IoT Wildfire Detection

Group 4

Aracely, Jose, Julia, Logan, Quinn, & Salvador

I. Executive Summary

Wildfires are a statewide problem in California. Every year they destroy large swathes of our natural areas because of how quickly they can grow out of control. This means we need a better way to handle them. Our product, the IoT Wildfire Detection Device, is a solution to this problem. One of the biggest hurdles in wildfire prevention is knowing exactly where and when they start. This is especially apparent when they start in remote wilderness areas uninhabited by humans. By utilising various sensors, cellular networks, and Microsoft Azure's IoT platform, we will be able to detect when and where wildfires happen and notify the proper authorities in record times. This will create faster response times and give more detailed information about the fire, giving firefighters every advantage we can.

It is true that our firefighters have always been able to protect us, but the world is changing and fires are getting more dangerous every year. We used to have a set fire season that we could prepare for each year, but with climate change, science has been showing that soon it might be fire season all year. These higher temperatures from climate change also mean that fires can grow out of control faster than ever before. Just last year large parts of Santa Cruz county were evacuated because a dry lightning storm started a wildfire in one of our state parks. If we had information about exactly when and where the fires started there is a chance that we could have contained it better before it grew so far out of control. At the end of the day, we might need more than just better detection. But we need to start somewhere and detection is affordable, effective, and most importantly, it is achievable.

II. Ethics Statement

1. We agree to be bound by the code of ethics of the IEEE.
2. We pledge to ethically source the electronic parts necessary to realize our design project.
3. We pledge to design an alarm system that benefits all parties of interest equally with no preferential treatment toward any one person or group.

IEEE Code of Ethics

We, the members of the IEEE, in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members and the communities we serve, do hereby commit ourselves to the highest ethical and professional conduct and agree:

I. To uphold the highest standards of integrity, responsible behavior, and ethical conduct in professional activities.

1. to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment;

2. to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;

3. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;

4. to avoid unlawful conduct in professional activities, and to reject bribery in all its forms;

5. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, to be honest and realistic in stating claims or estimates based on available data, and to credit properly the contributions of others;

6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;

II. To treat all persons fairly and with respect, to not engage in harassment or discrimination, and to avoid injuring others.

7. to treat all persons fairly and with respect, and to not engage in discrimination based on characteristics such as race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;

8. to not engage in harassment of any kind, including sexual harassment or bullying behavior;

9. to avoid injuring others, their property, reputation, or employment by false or malicious actions, rumors or any other verbal or physical abuses;

III. To strive to ensure this code is upheld by colleagues and co-workers.

10. to support colleagues and co-workers in following this code of ethics, to strive to ensure the code is upheld, and to not retaliate against individuals reporting a violation.

Adopted by the IEEE Board of Directors and incorporating revisions through June 2020.

Table of Contents

1. Introduction

1.1. Need Statement

We need to reduce the damage that is caused by wildfires.

1.2. Goal Statement

Minimize the delay for:

1. Firefighters to begin suppressing wildfires when they start
2. Local inhabitants to be notified of a fire near or in their community

2.3. Design Objectives

Design a system that can:

- Accurately detect fires in an ar ea with 4g available that spans at least 5 acres
- Display accessible, accurate and up-to-date information regarding the fire risk throughout the area
- Send firefighters and residents prompt and accurate notifications of an imminent fire
- Be easily set up and maintained (e.g. has a low battery replacement cost)
- Be quickly, cheaply, and sustainably manufactured and assembled

Design Objective	**Units**	**Target/Range**
Fire Detection Accuracy	%	> 90%
Fire Risk Accuracy	%	> 80%
Fire Alarm Alert Delay	Minutes	< 1
Time Between Battery Replacements	Months	> 3
Range of Fire Detection System	Acres	> 7,000
Cost to Manufacture System	USD	< 7,000,000
Time to Manufacture System	Months	< 3
Time to Assemble System	Months	< 1
Design Constraint	**Units**	**Target/Range**

Wireless Connectivity Required in Area	Protocols	4g LTE

In the above table, “system” refers to the whole network of devices covering a single area. In the above table the numbers stem from Wilder Ranch State Park, which is 7000 acres.

Our $1,000 per acre number comes from CAL Fire estimates that 4.3 million acres burned in 2020, and it cost roughly $12 billion in damages. This means that the average cost per acre was just over $2,700. We think that undercutting that by over half should be sufficient to make our product cost effective.

