Figure 2 - Kayleb's saltwater fishtank after a collapse. He is just getting ready to drain it. Everything salvageable is below in the sump tank, but it's a grim and heart-breaking scene. Without constant monitoring it can be a gamble knowing what's happening inside.

Fish Sense:

Aquariums and Fishkeeping Simplified

Written by Braden Trigg | Sebastien Robitaille | Kayleb Stetsko | Eric Samer

Keeping aquatic life is an exercise in frustration tolerance – there's always something going wrong. It's all about finding the problem in time and balancing the ecosystem before things begin cascading in the wrong direction. Before you know it, you're stuck with whatever living creatures you've managed to save living out of sight in the sump tank while you attempt to figure out what to do with 85 gallons of dirty saltwater (figure 1). Not to mention the hours draining and refilling the tank can take.

Kayleb has been keeping fish and corals for most of his life and is aware of the many struggles associated with aquariums, having had to reset his tank multiple times. Saltwater aquariums can require a secondary 'sump' tank (figure 2) to act as a refugium—somewhere delicate algae and other organisms can reproduce. As filter feeders, corals that are housed in these tanks need a constant current in the water to enable them to feed off the organisms from the sump tank. Also, to truly thrive, coral require a narrow range for parameters such as temperature, lighting, and pH.

Disaster is an opportunist, and water being pumped between tanks provides ample opportunity for it to strike. Leak and water level sensing is required, as well as some way to switch the pumps on and off. Monitoring, controlling, and keeping track of everything manually can be both challenging and time consuming. Being tech savvy, Kayleb has automated some of these tasks, but with only

Figure 1 - A Sump is a refugium, somewhere for the nutrient providing bacteria to thrive and feed the life in the display tank. This is Kayleb's current sump tank, where he also houses the various pumps and other equipment that keeps the tank healthy.

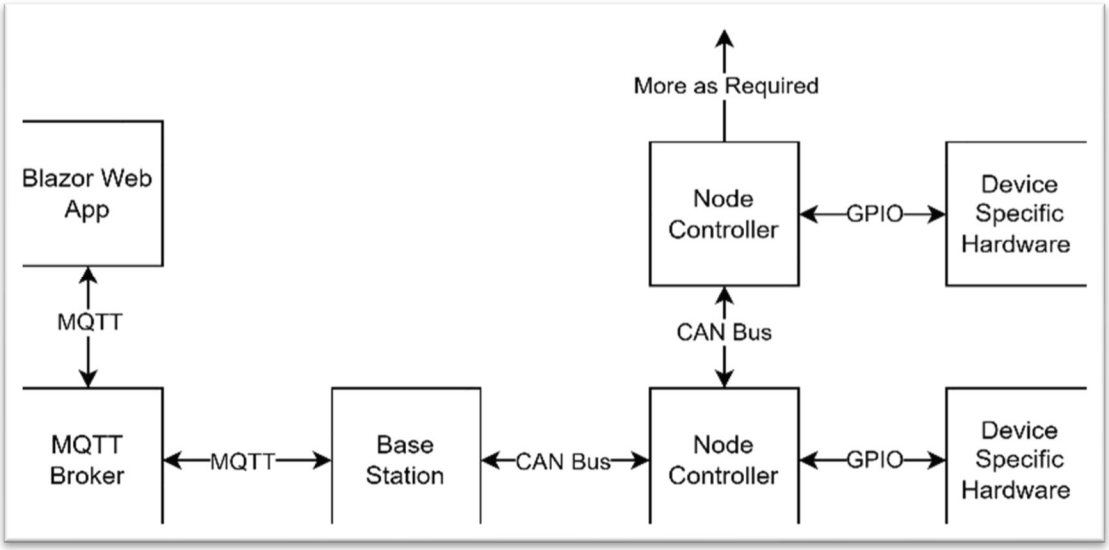

Figure 3 - The high-level block diagram of Fish Sense. Device specific hardware interface directly with our node controllers, which use CAN Bus to communicate with the rest of the network. Our base station, at the heart of it all, uses MQTT to reach out to our web app.

hobby-level knowledge, he had previously only been able to implement hobby-level solutions.

