Feederal Reserve - Automatic Fish Feeder

Team 43

Team Members:

Jack Croghan

Sarah Degen

Melanie Fuhrmann

Bella Leicht

Adan Maher

Brandon Mauss

Nathan Paskach

1 Introduction: Users and Requirements

1.1 Problem Statement

Campus Organizations Accounting and fish enthusiasts have problems monitoring the wellbeing of their fish over weekends and long breaks when access to the tanks is limited. While not in the office, the fish either can't be fed properly or have to be fed using slow "dissolving" fish food tablets that are over-kill for just a weekend. Most automatic fish-feeders release too much food at once, which is not ideal for fish tanks with only one (betta) fish. We will create a device that can deliver small amounts of fish food on a set schedule using an auger system. Also, it will measure the pH and temperature of the water. This will all be controlled by a secure web and iPhone application.

1.2 Intended Users And Uses

Oliver the Office Worker

1.) Wants a good work-life balance, wants their office fish to be well fed and cared for during weekends and vacations
2.) Needs a mechanism to feed and manage fish remotely so that they can enjoy being away from the office but know their "co-workers" are well taken care of
3.) They will be able to remotely manage the fish's feeding schedule, as well as check in on the general status of the fish tank

Francis the Fish Enthusiast

1.) Might forget to feed fish once in a while
2.) Not necessarily tech literate
3.) Would be able to continue to feed fish while on vacation and maintain their quality of care

Pete the Pet Store Owner (with aquariums)

1.) Doesn't want to spend time each day feeding all the fish tanks by hand
2.) Doesn't want to accidentally miss feeding one of them
3.) Would allow for regulated feeding of all fish so that none are forgotten and therefore starve

1.3 Requirements & Constraints

Functional

- Dispense up to 3 pellets of fish food per day and no more than 5 pellets/day
- Must be able to feed automatically on a schedule, manually, or on-demand via app request
- Must be able to store at least 7 days worth of food (~50 pellets)
- Must measure temperature within 1℉
- Must measure pH within 0.1
- Scheduled feeding must be able to be delayed by app user
- Visual indication/pop-up to indicate if a fish has been fed in a day or not.

- Must be able to indicate that fish has been fed by hand in the app
- Must not be able to be accessed by unauthorized users
- Powered via outlet plug in
- User login to secure app

Physical

- Attach to the top of a 1.5 gallon fish tank
- Size limit:
    - Fits on the back of fish tank lid (8x8in) or in the back gap of the tank
    - As small as possible so as not to distract from the fish

User Experience

- Must have a secure web app to display status of the fish tank, such as temperature, PH, and last feed
- App must be able to support connecting to multiple devices and displaying their information
- Aesthetically pleasing and intuitive design of app with themes being:
    - Purple
    - Red and gold
- Easy to refill
- Easy to remove and re-insert for tank cleaning
- Sleek, unobtrusive design
- iOS and Web applications
- Leave an access port/hole for manual feeding in design
- Notifications/indications in app for if the feeding failed or if the device is out of food
- Device should not be in front of the tank

Stretch goals:

- Variable amounts of food dispensed
- Salinity sensor
- Livestream of fish
- User log-in to be able to use ISU ID to login
- Fish pun name
- Backup battery in case of power outage

1.4 Engineering Standards

- 802.11 ac WiFi - for connecting to web app
- I^2C, SPI, UART - for connecting to sensors
- IEEE 1621-2004 - Power control of consumer products
- NISTIR 8259A - Cyber Security Baseline

2 Project Plan

2.1 Project Management/Tracking Procedures

Our major tasks will follow a waterfall management style because there are certain tasks we need to complete before we can start the next. An example of this is designing the feeding mechanism before wiring it into the control board along with the sensors. Inside each major task we will use an agile management style so that we can rework our designs as necessary until we have the result we want. Each part will not be perfect on the first try, so this allows us to make small changes that lead to a more efficient product before moving on to the next task.

We have been implementing a mixture of waterfall and agile. The mechanism team and the app team are working independently in a waterfall fashion until we reach the point in which the components are ready to connect. Within our teams, we are working in an agile style, where we hook up and discuss where we're at and if things need to be adjusted as we go.

We will use a spreadsheet (team) and Trello (app) to track the tasks, who should complete it, and when it needs to be completed.

2.2 Task Decomposition

App

- Research frameworks and SDKs (what we are researching)
- Design UI
- Create backend and data storage
- Create front end
- Connect frontend to backend
- Connect app with mechanism

Mechanism

- Brainstorm prototype
- Explore parts for prototype
- Order parts
- Feeding mechanism rapid prototyping
    - Basic workings
    - Working with power
- Develop firmware for ESP32
- Test sensors
- Build working food dispensing mechanism
- Assemble minimum viable product
- Redesign based on client feedback
- Final design testing

Presentation

- PowerPoint creation
- Who speaks on which subject

2.3 Project Proposed Milestones, Metrics, And Evaluation Criteria

Mechanism:

Food dispenser prototype within food dispensing margins/tolerance

- Week 8-10 - 5 pellets
- Week 11-14 - 3 pellets

Prototype circuit with sensors and microcontroller

- Get readings
- Within 1 °F
- pH within 0.1

Minimum viable prototype by week 14

App:

Functional Backend

- Sends/receives requests to/from mechanism
- Stores data for user use

Functional Frontend

- User gives at least an 8/10 score for usability and design
- Fish feeding manually/automatic
- Scheduling available to user

Presentation:

Week 14 - basics of project included

Week 15 - contains all necessary information and images to support words

2.4 Project Timeline/Schedule

Semester 1:

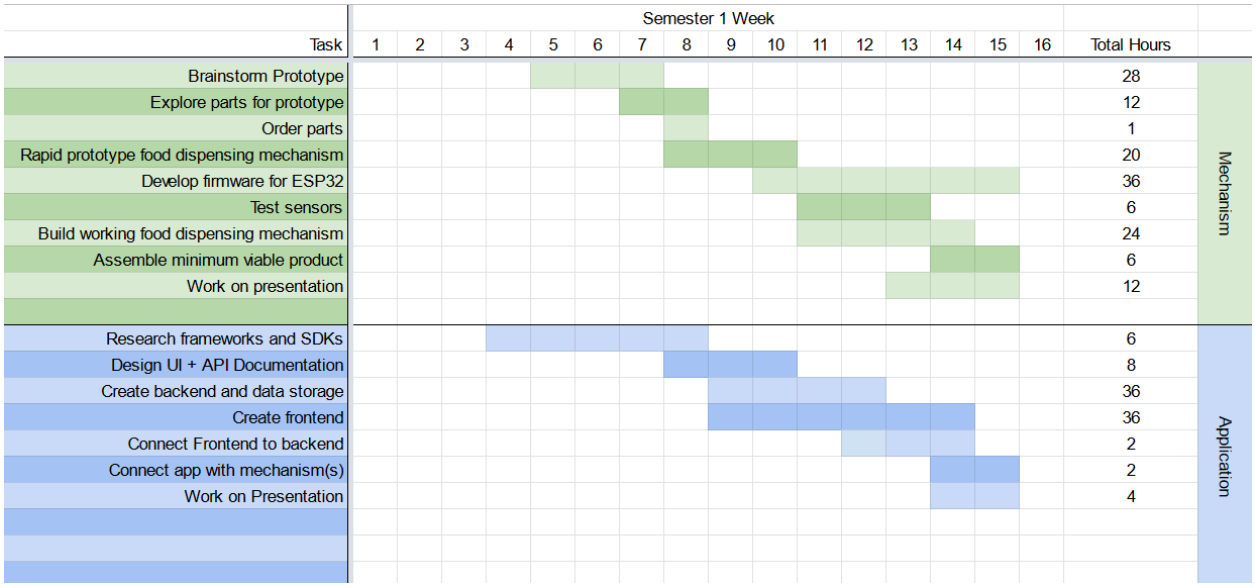

Figure 1: Semester 1 Gantt chart

Semester 2:

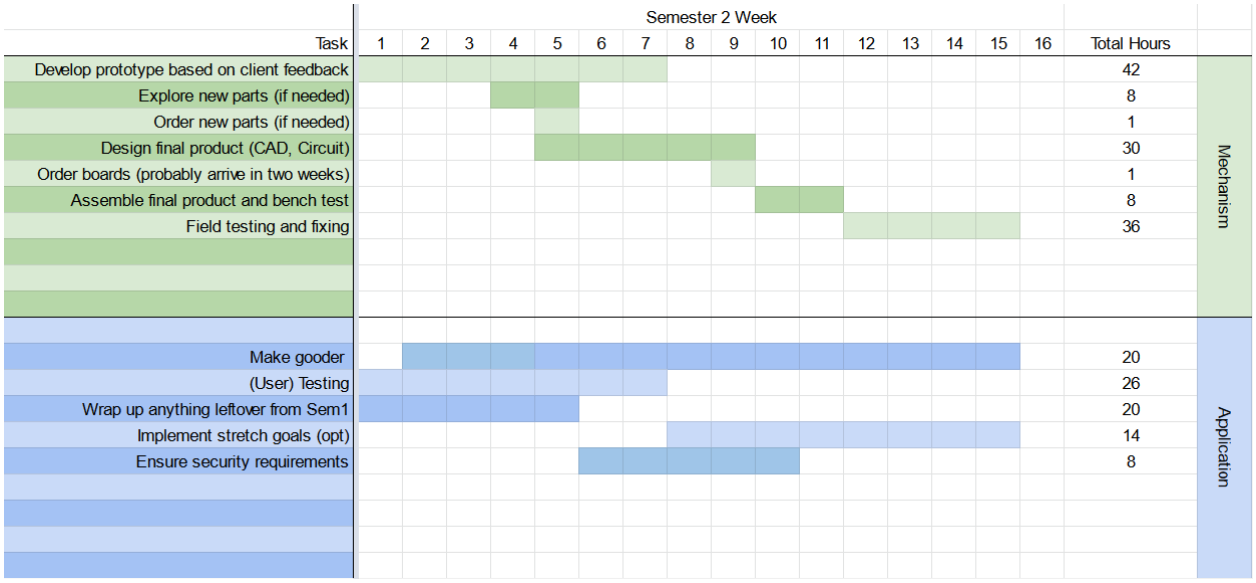

Figure 2: Semester 2 Gantt chart

2.5 Risks And Risk Management/Mitigation

Risk	Probability	Consequences	Mitigation	Notes
A part does not come within the expected timeframe	0.2	prototyping is delayed	N/A	
We fry a part without a replacement	0.05	need to order a new part, production potentially delayed	N/A	Depending on the part, like basic circuit components, we could get one from the ETG
Have trouble connecting software and hardware	0.9	a lot more work time to determine the problem and refactoring	Adhere to API between devices as strictly as possible	
Repeated rejections of UI Design	0.05	Delays development of frontend	N/A	Will require closer work with client if occurs
Flutter no longer supported	negligible	Have to switch to another platform of development	N/A	
Feeding mechanism falls outside of tolerance	0.3	will need to err on the side of less food to make sure we don’t overfeed	N/A	Test, test, test
Electronic components getting wet during semi-weekly cleaning	0.5	Damage to electronic components, failure of device	build a case for long term, put in ziploc bag for prototype testing	

Table 1: Risks and risk management

2.6 Personnel Effort Requirements

Task	Person Hours	Explanation
Prototyping Device Brainstorming Rapid prototyping Firmware Sensors Working food dispenser Minimum viable product Changes from client feedback	174 40 20 36 6 24 6 42	Lots of time breadboarding, wiring, soldering, programming, and then doing it all again.
Testing Device Client feedback Final bench testing	50 42 8	There are a lot of components that need individual testing, and then testing the full system once it's assembled
App Planning	14	A lot of the planning takes place alongside one another so it shouldn't be terrible in total time.
FrontEnd Development	36	Will take lots of time to build pages and to interconnect everything. May take some refactoring if the client is unhappy with the design or feel.
BackEnd Development	36	Should get set up early but might take a bit to get stable and perfectly done.
Connecting device to app	2	Should be a short task
Presentation	16	Will take collaboration from all subgroups to plan, create, finalize, and execute.

Table 2: Project hours

2.7 Other Resource Requirements

- Cooperation from ISU Solution Center for Active Directory use
- Server usage
- Cooperation from ISU IT to let the ESP32 onto the ISU network

3 Design

3.1 Design Context

3.1.1 Broader Context

The current fish feeders on the market dispense too much food for a single fish, which can make small tanks dirty. We are designing a smaller feeder for the Campus Organizations Accounting department. This will affect both the office workers and students who visit the office. Shown below are the considerations related to the project.

Section	Considerations
Public health, safety, and welfare	The most important thing that we have to consider when it comes to the safety, health, and welfare of those influenced by our project by far is for the overall welfare of the fish. This, in turn, creates an environment that those caring for the fish can be comfortable and feel secure in.
Global, cultural, and social	Our global cultural and social significance is fairly negligible. There aren't many ethical considerations that need to be given inside of the mainstream purview. The main impact would be for the wellness of fish to be more of a focus by individuals engaging with the product.
Environmental	Our design will use electricity from the wall at all times, but will save on gas from driving to the office to check on the fish. However, it prevents a more frequent changing of water in the tanks which will in turn save potable water.
Economic	The device will have an electricity cost associated with it, whether it is being actively used or not since it is taking pH and temperature reading often and always plugged into the wall. Our rapid prototyping supports tech businesses through the purchase of components.

Table 3: Broader context breakdown

3.1.2 Prior Work/Solutions

At the time of receiving this project, the client had described their process for trying to solve this problem before, and discussed the shortcomings of things on the market with respect to what they wanted for their tank. The main issue with on the market products was how large the volumes of food

they fed were. Due to this, we haven't been comparing our feeder with any one product on the market currently, but more so what our client had noticed about alternative options and what they wanted or didn't want.