1.4. Personas

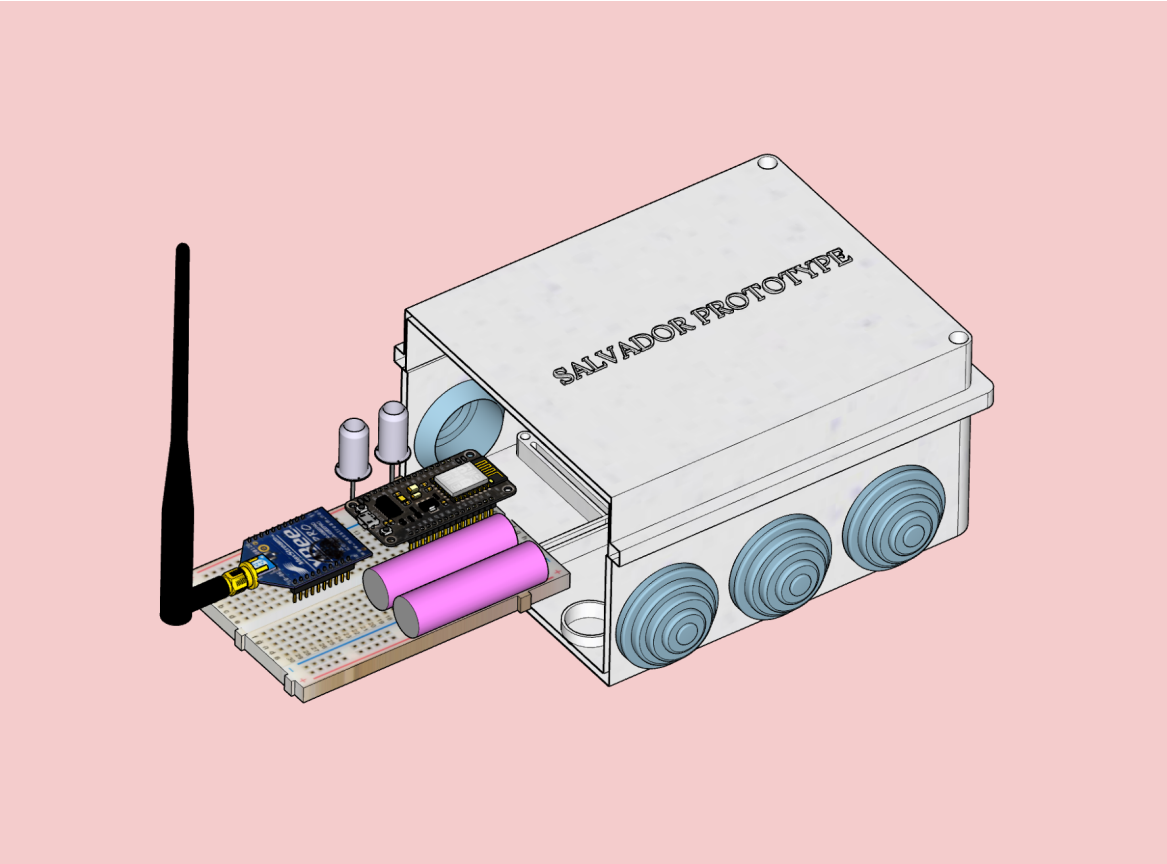
Name: Tim Sexton
Role: Program Manager for the Wildland Fire Research Development & Applications program. His responsibilities include management of the Wildland Fire Decision Support System as well as facilitating technology transfer of new science associated with wildland fire to the field.
Background/ Context: He previously served as a Type 1 Incident Commander on Great Basin IMT 1 and as a Type 2 IC on Rocky Mountain IMT #2. He remains active in large fire management, serving on Area Command and the Command and General Staff of Type 1 IMTs. Tim has a Bachelor’s Degree in History from Boise State University and a Master’s Degree in Fire Ecology from Oregon State University. Tim started his fire career as an engine and fuels crewmember on the Shasta-Trinity NF at Weaverville Ranger District
Profile Link: https://wfmrda.nwcg.gov/about-us/meet-wfm-rda-staff
Interview: https://www.youtube.com/watch?v=U4uaqAWMLtw

Name: Scott Stephens
Role: Professor at UC Berkeley and researcher for the Fire Science Laboratory conducting scientific research and providing academic training in the fields of wildland fire science, ecology, and resource management.
Profile Link:
https://ourenvironment.berkeley.edu/people/scott-stephens
Research Lab: https://nature.berkeley.edu/stephenslab/
Ted Talk: https://www.youtube.com/watch?v=2r7JI6zVwf0

Name: Ian Larkin (Client)
Role: Unit Chief of the San Mateo Santa Cruz Unit
Goal/Mission: The management and protection of California's natural resources; a goal that is accomplished through ongoing assessment and study of the State's natural resources and an extensive CAL FIRE Resource Management Program. CAL FIRE oversees enforcement of California's forest practice regulations, which guide timber harvesting on private lands. Department foresters and fire personnel work closely to encourage and implement fuels management projects to reduce the threat of uncontrolled wildfires. CAL FIRE Foresters promote conservation and the importance of our trees and forests to Californians of all ages.
Profile Link: https://www.coastsidefire.org/about-the-chief

2. Design

2.1. CAD Prototype



The CAD model above will be further developed once we finalize the electrical configuration of the device as well as the design of the enclosure. It currently shows the battery, 4g module and ESP32 connected together.