When the four of us began discussing ideas for our final project for our electronics technologist diploma, creating an aquarium monitoring and control system seemed like a natural choice. We were given a full semester for this project starting in September 2024. But not being okay with good enough, we got to work at the start of the year.

After researching the competition, we assembled a list of concerns with the existing solutions on the market. Almost all were a single unit, rigid in what they did, and didn't allow for easy expandability or partial systems; you really had to commit financially (usually to the tune of thousands of dollars).

So, to set our product apart from the rest, we focused on modularity and expandability.

Modularity allows for customization. Each ecosystem will have its own requirements and each fish keeper their own list of priorities. We needed our system to be user defined in terms of what devices and components keepers could add. Not just a black box of 'this is what it does,' but several black boxes you can use together.

Expandability allows for growth. Having gone through several tank expansions himself, Kayleb is all too familiar with how a simple fish tank can grow into something much bigger. (Remember that 85-gallon tank—that was after his wife insisted on cutting it back!) We built our system to allow users to grow their tank without being held back by the technology.

With these two tenets to guide us forward, we would just build a system to monitor and control everything, that's simple to use, expandable, and completely modular. Easy, right?

Our first real task was making the decisions about what technology would make our project a reality. ESP32's, CAN Bus, MQTT, Ethernet cords, and more. Many summer evenings were lost comparing and digging through datasheets.

System Design

To satisfy the modularity and expandability requirements for the project, we decided that each device in the system would have its own microcontroller. This would allow each device to operate independently, giving users the ability to easily pick and chose what devices they want in their system. We decided that a wired network would be preferable as we didn't want to deal with connecting so many devices to WiFi or having to power them individually. Lastly, we wanted the system to be accessible from a mobile app or similar, allowing users to monitor and control their system from anywhere.

With this system topology (figure 3) decided upon, we started fleshing out the technologies that would comprise it. We decided on two custom circuit boards for the system.

Our node controller would act as the general-purpose controller for each device. It needed enough GPIO to connect to various peripherals (sensors, controllers, etc). It also needed some way to connect to our second board, the base station. The base station would act as the controller for the system; it would have a WiFi connection and would interface all the devices with the mobile app.

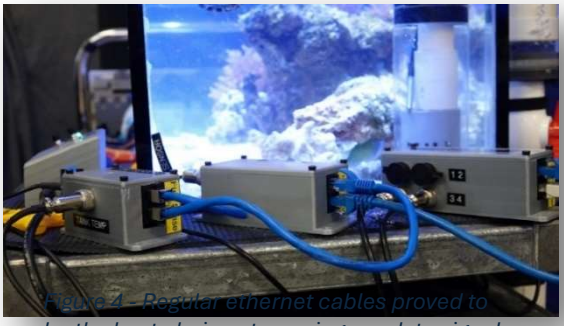

Figure 4 - Regular ethernet cables proved to be the best choice at carrying our data signal and power. They have enough conductors to carry the CAN Bus signal and power throughout the device network.

We settled on the ESP32 family of microprocessors for these two boards. The ESP32s are available in module form, with all the critical circuitry—such as flash and the RF circuits—packaged together which simplified our PCB layouts.

Next, we needed some way to connect the devices together and to the base station. I2C was briefly considered, but quickly abandoned due to its range restrictions and limiting controller target topology. After some further research, we landed on CAN Bus, developed by Bosh in the 90's, which is a serial communication protocol that includes built-in addressing and collision detection/avoidance. CAN Bus also has a longer range than I2C (hundreds to thousands of meters). While this was a much longer range than anything we needed in the short term, having the flexibility of creating larger and more distributed systems would leave our options open for future versions of this product.

Once we had a protocol picked out, we needed a physical layer to run the CAN Bus signal over. We wanted something reasonably durable and easy for the user to plug and unplug. Standard network cables, consisting of CAT 5 or 6 cable with RJ45 terminations, emerged as a clear choice for us (figure 4). These cables are super easy use and had a close enough characteristic impedance to the CAN Bus standard. They also have plenty of conductor pairs, one of which we used for the CAN Bus signal and the rest to supply power between connected devices.