Pros of our design	Cons of our design
● Small, precise amount of food dispensed ● Fits on small fish tanks ● Senses water temperature and pH ● Connected to phone/web app for easy access and schedule changes ● Less wasteful, as it doesn't use batteries ● Stores multiple feedings worth of food ● Won't muck up water like tablets	● Depends on server connectivity ● Relies on building electricity ● Does not include a protein skimmer ● Can only work with one kind of feed (pellets)

Table 4: Design contrast

Information about current feeders on the market was found at:
https://aquariumstoredepot.com/blogs/news/best-automatic-fish-feeder

3.1.3 Technical Complexity

- Mechanism
    - Motor with encoder - for dispensing food and detecting if there has been a jam
    - Temperature sensor - for sensing temperature (a client requirement)
    - pH sensor - for sensing pH (a client requirement)
- App
    - Client-server relationship - The app needs to be able to connect and communicate with the mechanism, so therefore we need a server as well as clients in order for this communication to occur.
    - iOS and Web app frontends - the client needs a way to interact with the mechanism and to know the status of all the sensors. We decided to use iOS and Web apps for the client to be able to access the app from their phone or their computers depending on which is easier for them at any given time.
    - Backend with a database - a server is needed to communicate with the client and make higher-level decisions for the mechanism. A database can be used to store historical data for the temp and ph of the tanks.
- Integration
    - Connect hardware and software with server connection
    - Send and receive requests from both app and mechanism

The current industry standard for fish feeding typically only works for larger fish and/or aquariums. Our system works for individual smaller fish in a compact environment – dispensing a small volume of food as necessary.

3.2 Design Exploration

3.2.1 Design Decisions

- User Authentication: Okta SSO - for ease of sign-in by clients and keeping out non-admins if we choose to add a livestream option for student viewers.
- Feeder design: Gumball/Drum - small amount of food dispensing is the whole reason for the project. We wanted to be picky about precision and size to best suit the client's needs and wishes.
- App platform: iOS and Web - whether we implement it for iOS or not completely changes what tools we can use and the overall look/feel of our app's frontend. iOS has their own design requirements/guidelines that we'll need to follow. We'd also like to implement it for the web, so the users can easily use the app on their phone or on their computer, whichever is easiest. This means to support the two, we need a SDK to manage both, which is why we chose Flutter.

3.2.2 Ideation

We identified potential options by brainstorming ideas and sketching them out on the whiteboard. We then asked questions of the person who gave the option if something was not understood.

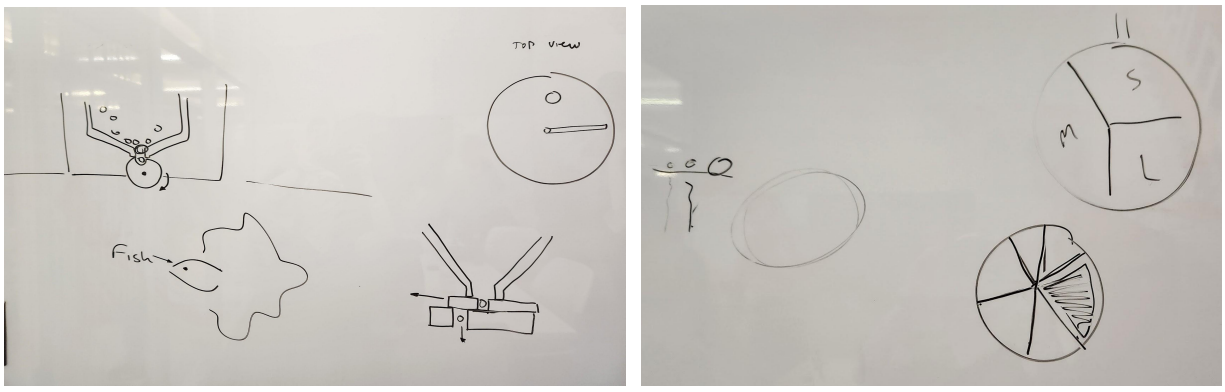

Figure 3: Initial design ideas

Feeder Designs:

- Coin slot – sorting pellets based on size to give a precise number of food pellets. Sorting works similar to how a coin sorter works, in that there are increasingly large holes on a track that food can fall into.
- Screw rotator - have the height between the levels on the screw be the size of 1.5 pellets, turn the screw a small amount of time to dispense the food
- Cereal dispenser - like the screw rotator but less accurate due to larger partitions
- Gumball machine - a scoop is used to move a group of pellets up higher than their resting position. At the apex of the scoop, there is a hole that's the same size as a pellet with the desired volume of food. This allows the desired volume of food to be delivered to the fish.

- Drum with food crevice- A vertical drum that catches one pellet from a hopper at a time as it rotates. Similar to the cereal dispenser, but only grabs one pellet at a time.
- Weekly pill tray- small compartments are individually loaded with desired amount of food by the user, and a slot of food is released for each feeding. Could be built into a wheel so the wheel simply rotates to line up the next day's food with the chute.

3.2.3 Decision-Making and Trade-Off

We decided to test the Gumball and Drum designs because:

- The weekly pill tray method would require too much of the users, and we'd like the user to have as little maintenance required as possible. Also, it would limit the user to using the feeder for only a week (7 days) max.
- The cereal dispenser will be too inaccurate.
- The coin slot won't be used because we determined the volume of food is more important than the number of pellets.
- The screw rotator accuracy depends highly on the amount the motor spins, which can change depending on the power provided to the motor and how well the motor responds. It also has a high probability of getting jammed by food dust.

Rather than locking into one design before testing, we've narrowed down the most likely options – gumball and drum designs – to experiment with them firsthand. Once we have working models of both we can do some more in-depth testing to make a more educated final decision based on evidence.

3.3 Proposed Design

3.3.1 Overview

The automatic fish feeder consists of a food hopper, dispensing mechanism, temperature sensor, pH sensor, iPhone and Web app, database, and a microcontroller that connects the app and database to the dispensing mechanism and sensors.

Food pellets go into a hopper which feeds into a dispensing mechanism. The feeding mechanism's motor rotates to select a single pellet of food from the hopper and delivers it to the fish. To get more than one pellet of food, the mechanism repeats this action multiple times. The action will also be repeated if the mechanism does not detect that a pellet entered the trough. The automatic feeding times and amounts can be configured with an iPhone or Web app. Also available in the app are the water temperature and pH readings read by the microcontroller.