2.2. Proposed Enclosure/Placement Design

Key:

1) Solar panel
2) Dust filters and scaffolding
3) Electronic components and battery

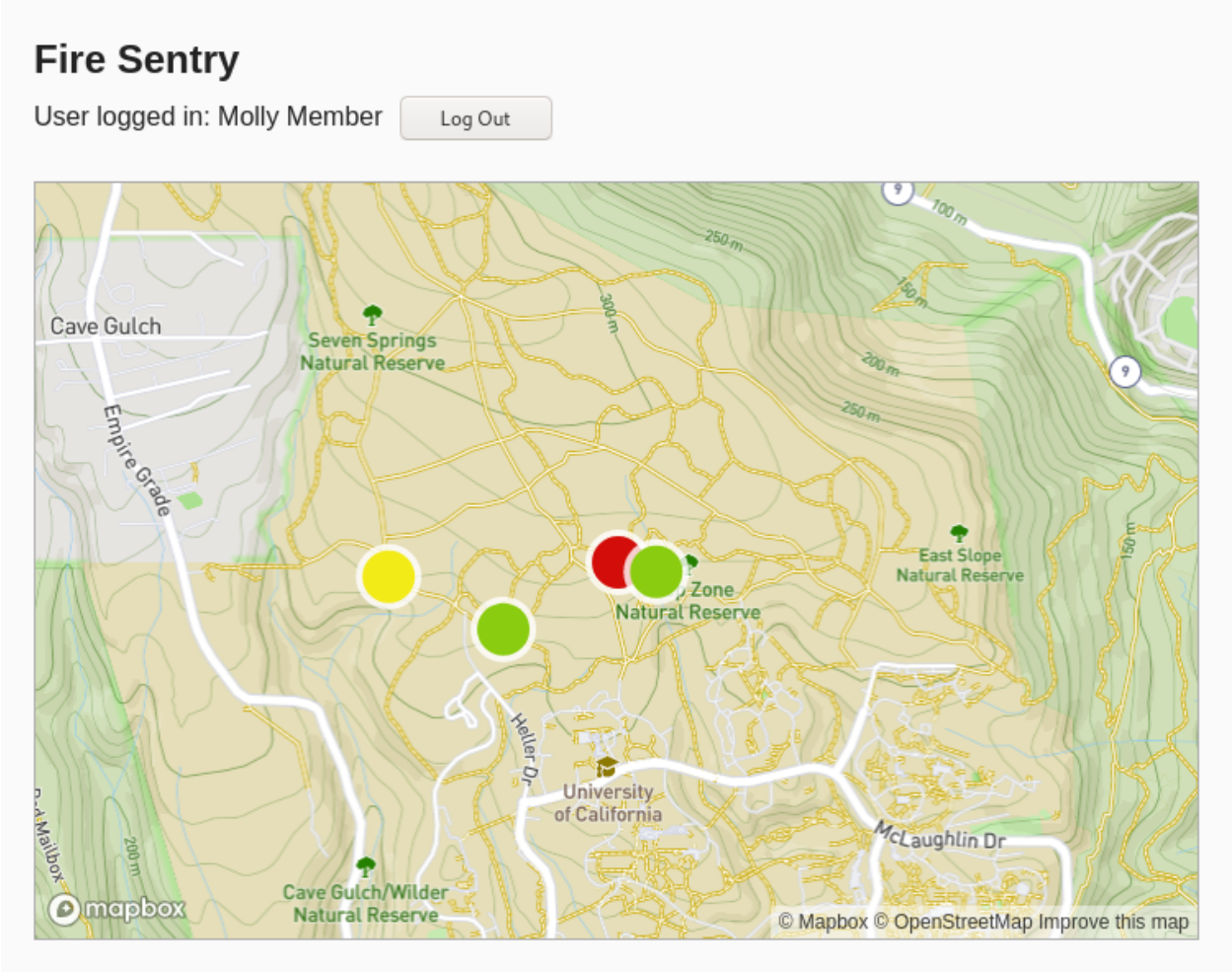
This sketch shows the design of our enclosure, which is shown as a cross-section. It was created to address some concerns that come from its intended deployment in a remote, outdoor environment.

In order to function correctly, each device in the network has to be ventilated so that it can detect changes in CO2 and humidity in the surrounding air, but its components also need to be protected from rain and dust. The enclosure's only opening is a vertical tunnel that allows air to rise into the device while blocking rain from the front. The tunnel would also contain scaffolding for stability as well as filters to block out dust.

We decided to fasten each device to a tree because wind could move the devices out of their intended positions, giving the monitored area uneven coverage, and it could also cause them to get damaged. The enclosure in the sketch has two tabs on the top and bottom that could be drilled or stapled to the tree.

A solar panel on top of the enclosure would harvest energy, reducing the cost of battery replacements. The solar panel's mount would be adjustable during deployment to ensure that each device could harvest as much energy as possible.

2.3. Web Application Prototype



This is a screenshot of our web app's prototype. The app contains a map of the area our devices would monitor, with each dot corresponding to a sensor device. The color of the dot represents the fire risk at each device.