The last piece of the puzzle was the mobile app. To limit our development complexity, we decided to create a web-based app that would run directly in a browser. This would allow us to target both mobile and desktop environments from a single codebase. We settled on the Blazor framework from Microsoft. Blazor is a part of ASP.NET and

allows for the creation of interactive web UIs using C# alongside HTML and CSS. We all have experience programming in C# from our studies at college, so we felt that it was a good fit for our team.

To facilitate the transfer of data between the base station and our web app, we chose the MQTT protocol. MQTT is a publish-subscribe network protocol usually used in bandwidth-limited situations, such IoT devices. The MQTT protocol involves an intermediate server called a broker. Clients—the base station and the web app, in our case—send messages to the broker on topics and can subscribe to topics to receive messages from other clients. We were able to use a free broker from HiveMQ.com. However, HiveMQ only has a three-day message retention period, so a different solution will be required to allow for long term use.

This brings us to September, when we started working on the project full time. With the major details decided on, we turned to getting our two custom circuit boards designed.

Node Controller

The node controller (figure 5) is a fairly simple board. The ESP32 C3 that we chose for the board has a dedicated CAN Bus peripheral (called the TWAI driver), but it still requires an external transceiver chip to create the differential CAN Bus signal. We used the Microchip MCP2551 IC for this purpose (The MCP2551 is now discontinued, but similar alternatives exist). For GPIO, we simply broke out the 13 remaining pins not used for the CAN Bus transceiver to header pins. These pins allowed us to create protoboard 'hats' to add any device-specific circuits to the node.

For power, we ended up with a three-rail setup. We receive 24V DC from the base station via the RJ45 plug. This voltage is then reduced to 5V using an integrated switching regulator to power the CAN Bus transceiver. Next, a linear regulator drops that 5V to the 3.3V required to power the ESP32 and most peripherals (some sensors and other devices use 5V though, so it's nice to have as well). We used the ESP32-C3-WROOM-02 module with an included PCB antenna. While we didn't end up using WiFi or Bluetooth in our final version of the project, the cost was the same as the module without

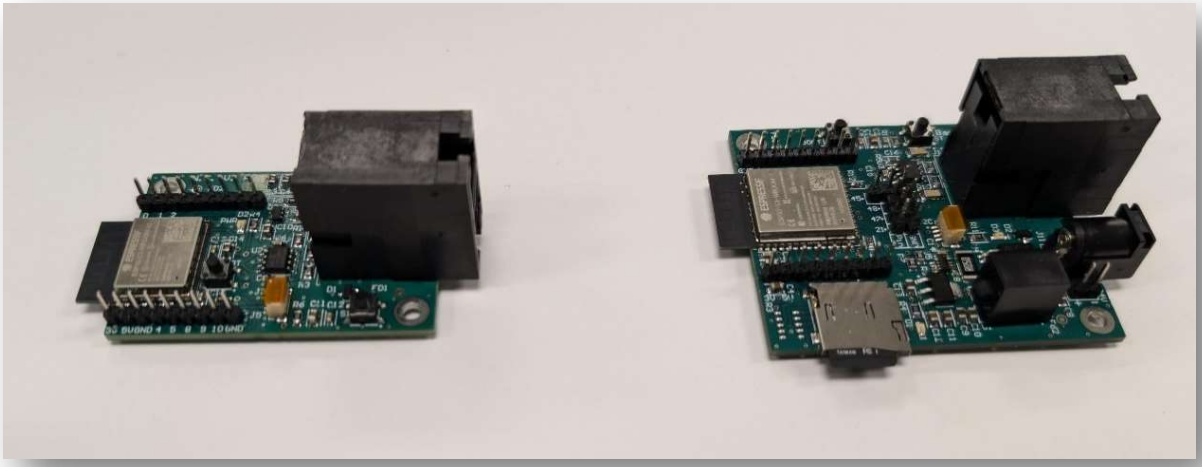

Figure 5 - Our two custom PCBs. (Left) Our node controller with an ESP32 C3. (Right) The more powerful base station, with the ESP32 S3, SD Card slot and barrel jack to connect to power.

the antenna, and we wanted to leave our options open in the future.