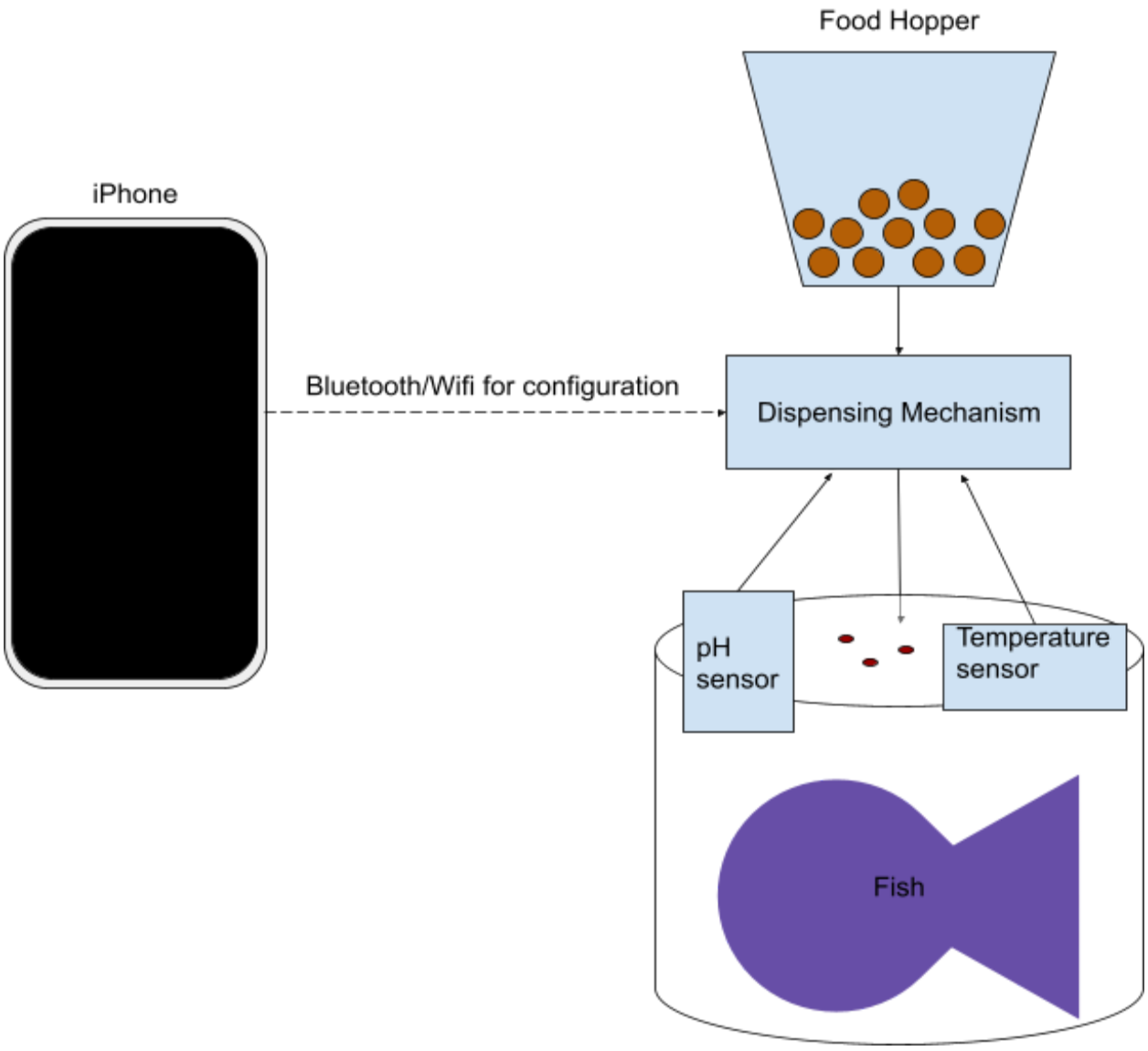

Figure 4: High level design

3.3.2 Detailed Design and Visuals

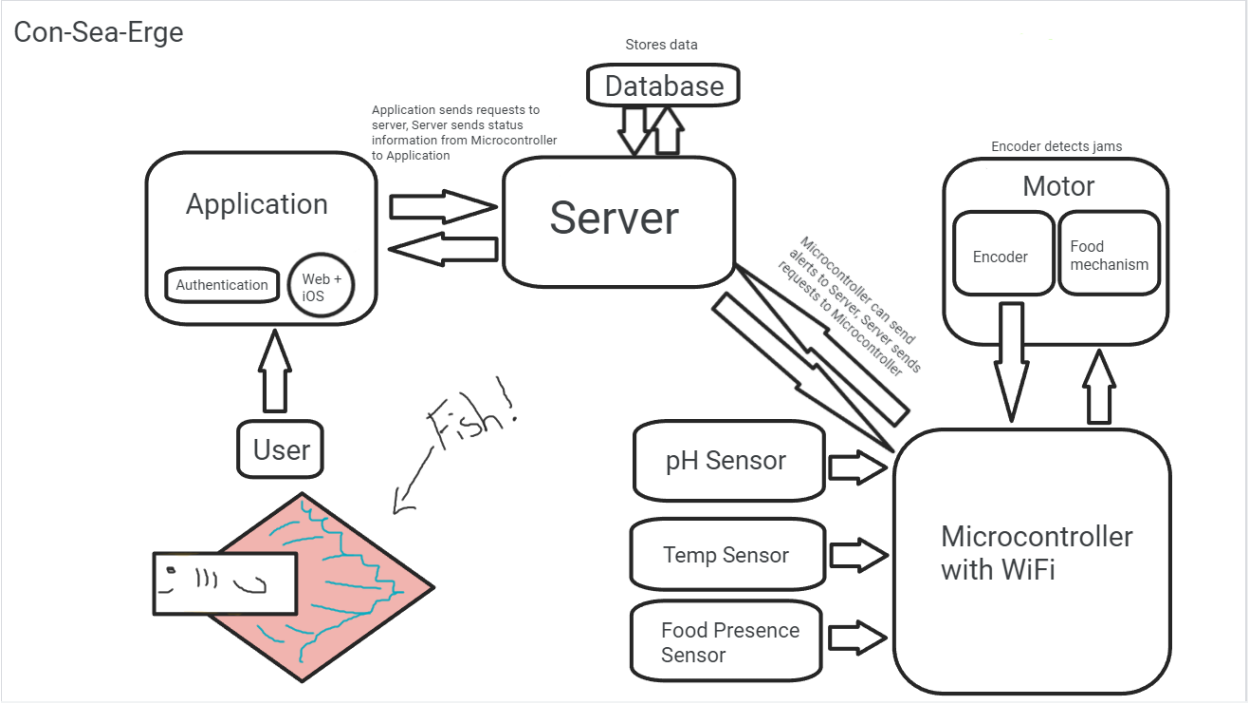

Figure 5: Detailed block diagram of design

Our team can primarily be broken up into two sub-groups, working on the two main components of our device: software and hardware/firmware.