2.4. Design for Manufacture & Assembly

Design for Manufacture

- To make our design easy to manufacture we attempted to use parts that were readily available on the market. Our carbon and rain sensors and microcontroller came from Banggood, an online distributor. Our new temperature and humidity sensor is arduino compatible so it is available on many online retailers such as Adafruit. The part for us to source was our battery because we could not find

them in stock in December, however, they also became readily available early this year.
- For any parts to our design that were not going to be made by a large manufacturer, we discussed that we would design them to be 3D printed so that we could either buy a 3D printer and generate them ourselves or go to a company and bulk 3D print them. This would include our casing to hold all of the sensors and microcontroller, as well as the weather-proof enclosure we would screw into trees to keep our devices in place.

Design for Assembly & Maintenance

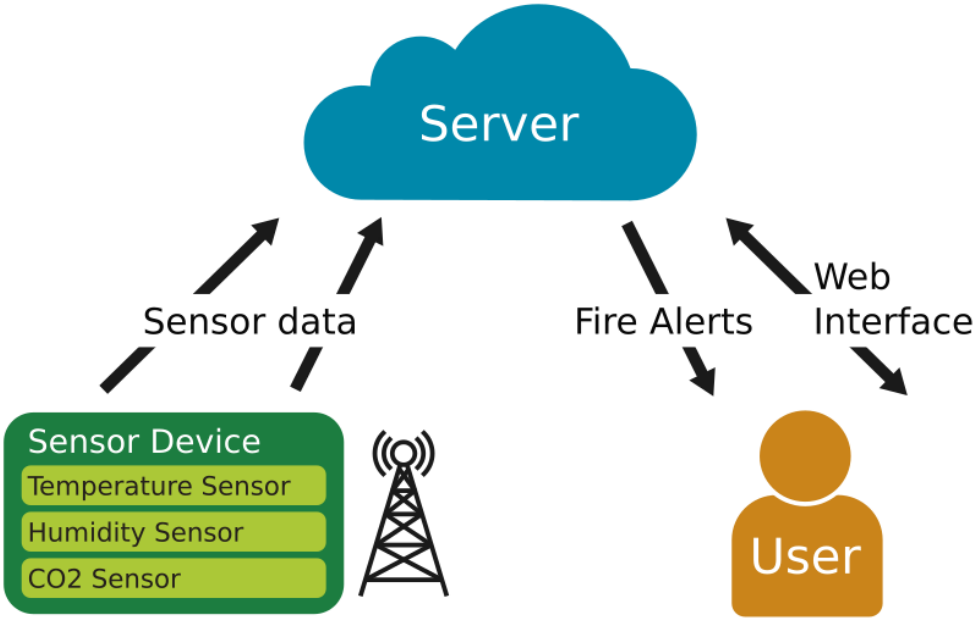
- To make our devices as easy to assemble as possible, we have ordered parts that do not require any special wiring or assembly after we receive them. We will also plan to use standard eighth inch screws for securing our enclosures, and long quarter inch screws to attach the devices into trees.
- To make attaching our devices to trees easy, we will have two holes for screws on the main enclosure. There will be one on top and one on the bottom which will be used to secure the whole enclosure to trees.
- The main two types of maintenance that our group has predicted are replacing broken components or dead batteries. To make maintaining our devices as easy as possible, we will include an easy to open flap on the back of the enclosures (side facing the tree). Having the opening on the back ensures that the device will not open unless it is removed from the tree first, and that we can access each of the components without worrying about damaging any of the internals.

2.5. Life Cycle Assessment

We chose an ESP32 model development board that has a working current of 200mA and 3350mAh 4.87A batteries to power the device. We have plans on using a solar panel to recharge the batteries and ultimately extend the lifespan of the device. This aspect of the project is still under development.