Firmware on the node controllers is also not overly complex. Sebastien, our programing wizard, decided that using C++ and the Arduino framework would enable us to use the existing Arduino libraries for various hardware interfacing, saving us a lot of firmware development time. We created a custom library to handle all the shared CAN Bus functionality. This library contained a function to send data to the CAN Bus network and a callback function that would be called every time the base station or another device on the network transmits data. Using this library, the node controller can send data to the base station, which then sends it to the web app. The node can also receive data, either from another device, or from the web app via the base station.

Figure 6 - Loud and bright, we ensured that our base station would grab your attention if there was something wrong with your tank. It can also send you an email alert if you are away from home.

Base Station

The base station (figure 5) is the main interface for our devices to connect to the web app. With the base station acting as a hub for the system, we also decided to add some extra functionality to it. First, we wanted to notify the user if there was an issue with their system (figure 6)—for example, if a leak was detected or the water temperature was too high. We added RGB LEDs and a buzzer to alert users in-person and gave the base station the ability to send emails to users if they were away from home. We also added two buttons and a small OLED screen to provide a basic user interface for configuring options such as the WiFi connection. The final function that we tasked the base station with was to log data from the devices to a SD card, so historical data could be accessed from the web app.

As one would imagine, these extra features required a more complex PCB design, and a more powerful processor; we chose the larger and more powerful ESP32-S3-WROOM-2 module. This module gave us more GPIO to work with, as well as a dual core processor that would allow us to split the slower WiFi and SD card operations from the more real-time CAN Bus operations.

The power supply for the base station is mostly the same as the node controller, except that the 24V power is provided by an external AC-DC adapter and barrel jack. We also put the 24V power through a high side power switch (ITS428L2), to give us better control of the network boot-up timing and a current/power monitoring IC (INA219) to monitor device network power usage.

Having not finished our user interface decisions when designing the board, we opted to just break out all the GPIO, with some paired with their own ground and 5V pins to power the LEDs or other potential peripherals.

Finally, with the help of our instructors and some late nights, we finished the board designs using Altium Designer and had them

manufactured by JLC PCB. We assembled the boards with a solder-paste stencil, our school's manual pick and place machine, and a vapour phase reflow oven. While there were a few minor issues with our boards (and a few bodges required), we made it through without needing any revisions. There are many things we would like to improve with these boards, with plans already in the works for subsequent revisions after this capstone project is completed.

With hardware completed, we needed to map out how the base station's firmware would handle data flow between the CAN Bus network and our web app. The easiest way to break it down is to follow the path of, for example, a temperature reading.

Starting in the device's node controller firmware, we receive a floating-point number from the temperature sensor. CAN Bus messages are broken up into two main fields, giving us eight bytes for our payload and 29 bits for an identifier. We use the identifier field to transmit the node's unique 8bit ID, as well as a 16bit message ID that indicates what data is being sent. The temperature value goes in the payload field, and this message is then transmitted over the CAN Bus network to be received by the base station.

Once the base station receives the message, an MQTT topic is created with the node and message IDs and a unique ID for the base station. The temperature data from the CAN Bus message, along with the time the message was received, is added to the MQTT message payload and sent to the broker. The web app can now subscribe to this topic on the broker and receive up-to-date temperature readings.

This whole process is repeated in reverse when we want to move data from the app to the devices, such as when we want to modify a temperature alarm setpoint.

Figure 7 - Here are the devices we chose for the first version of Fish Sense. Starting from the bottom right, going clockwise: Leak Detection, AC Control, Lighting, Temperature Sensing, pH Sensing, and the Base Station.

Devices

Come October, we finally had what you could call the 'brains' of the system in the base station, and a network of nodes to act as a 'nervous system;' there for it was time to start incorporating extremities, our devices (figure 7).