The application will be available on Web and iOS platforms as we will be implementing it in Flutter. This app will be the primary interface between our product and the user, so we will be heavily focusing on our visual design and giving our users as many options as they want/need. We will be displaying all the sensor data collected by the mechanism, then giving options for schedule use/creation so the user can set when they'd like the automatic feeder to run. We will also give them options to inform the system that they fed the fish by hand, as well as telling the feeder to run on user request. We will be implementing alerts so that we can inform the user if the water is out of pH range or temperature range, if the mechanism fails in any way, or if the feeder is out of food. For the UI, we will have a home/dashboard page where the quick details about the fish tanks can be viewed, such as the pH, temperature, and if it's been fed so far that day. We may also include the option to hit a button on the dashboard for each device to feed their fish. There will be a schedule tab, where the user can see all created schedules, toggle which schedule should be running if any, and create a new schedule. When "Create a new schedule" is clicked, it will take the user to a new page, where they can select which days, if any, they would like the device to dispense on and what time they'd like it to occur during the day as well as name the schedule. This new schedule will then be displayed in the schedule tab/page. Tentatively, we plan to add a settings page where the user may be able to change the color scheme of the app to their liking. We are also tentatively planning to store previous status

data, so the user would be able to open a page and view the pH and temperatures over the past 50 days for each tank. This way if the user is interested in patterns with water quality, they can do so.

From the user end, accessing the device's setting/specifications will be a little bit more involved than a typical web-app requires. As a result of hosting the app on Iowa State's servers, the user will be required to either be on campus or connect to the Iowa State VPN for access. Past this requirement, user authentication is required to protect the security of both the fish and the user data; while the possibility of this hasn't been confirmed, we are hoping to implement Okta SSO to do that with the help of the Identity Team manipulating Active Directory roles.

The feeding mechanism will consist of a food hopper, a motor for actuating the dispenser, an encoder for sensing jams, and a food detection device using an LED and a photocell to confirm the presence of a food pellet. This is controlled by an embedded WiFi and Bluetooth enabled microcontroller. Automatic feeding times and amounts are configurable through the phone app. Also connected to the microcontroller are a temperature sensor and a pH sensor which will monitor the status of the fish tank's water. The microcontroller will put these readings up on the server to be scanned for anomalous values in order to alert the user.

3.3.3 Functionality

This design is anticipated to operate at the user level. However, it is possible to be developed further into a commercial product just as other fish feeders, that is beyond the scope of our project. At the user level, anyone with access to the devices and the application would be able to feed the fish either by hand or the machine, monitor the water conditions to decide when it might be necessary to change, and will alert the user on when the water conditions are no longer safe for the fish.

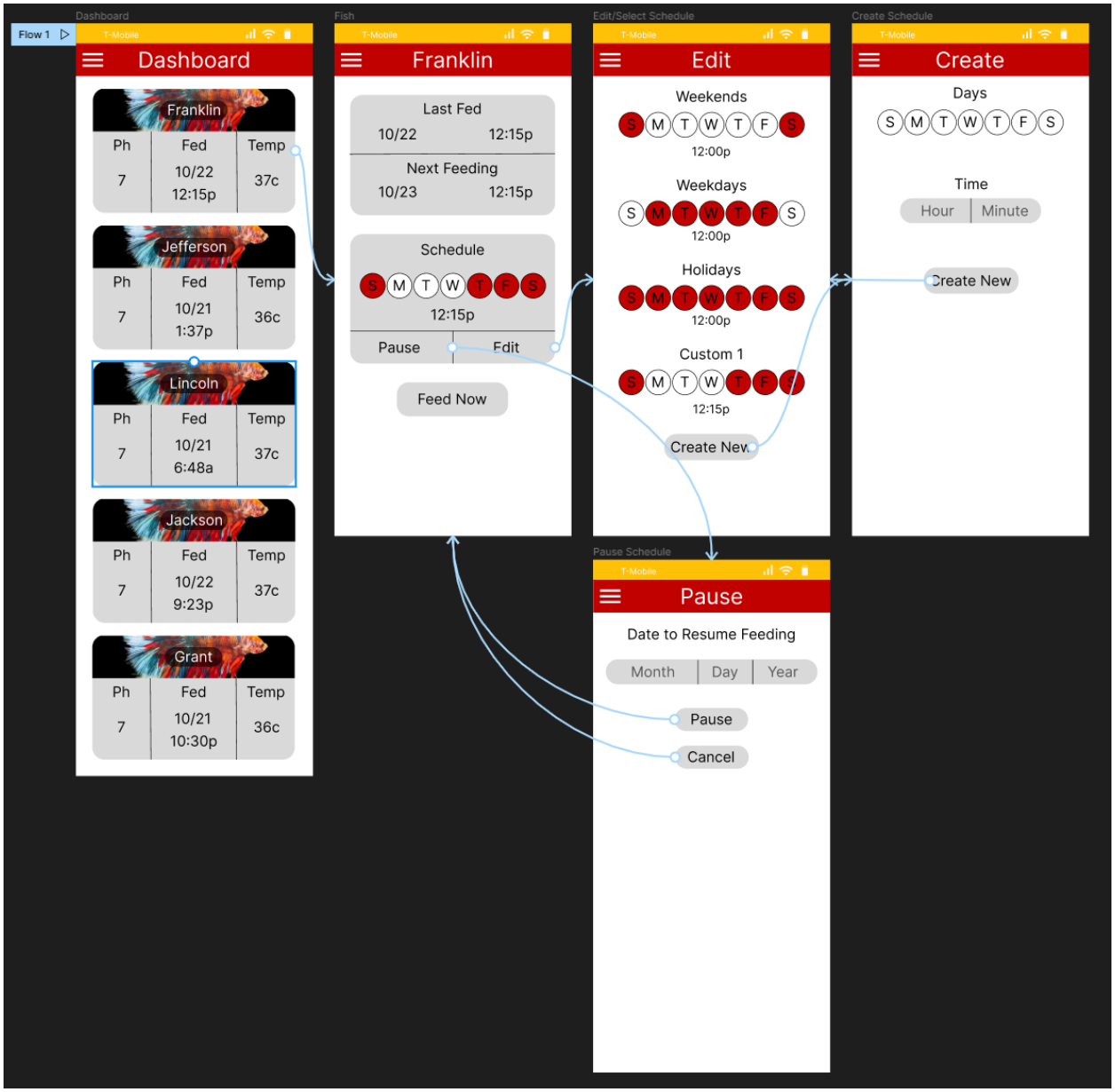

Figure 6: UI flow chart

After the user sets the schedule as shown above, the database and the microcontroller take over. When the scheduled time comes to feed the fish, the dispensing mechanism will rotate to remove one pellet from the hopper and drop it into the tank. This rotation step will continue until the necessary number of pellets have been dropped into the tank. If no pellet enters the dispensing area from the hopper, as registered by the LED and photocell, the mechanism will rotate again a max of two times to attempt to get another pellet. If all three of these rotations come up empty, the user will receive a notification through the app that the hopper is out of food. The user will also receive a notification if the motor in the dispensing mechanism encounters a jam which is read by the motor's encoder.

The temperature and pH sensors will take readings every few minutes and the microcontroller will send them to the backend. These readings will be visible for each tank in the app.