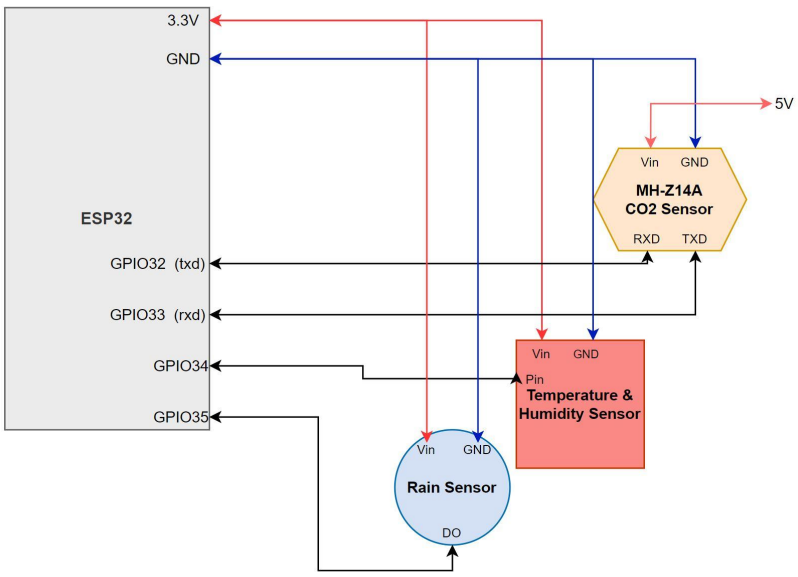
3. Diagrams

3.1. High Level System Overview



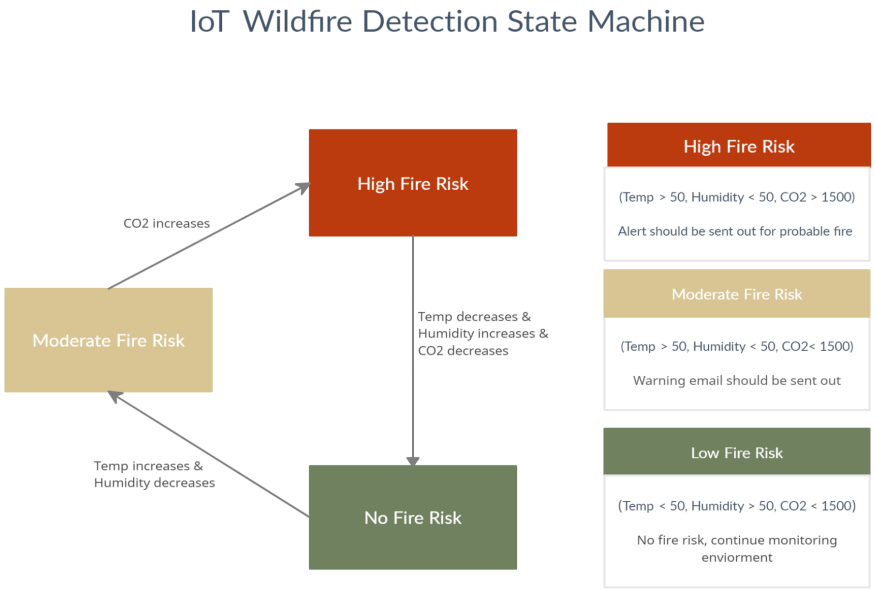
To fulfill our Need & Goal Statements, we decided that this design depicted by the block diagram was best for us. We have sensors that are responsible for reading external input from the surrounding environment. We have a microcontroller responsible for sending data from our sensors to our web server. The web server will notify interested parties of the potential fire hazard in the device's immediate environment.

3.2. Wiring Diagram



Above is the wiring diagram for our hardware component of the project. We have the sensors connected to the microcontroller. We also needed to supply our carbon sensor with an external 5V as the ESP32 only offers 3.3V.

3.3 Fire Risk State Diagram



There are three states in our detection system: no fire risk, moderate fire risk, and high fire risk. In order to transition from one state to another, a sensor's data needs to change, indicating whether there is an increase or decrease in fire risk.

4. Technology

4.1. Hardware

- Temperature/Humidity sensor: For our initial design we attempted to use the chip YL-38 connected to the sensor HR202 . While attempting to use it, we were unable to find the proper documentation, and had issues integrating it. In the next iteration, we would be using the *SHT40*, a sensor from Adafruit, which has many tutorials online and easily available documentation.
- Carbon sensor: For our carbon sensor we used the *MH-Z14A PQM NDIR Infrared Carbon Dioxide Sensor Module*. We chose this sensor because it was relatively cheap compared to its competitors but still offered a good enough detection range for us.
- Rain sensor: To try and preserve power we used the *HL-84* Rain sensor for Arduino. This sensor was implemented so that we could determine when it is raining and so we can know when to go to sleep to preserve power.

- Microcontroller: For our microcontroller we are using the *ESP32* with a soldered-on cellular slot. One of the main reasons we chose the ESP32 is because it was designed for IoT applications. It has support for low-power operations, can operate in a wide temperature range, and has good support for internet connectivity.

4.2. Microcontroller code

- Technologies used:
    - C
- Our microcontroller has code with two main functions, reading data from the sensors and sending it to our web server. We chose to use C because it allows us to interact with GPIO pins and handle serial communication to read from our sensors, while also allowing us to send https requests to our web server.

4.3. Server Backend

- Technologies used:
    - API: Node.js, Express
    - Deployment: Heroku (prototype)/Azure (final design)
- Employs authentication using JSON Web Tokens
    - Motivation: ensure that the data stored on the server is valid and private, and prevent false alarms from being triggered by a malicious client
    - Sensor devices receive a unique web token when they are manufactured and keep them throughout their lifespan
    - Users of the web app receive a temporary web token if they log in with a recognized email-password pair
    - A valid web token needs to be sent in the header of every request to the server for it to be accepted
    - There are different user roles that access the server, and they each have their own set of HTTP endpoints that they are authorized to access. The table below lists which roles can access each endpoint.
- API:
    - Request inputs would be sanitized and checked that they match the API specification, but this has not been done in the prototype. The DELETE /nodes endpoint was also left out of the prototype.