The first device that we started working on was our aquarium lighting control system. To ensure the comfort of our corals, we used two strings of high intensity LEDs—one white and one blue. Both strings contain eight series LEDs driven by MEAN WELL constant current LED drivers. We dim the LED drivers using a 0-10V PWM signal from a node controller, allowing us to create a customizable day to

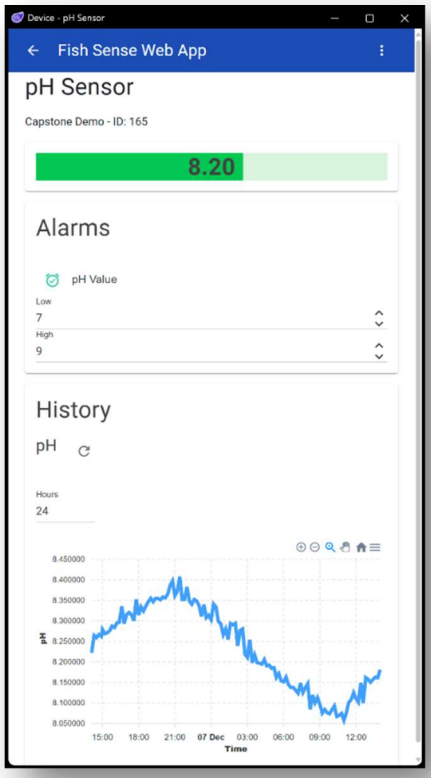

Figure 8 - The Fish Sense Web App, showing the details page of our pH Sensor. We have the alarms set should the pH fall out of the 7-9 range, and 24 hours of historical data is shown.

night routine and manually set brightness in the web app.

While there is an almost endless number of parameters that could be monitored in an aquarium, we decided that water temperature and pH, along with air temperature and humidity, would be good starting point. All these parameters can be monitored on the web app. Further, users can configure high and low alarm setpoints so that an alarm can be set off if any of the parameters are out of range (figure 8).

However, an alarm going off doesn't do much to mitigate the potential danger. Eric, our team's Red Seal Electrician, took it upon himself to design and construct an AC outlet control device. This would allow the system to take action in response to these alarms automatically toggling outlets as required. For example, turning off a pump if the water level is too high, or turning on a heater if the temperature is low.

We decided place all the components for the device (outlets, control relays, and the node controller) in a divided 5-gang electrical box. The divider meant that we could provide some physical isolation between the AC outlets and relays and the lower voltage node controller. We also chose a dual-channel relay module with a built-in AC to DC power supply for the relay coils. Using opto-isolators for the relay control signals meant we could fully electrically isolate the node controller from the relays and outlets. Finally, a GFCI cord end was used, since the outlets were near water.

We designed and 3D printed a bracket to hold the node controller in one blank cover plate, and some RGB LEDs—to act as status indicators—in a second one. This allowed everything to fit into any 3 ½ inch deep electrical box, leaving our options open for in-wall mounting in the future.

Eric constructed the AC outlet device with three pairs of outlets: one always on, one connected to a normally open relay, and one connected to a normally closed relay. This allows the user to decide if they want a given outlet to be on or off in the event of a control system failure. We also added a current clamp so that the power usage of all the connected AC devices could be monitored in the web app.

As this is arguably our most complex device, it is still in the early stage of development. Although we were able to complete this version (figure 9), in the future, we would like to add more outlet configurations, per-outlet power monitoring, and more complex alert triggering and scheduling options in the web app.