3.3.4 Areas of Concern and Development

As far as we can tell, this should perfectly fit the needs of the user. Our greatest concern is accuracy of the feeding mechanism which has yet to be tested. We need it to not overfeed the fish, so we will be testing a couple designs and troubleshooting them throughout the process to try to get our mechanism to be as accurate and reliable as possible.

One concern we have is the user's connection to the devices while at home. They would need to be connected through the VPN if they want to use the Web-based application. We know that it is possible to have the iOS app connect, but are questioning how many issues this could cause. To address this concern, we will continue to research and then test many times to determine what needs to change.

3.4 Technology Considerations

The technologies implemented within our design are the pH, temperature, and feeding sensors. The pH sensor is an off the shelf pH sensor which connects to a microcontroller over a certain protocol. It is a known-working design, but it is also the most expensive piece of the design. There are very few alternatives, though. For the temperature sensor, we will use either a thermocouple with a control circuit or a thermistor in a resistor ladder. The thermocouple would be abstracted to another peripheral through a communication protocol, but it is much more expensive and probably less waterproof. The thermistor is very cheap and waterproof, but will require characterization and calibration before it can be used.

3.5 Design Analysis

We have not been able to start building anything yet, but once we have parts we plan to rapidly prototype the feeding mechanism.

4 Testing

Our testing plan will start with separated tests for the software and hardware portions. We will test each component first to make sure everything responds as we are expecting and then test the combination of the parts into each subsystem. Once each subsystem passes its test, we will combine them all together and run system tests to ensure the entire design works as we want it to. We will then give the system to the Campus Organization Accounting Office for acceptance testing. The feedback we get will then determine our next steps and tests. All of our testing will be done throughout the project to ensure we are meeting project requirements.

4.1 Unit Testing

We will be testing the software using unit testing. We will use postman to mock up a server in order to test the frontend separately from the backend, and vice versa for the backend to test independently of the front end.

Frontend: We will test many smaller components in the frontend individually, with both unit tests, UI testing, and postman mocked server tests. With postman's mocked server, we will test all of the following to answer two questions: Does it send the request to the server in the correct format and when requesting data from the server and receiving it, does it display it correctly? We will attempt to both send information to the server from the frontend, and then try to send data from the mocked server to the app. We will also be able to test whether the app displays what we expect it to if a connection to the server is not made, so unit testing will be a great time to check if the frontend of the application properly displays some of the error messages we will be implementing.

- Adding schedules
- Deleting schedules
- Switching between schedules
- Naming or renaming schedules
- Adding pictures for a tank
- Naming/renaming the tank
- Multiple devices displaying correctly
- pH displays correctly
- Temperature displays correctly
- Last fed time displays correctly
- Changing the color scheme is saved by the server properly
- Color scheme is set at app opening by saved color scheme setting

For UI testing:

- Ensure all buttons navigate properly- Does each button navigate to the correct page
- Color scheme changes in settings correctly and displays that scheme in all screens for that user.
    - Does closing and reopening the app reset the color scheme
    - Does navigating to a different screen in the app reset the color scheme
- Buttons and displays are arranged as intended in both iOS and Web displays
    - Run iOS and Web apps to ensure a cohesive look and feel in both.

For security test:

- Require user authentication
- Ensure TLS layer is implemented

Backend: The components being tested will be separated by packages in the file hierarchy. This includes checking auth functions, functions with the devices, and functions for editing/creating

schedules. It is also important to test database connections and ensure data is being stored and retrieved correctly.

Hardware: The hardware units to be tested are the feeding mechanism and each of the sensors (motor encoder, food presence, pH, temperature). We will use the ESP32 with test-specific firmware to test these components by themselves. We will be looking at the values returned by the sensors to determine if they are near the expected values.

To test the temperature sensor, we will use a digital thermometer to read the temperature of a cup of water and compare it to the thermistor reading. The thermistor reading will then be calibrated by graphing the digital thermometer's reading against the thermistor's and using a best fit line to determine the calibration equation.

The pH sensor will be tested and calibrated using buffer solutions with known pH. We will start with the neutral buffer and use the adjustment knobs on the pH sensor to ensure this reading is 7. While we were testing the sensor with this method, we realized the adjustment knobs could only drop the reading to 7.9. Therefore an equation is necessary to drop it the rest of the way. Two more buffer solutions of pH 4 and 10 were used to create two more points for the best fit line. After calibration, two sanity checks will be done using lime juice and baking soda dissolved in water since these have different pH values than the buffer solutions, of about 2 and 9 respectively.

The food presence sensor reading is an analog value, but we will be comparing the voltage read when there is a pellet to no pellet to calibrate the levels we expect. The actual values do not matter, just the difference between them.

The motor encoder will be tested by running the motor without any contact with the food to have a base reading and then creating a jam to determine the output this creates. This will also test the hardware interfaces between the ESP32 and the peripherals.

4.2 Interface Testing

We will test the interface between the device and the app. We can test that the device can receive feeding time updates from the app by monitoring its debug outputs. Also, we can test that the device status makes it to the app by observing the values in the app and if they reflect the current state of the device. The conditions in which the device is in will be determined before the test is run, specifically the pH and temperature of the water. We will also test this connection by sending the request/command to feed and witnessing the device dispense real time.

4.3 Integration Testing

One integration is connecting the motor with the encoder with the food dispenser and food presence sensor and testing if that subsystem to reliably detect when a piece of food has been successfully dispensed. We will test if the dispenser can reliably transport one piece of food at a time via visual

confirmation, test if the food presence sensor detects it correctly every time, and if we can tell if the food mechanism is stuck by the reading from the motor encoder. The latter two tests will be similar to those in section 5.1 Unit Testing subsection Hardware.

Another is connecting all of the sensors to the ESP32 and testing that all are reporting correct values when polled. These tests are the same as those described in section 5.1 Unit Testing subsection Hardware.

A third is connecting the frontend and backend of the app, to ensure that all settings and data is passed correctly. In order to test the connection we will run the app with frontend and backend connected, then we will run the tests from the frontend, likely with similar tests or the exact same tests used with postman's mocked server. We can fill the backend with various information to make sure the expected information is displayed. We will also test that the information changed/saved in the front end is properly saved in the backend and that the information is retained and displayed upon the app closing and reopening.

Finally, we will connect the app and the feeding mechanism and repeat the previous test with real values from the device.

4.4 System Testing

When the app and the mechanism are fully connected, we can begin system testing. In order to test the full system, we will need to test that:

- When the "feed" option is pressed on the frontend, the mechanism dispenses the correct amount of food via visual confirmation
- When the feeder is empty, the frontend displays a error notification
- Waterproof testing of final hardware housing before putting the hardware in
- Error reporting on frontend of:
    - Food depleted
        - Using a full, partially full, and empty hopper
    - Dispenser jammed
        - Either by purposefully getting a piece of food stuck or applying pressure to the feeding mechanism's rotating portion to simulate a jam
    - pH out of range
        - Using lemon juice or baking soda dissolved in water
    - pH spike
        - By moving the pH sensor from water to lemon juice or baking soda water and back to the water
    - Temperature out of range
        - By placing the temperature probe in an environment that is hotter or colder than is safe for betta fish
    - Temperature spike

        - By moving the temperature probe from the correct temperature to an drastically different temperature environment and back to the original temperature
    - (Maybe power loss if we can get a big enough hold-up capacitor)
- When a schedule change is made, the device changes its dispensing timing to reflect the change
- If the user indicates that they want a scheduled feeding delay to feed the fish by hand, then the device changes its dispensing timing to reflect the change.