API Specification

Endpoint	Description	User Roles	Request header	Request Body	Response Body

POST /nodes/location	Provide location of sensor node	Machine	Authorization: Bearer <API key>	{"lat": Latitude of device, "lon": Longitude of device}	None
POST /nodes/data	Submit data from a sensor node	Machine	Authorization: Bearer <API key>	{"temp": Temperature, "humid": Humidity , "wind": Wind speed, "co2": CO2}	None
POST /authenticate	Authenticate user	None required	Content-Type: application/json	{"email": User's email, "password": User's password}	None
GET /nodes	Receive data about all nodes	Member or Admin	Authorization: Bearer <API key>	None	{"nodes": [{ "lat": Latitude of node, "lon": Longitude of node, "fireRisk": Fire risk }] }
DELETE /nodes	Deauthorize a node	Admin	Authorization: Bearer <API key>	{“id”: ID of node to delete}	None

- Data handling workflow:
    - Our final design would use a fire detection method that would prevent as many false-positive fire alerts as possible and have high server performance. In the future we would discover a suitable method for this by looking at current research into the topic. In the current iteration of our design, on the endpoint handler for POST /nodes/data, this new data is added to the database and also passed into a fire detection function. The function returns the fire risk as a value between 0 and 100, with a larger value representing a higher risk. An alarm is sent if the risk is higher than a predefined threshold value. We would have to tune this threshold by testing this part of the system.

- Fire Alerts:
    - We could use an already-existing channel for sending evacuation alerts
    - Alerts could prompt first-responders to examine the location of interest for a fire, and they would be prompted to either confirm it or flag it as a false-positive

4.4. Server Frontend

- Technologies used: HTML, CSS, JavaScript, D3.js, Mapbox GL JS
- Workflow of app:
    - User logs in with form, which is sent as a POST request to /auth
    - If the user is successfully authenticated, the response of POST /auth is stored in local storage. The map then loads, and the app sends a GET request to /nodes every 2 seconds to fetch new data about the sensor devices. This data changes the color and position of the dots according to the fire risk and location of the corresponding sensor node
    - If authentication fails, the app displays a pop-up with the message “Incorrect email or password”
- Modifications for final design:
    - User should be able to deauthorize nodes from web UI
    - When the web token times out, the app should stop sending requests and ask the user to log in again

4.5. Database

- Technologies used:
    - Mysql, Javascript, Azure
- Our database uses a MySQL database in Microsoft Azure to hold our data for future use. It is set up with a large database to hold a table for each of our devices. The table has fields: An entry ID number, the date it was entered, and a field for each of our sensor readings as shown below.

	Field	Type	Null	Key	Default	Extra
▶	entryID	int	NO	PRI	NULL	auto_increment
	sampleTime	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
	temp	float	YES		NULL	
	humid	float	YES		NULL	
	carbon	float	YES		NULL	
	wind	float	YES		NULL	

- In our current system we are only using our database for storing the data we collect. We hope that in the future if we collect enough data, we will be able to use it to improve our fire risk thresholds and maybe turn our system into a predictive system as opposed to the reactive system it currently is.

5. Simulation

5.1. Simulating devices

In order to test our server/web application independently from our devices, we used Postman to send HTTPS requests to our web server. Using different API keys we were able to simulate multiple devices to see how they were represented in our web application. Using this method we were also able to test our thresholds for changing the color of the dots from green to yellow to red very easily.

5.2. Simulating web server

While waiting for the web server to be hosted with a public endpoint, we created a basic HTTP server that matched the specification for our main server except without the API key security. This allowed us to test sending messages from our microcontroller to the web server independently of the main web server.

6. Functional Prototype

6.1. Implemented features

For our functional prototype we created an end-end demo. We were able to connect a rain sensor and carbon sensor to our microcontroller. The temperature and humidity sensor we ordered did not work for us, so we replaced it with a potentiometer for the purposes of our demo. Our device was set up as shown in figure 1 using a breadboard.