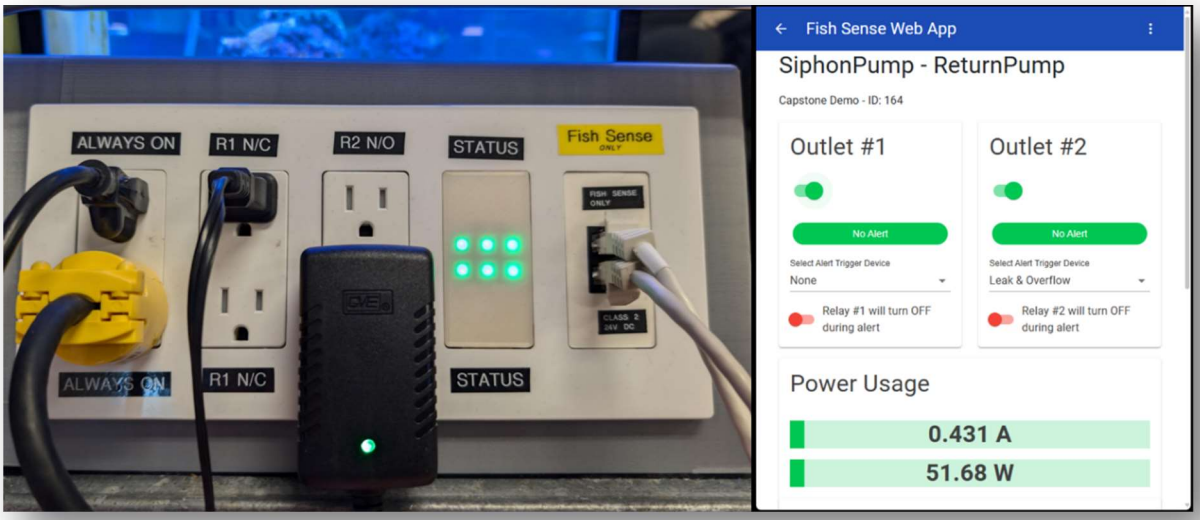

Figure 9 – (Left) The finished AC Control Device provides controllable outlets for our tank. (Right) The web app device page allows users to configure how alarms effect each outlet and view the power consumption of the system.

From the very beginning, Kayleb's dream of a completely modular solution for his aquariums promised to be not only an exciting and fun project, but something that could truly help make the joy of fish keeping accessible to a larger audience. Over the past four months we have put our all into Fish Sense, and we believe we have made something truly novel. We set out to simplify the aquarium experience, making it as easy to grow your monitoring system as it is to grow your aquarium; with the ease of set up and modular, expandable design we have achieved, we feel we have accomplished this goal.

That said, we would like to continue to improve our product Fish Sense and have many plans for the future. We have more devices we would like to implement: Atlas Scientific's line of environmental sensors, a variable speed pump controller, and an automatic feeder. In addition, we would like to make improvements to our existing technology, including a full redesign of the lighting system and new versions of the node controller and base station PCBs. We also see many other use cases for Fish Sense outside of just aquariums. From hydroponics in agriculture to fish and shellfish farms, we believe our monitoring and control system is adaptable to any set of needs.

Until then, we know that the fish and coral in our demo tank (figure 10) will live the easy life in their tank managed by Fish Sense.

Figure 10 -We are extremely proud of our completed demonstration tank. Almost every aspect of it is controlled by Fish Sense, with even more controls planned in the future.

Visionary

Kayleb Stetsko

The visionary with a lifelong love for fishkeeping. Above all he is a loving father, who would do anything for his children. Beyond that, anything alive he treats like his own, including the fish of the world. His love for technology and engineering got him to where he is today. His beard is legendary.

Electron Rancher

Eric Samer

A Red Seal Electrician focused on safe and effective power solutions. Eric worked as a Red Seal Electrician until an accident forced him off his feet and has since decided to follow his love of technologies. He has a huge passion for space and science (and his cats). His beard is also legendary.

Code wizard

Sebastien Robitaille

Our programming expert, skilled in integrating complex systems. He is the youngest of the group, but probably the smartest and his drive to learn and get things done is probably going to change the world. In his spare time, he helps track the bird populations and plays with computers. His beard, by the way, is legendary.

Author

Braden Trigg

The organizer and writer, translating technical details into clear communication. After spending 16 years in hospitality working as a Chef he has since moved into IT. Braden is a passionate video gamer and is looking forward to furthering his passion in electronics, computers and science in general. He can’t grow a beard but is still legendary.

Contact

Name	Email
Eric Samer	eric.samer@gmail.com
Braden Trigg	triggbraden@yahoo.ca
Sebastian Robitaille	sebastien@robitaille.info
Kayleb Stetsko	kayleb_s@hotmail.com