4.5 Regression Testing

Our team plan is each time an update is made to the application, the previously written tests will be automatically run by a gitlab CI runner.

4.6 Acceptance Testing

Acceptance testing will be continuous throughout our project. As we progress through the project and continue to develop our product, our clients will use each prototype. When the newest prototype is developed, we will question the client on any problems they incurred with the feeding mechanisms, the temperature and pH sensors, the application and any of its features. Then, when we begin developing the next prototype, we will attempt to include any suggestions they might have for their best usability. We have already completed some acceptance testing, as we have shown the client our various UI designs to receive feedback. This along with previous discussions, has helped guide our project and where we need to go with it, and special functionality the users would like.

4.7 Security Testing

Following the InfoSec guideline for information security, CIA, there are three things that must be tested and ensured in this project: confidentiality, integrity, and availability. On top of this, security testing is vastly different depending on our sub-group and will be broken down as follows:

Confidentiality	**Software**	On the software side, confidentiality is the ability for activity done by the client and our device to be private. We can ensure this by using transport layer security (TLS) in our HTTP communications, ensuring device settings are only available for reading by administrators of the device, and similar best practices. This can be tested using network sniffing techniques.
	Hardware	Luckily, there is no communication between the hardware side of the device and the user – there should be no need to ensure confidentiality in this case beyond the baseline design.
Integrity	**Software**	To ensure that communications to the device are received as intended by the administrator, we will use the TLS layer as stated in the confidentiality section. On top of this, we can have certain requirements of a user profile before allowing them access. We can use an offensive security penetration test to ensure that our measures are successful.

	Hardware	In this case, we recommend that the client/user lock the door when they're stepping away from the device – so as to ensure that there is no physical tampering of the device by visitors to the office. We can secure for accidental or unskilled tampering by light percussive testing, in cases such as a curious child or unknowing adult.
Availability	**Software**	While this is definitely going to be implemented and tested more at the software developing level, there are certain measures that we can take to add consistency to the device's communications. We'll be able to measure any changes and test their efficacy by network manipulation – creating low bandwidth scenarios, weak DOS attacks, etc.
	Hardware	Hardware availability is going to be similar to the testing done for hardware integrity – in that, more than likely, there is not going to be a case where users are maliciously tampering with the physical device. Testing will be done to ensure that minor accidents and curious prodding won't affect the device's functionality.

Table 5: Security testing breakdown

4.8 Results

4.8.1 Mechanism

The initial testing of the circuit returns values for the temperature, pH, and food presence sensors. The motor also turns when voltage is applied to it. Calibrated values for the pH and temperature sensor are shown in Figure 7. The readings after calibration fall within required tolerances.

```
pH: 8.6 temp: 68.94F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.27F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.30F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.20F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 68.94F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 68.96F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 68.86F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 68.94F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.20F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.18F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.22F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.18F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.27F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 68.96F    food: Not present      position: 2087  target: 2086    rpm: 2.39
pH: 8.6 temp: 69.32F    food: Not present      position: 2087  target: 2086    rpm: 2.39
```

Figure 7: Feederal Reserve debug outputs after calibration

Figures 8 and 9 are how we determined the equation used to calibrate the sensors. The expected values for the pH readings came from buffer solutions with specific pH levels. Since there was no documentation for how the pH sensor was supposed to be connected, we determined that we interpreted the sensor values backwards. Therefore, basic solutions were read as acidic and vice

versa. We chose to use a second order function to describe the calibration because the first order function was great for neutral solutions but did not give values within our tolerance of 0.1 for the acidic and basic solutions.

pH	
Actual	Expected
7.9	7
9.6	4
6.3	10

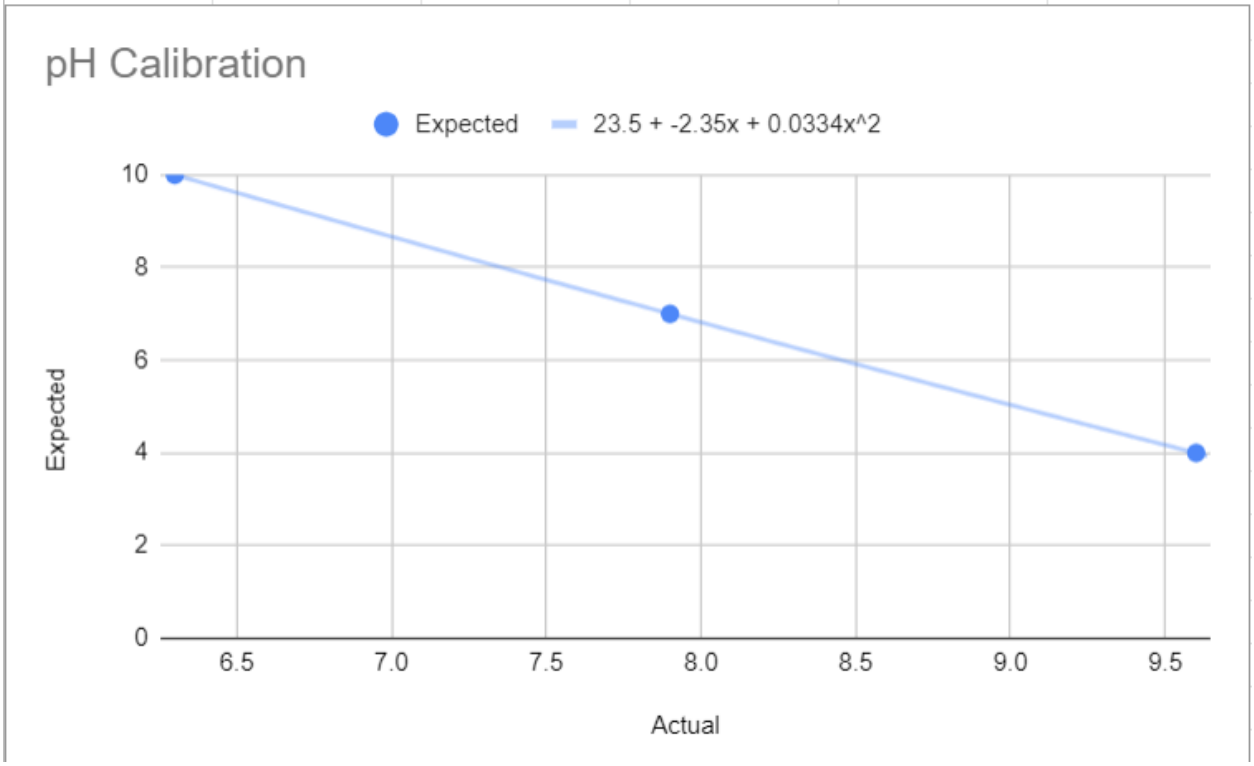

Figure 8: Feederal Reserve pH sensor vs pH buffer solutions

To determine the expected values for the temperature calibration, we used a digital meat thermometer that could read to the tenths place. Almost all of the sensor's measurements were within 2℉ of the digital thermometer. Even though the requirement for the temperature sensor is ±1℉, we read to the tenths place to limit the error in our equation.