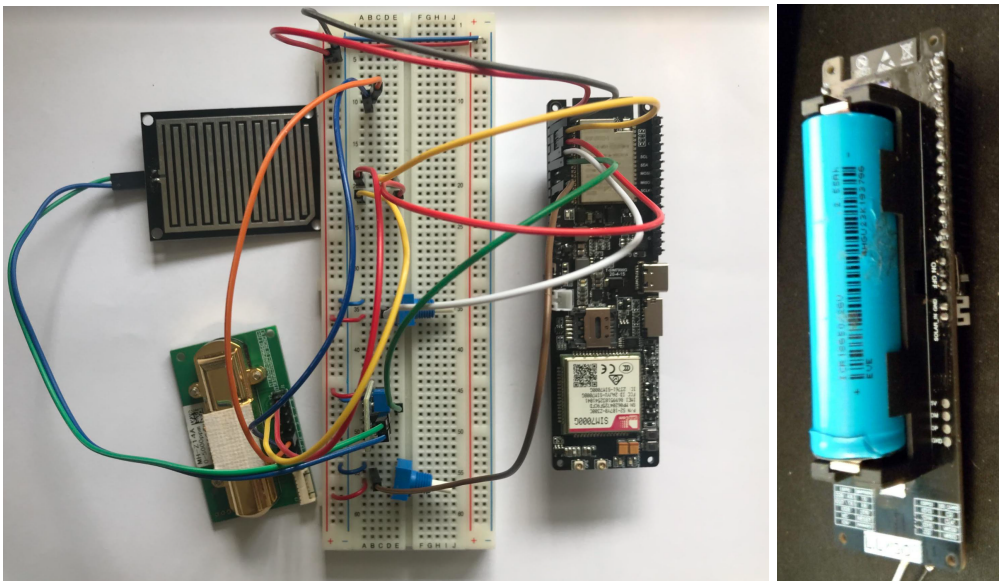


Figure Prototype of Device

Our microcontroller was connected to the internet over WiFi. We also planned to use cellular data, however, we ran into SIM card issues which meant we were unable to test the functionality of it. For our prototype we had our sensors sending data on a very tight loop so that we would not need to wait for a sampling period to update the web

application. In our final design the sampling loop would most likely be 10+ minutes of a modem sleep, allowing us to still passively monitor the environment and wake up to alert the server if it detects extreme situations.

For our web application our prototype used an API key for security to identify which device was sending it information and would store the information temporarily on a flat JSON page. In our final design instead of storing the data temporarily, we would put it into a Azure MySQL database. After storing the data, our web application would update the device on its map if required. For example, if the fire risk was low the device's dot would be green, and if it then sent high fire risk readings the dot would turn red.

7. Testing

7.1. Hardware Tests

- In order to determine that our device will work correctly, we needed to test each component of the IoT device separately as well as test how each component will work together. We began by testing each sensor individually to make sure we were getting the expected readings. The carbon sensor was tested by blowing deeply into the sensor, causing the CO2 levels to rise up. As expected, the output began to increase from normal ppm levels to the highest ppm concentration level. We then left it alone for a couple of minutes and watched as the ppm concentration went down. For the rain sensor, we tested it by submerging it in water. The sensor's sensitivity needed to be adjusted to not interpret a finger touch as water. Lastly, we tried testing the temperature and humidity sensor. We used a hair dryer on high heat to increase the temperature. In addition, we placed the sensor in a humid propagation space, but the sensor was not responsive to the change in environment. Therefore we had to discard that sensor.

7.2. Microcontroller Software Tests

- In order to send and receive data to the server we needed to ensure that the microcontroller had the ability to access WiFi or cellular connection. Testing the WiFi and cellular connectivity of the microcontroller consisted of creating an HTML web page that would go online when the microcontroller was turned on and users would be able to access it using their personal device. When the microcontroller is turned off the web page should also be offline.

7.3. Cloud Tests

- When creating the cloud server we started with creating a HTTPS specification to describe the commands that we would accept. We created a local webserver and

began testing them using Postman and curl to send HTTPS requests. Once we knew that the basic commands worked we hosted the server in Heroku. We then tried to access the web server from different computers to make sure it was publicly available. After we made sure the server was available we added API keys to increase the security and made sure that only requests with the right permissions made it through. We then secured the front end of the server by adding a login prompt and JWTs. At this point we started focusing on making our fire map more interactive and integrated Mapbox so that we could have an interactive map instead of just floating dots. After making sure that we could draw dots we used curl requests to update the node information and show that the dots changed color when the device location's environment changed. At this point we gave our microcontrollers API keys and started using them to test the full end-end functionality of our design. We also tested the basic functionality of our database such as registering new devices and adding/retrieving data locally, but we did not end up putting it up on Azure to truly test it.

A. Appendix

1. Problem Formulation

Brainstorming Output

Logan:

- (1) IoT Device to track use of bike paths to see which routes are most used
    - “need” : a way to find the safest, smoothest, most efficient bike paths

Sal:

- (2) Designing a device that monitors posture and movements and predicts high risk positions. (not IoT though so probably not the best choice since that’s the direction we want to go in)
    - “need” : a way to prevent workplace injuries and increase safety

Aracely:

- (3) Creating a tracking device to attach to veeries, tracking their migration patterns can help predict hurricane seasons.
    - “need” : better technology to help predict the intensity of hurricanes
- (4) An IoT device that can detect CO2 & temperatures in wildfire hotspots
    - “need” : a way to predict and reduce wildfires

Julia:

- (5) Create a water dispenser that monitors how much water is left and can send a notification when it is running low
    - “Need”: could be useful for portable water dispensers, which could be used within environments such as a homeless encampment

