
Ethics Assignment							🟪									
Interim Demo											Team					
Final Presentation													Team	🟪		
Make Video Documentation															🟪	
Make Poster Board/Presentation															🟪	
Final Demo																Team
Hardware Implementation																
Preliminary Demo Hardware (Arduino based demo)			Lucas	🟥	▨											
Preliminary Demo Hardware (RPI Hub Server Set Up)				T + L	▨											
Parts Seletion - BLE module / MCU				Lucas	🟥											
Parts Selection - Load Cell + IC						🟥										
Parts Selection - RFID IC/Module							🟥									
Prototype Tag Schematic								🟥	🟥							
Prototype Tag Layout									🟥							
Prototype Tag Etched/Populated/Assembled										🟥						
Prototype Tag Mechanical Enclosure										Lucas						
BLE Implemented											🟥					
Power Testing Implemented												🟥				
Weight Tag Implemented													🟥			
Tag Revisions/Design Polish														Lucas	🟥	
Software Implementation																
figure out which protocol to use and start developing it				MeeDm												
Determine how to read in information using bluepy					🟨											
Hub Bluetooth Driver - run demo and read in info						L+M	🟧	🟧								
Preliminary RFID Analysis Code (Hub)							MeeDm									
Preliminary Weight Analysis Code (Hub)								MeeDm								
Debug any problems with recieving packages via bluetooth								🟨								
Integrate earlier code that analyzes values to decode packets of information									🟨							
Put packets of information into json file										MeeDm						
CAD design load cell										Lucas						
CAD design enclosure										🟥						
Error Handling										MeeDm						
Sensor Tag Firmware - Core Routine										🟥						
write code to organize arrays by microchip #											MeeDm					
Sensor Tag Firmware - BLE											Lucas					
Sensor Tag Firmware - Weight Sensing												L+M				
Load Cell Polling Firmware												Lucas				
Sensor Tag Firmware - Power Sensing / Rationing													🟥			
Data Analysis (statistical analysis to determine unusual consumption)												MeeDm	🟨			
User Alert System													MeeDm			
Converting Little Endian Byte Array to List Type														MeeDm		
BLE Data Tag to Hub Implemented															L+M	
Software Slack															▨	
Web App																
Design Diagram for info flow, research/trade study for stack				Tarush	🟦											
Set up basic web app framework					🟦											
Wireframe UI for the web app							Tarush									
Preliminary dummy data analysis (dummy food data)									Tarush							
Learn and integrate charts into Django									🟦							
Split data over various time ranges and varied factors									🟦							
Database setup										Tarush						
Set up login for different users and cats										🟦						
Set up directories and JSON files for the cats										🟦						
Send organizedinformation to global JSON file for RPi										🟦						
Email and alert notifications											Tarush					
Receive information in correct manner from Raspberry Pi											🟦					
Deployment on RPi											🟦					
Integrate code with actual users and data												Tarush				
Graphs dynamically update when visiting page												🟦				
Integrate buetooth into web application												🟦				
Allow full scaled modularity of the web application												🟦				
Debug any issues with receiving information/integration													Tarush	🟦		
Update models, forms based on later changes													🟦			
Web App Slack														▨	▨	
Integration																
Communication between RPi and web app							T + M	🟩								
BLE communication established between Tag and Hub														L+M		
BLE communiction between maximum # Tags and Hub															🟧	
Integration Slack														▨	▨	
Testing																
Preliminary System Tests									Team							
Various Final System Tests And Tweak																Team
Testing Slack														▨	▨	

25. Project Budget