Temperature	
Actual	Expected
101.3	103.8
100.4	102.3
97.2	99.1
94.4	96.1
91.3	92.6
87.5	89.5
86.3	88
55.2	60.9
85.3	87.2
81	82.6
76	78
74	76.1
70.5	72.9
68.1	70.7
66.2	69.5
66.2	69.4

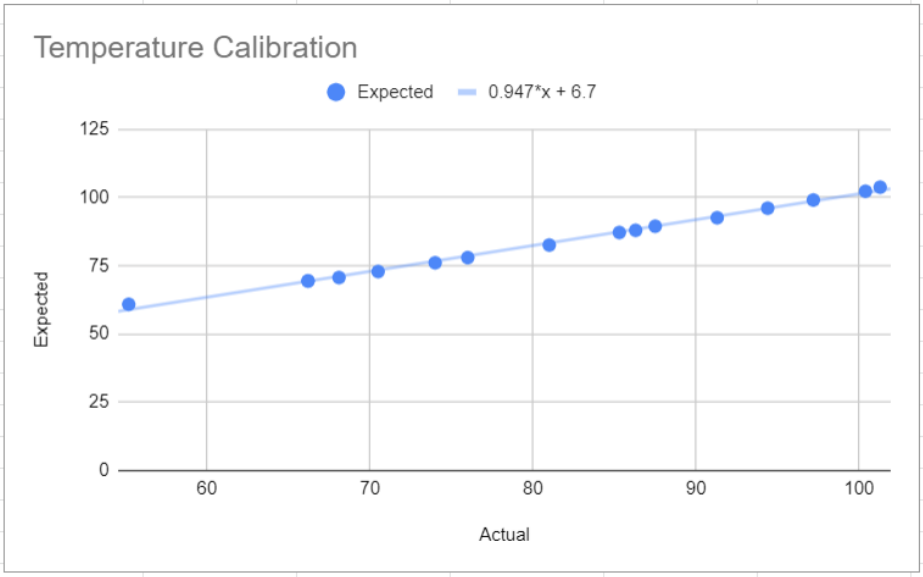

Figure 9: Feederal Reserve temp sensor. vs. digital thermometer

No testing has been done on accuracy or precision of the feeding mechanism yet. This will be done during the second semester.

4.8.2 App

The backend is at a point where every major feature has been created aside from the inclusion of websockets for the device connections. All other routes from the API documentation have been implemented and work for expected use cases. More work will need to be put in to ensure that everything is stable but as of right now it is functional. The job queue and ability to execute functions on a specified day at a specified time is also complete. The jobs are stored in a database so if the server ever loses lower or needs to restart, everything loads back in correctly and can continue to be used.

Frontend home page/dashboard has been created and is being implemented. There is still progress to be made in the page to make it more appealing to the client, but the results so far have been good.

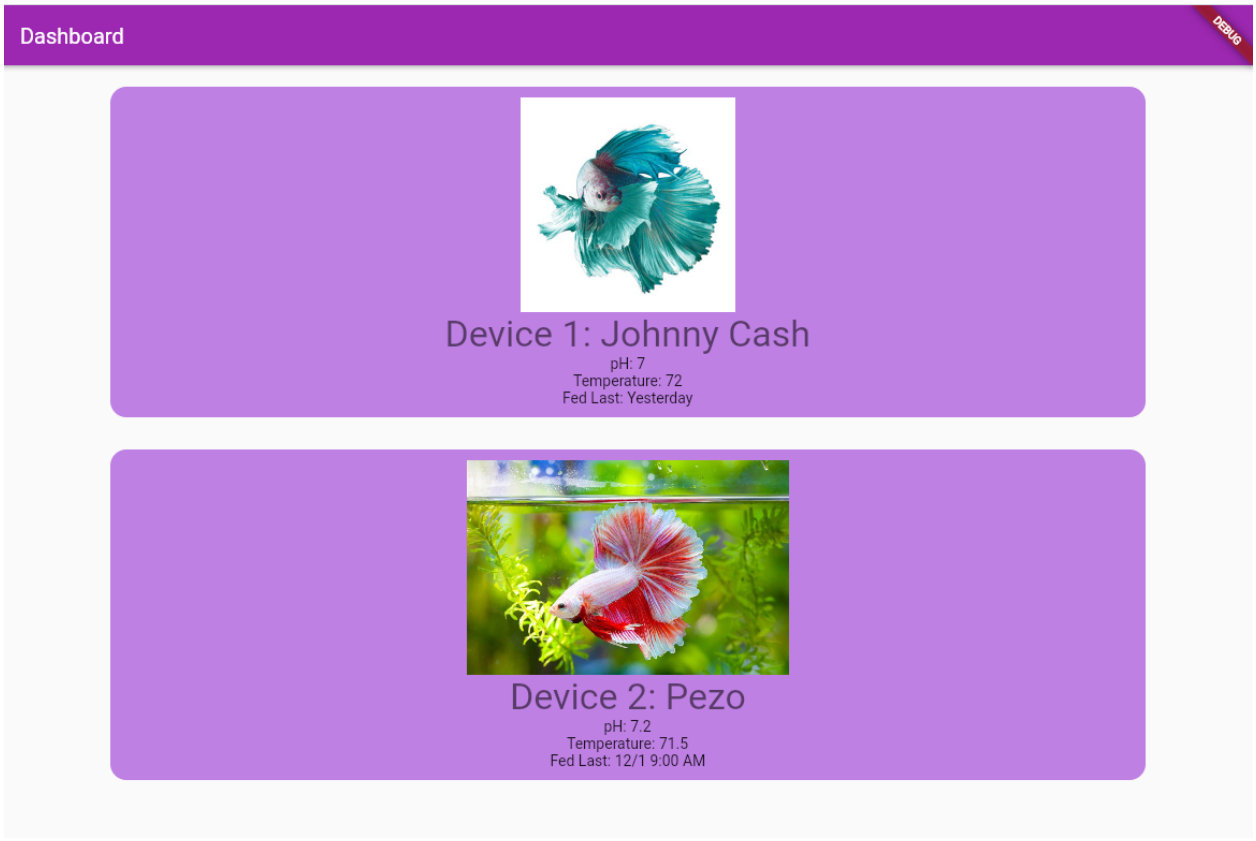

Figure 10: Current Dashboard view on Web