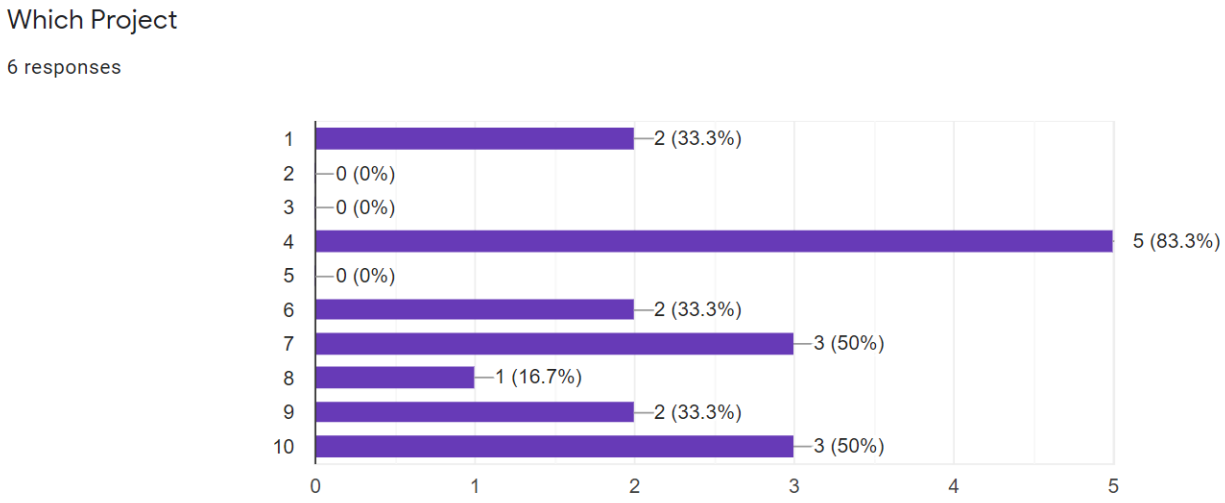
Jose:

- (6) Design a robot arm to sort recyclables with sensors(visual or tactile).
    - Need: A method for sorting recyclable materials that is more cost effective than human workers
- (7) Design a smart watering system for gardens that can water plants with precision. Use sensors to measure moisture in the soil to water accordingly (might be tricky depending on the plants being watered).
    - Need: A way to minimize the water spent on gardens

Quinn:

- (8) Create a budget friendly nature camera that hooks into a civilian science database to allow passive data collection in neighborhoods.

    - Need: A better way for data to be collected in neighborhoods and other hard to survey areas.
- (9) Build a device to help track animals' roaming patterns.
    - Need: We know the general area an animal roams, but want to understand how temperature/humidity/etc affects their behavior.
- (10) Design a morning routine assistant (bot or just IoT controlled devices) that will determine when you wake up and start tea/coffee, turn on a space heater, your room lights, making toast, etc. Anything to help you get out of bed in the morning.
    - Need: Getting out of bed in the morning can be hard.



Our group took a survey of the above ideas and decided that idea number 4 would be best.

Morphological chart

Wildfire Detection System Morphological Chart					
System Components	**Options**				
	1	2	3	4	5
Microcontroller	Arduino Nano 33 IoT **Cost:** $18.40	Raspberry Pi Model B 2GB **Cost:** $35.00	Raspberry Pi Zero W **Cost:** $10.00	ESP32 **Cost:** $15.00	
Temp/Humidity Sensor	TI HDC2021 Low-power humidity and temperature digital sensor **Effective measurement range:** $(-40°C) - (+1$ 0 - 100 RH% **Cost:** $2.67	DHT11 Temperature/ Humidity Sensor (pro- compatible w arduino) **Effective measurement range:** $(0°C) - (50°C)$ 20 - 80 RH% **Cost:** $5.00	Adafruit AHT20 Temperature & Humidity Sensor (pro- reaches a lil higher temp ranges ~185 F) **Effective measurement range:** $(-40°C) - (+8$ 0 - 100 RH% **Cost:** $4.50	DHT22 Temperature/Humidity Sensor **Effective measurement range:** $(-40°C) - (80°$ 0 - 100 RH% **Cost:** $9.95	Silicon Labs SI7013-A20-GM1 **Effective measurement range:** $(-10°C) - (85°$ 0 - 100 RH% **Cost:** $3.13
Wind Sensor	Anemometer SKU:SEN0170: **Effective measurement range:** 0-30m/s **Cost:** $48.00	Anemometer Adafruit 1733 **Effective measurement range:** 0.5-50m/s **Cost:** $44.95	Wind Sensor Rev. C **Effective measurement range:** 0 - 26.8224m/s **Cost:** $17.00	Wind/Rain sensor Argent Systems (Raspberry Pi compatible) Cost: $68.00	
Carbon Sensor	Senseair K30 10,000ppm CO2 sensor (pro - cheap) **Effective measurement range:** 0 - 10,000 ppm CO2 **Cost:** $85.00	Gravity: Analog CO2 gas sensor (pro- compatible w arduino) **Effective measurement range:** 350-10,000 ppm CO2 **Cost:** $56.00	Adafruit SCD-30 - NDIR CO2 Temperature and Humidity Sensor **Effective measurement range:** 400-10,000 ppm CO2 0 - 100 RH% $(-40°C) - (+1$	Carbon Dioxide CO2 Sensor (Arduino & Raspberry Pi Compatible) **Effective measurement range:** 0 - 10,000 ppm CO2 **Cost:** $49.97	NDIR CO2 Sensor MH-Z14A PQM NDIR Infrared Carbon Dioxide Sensor Module **Effective measurement range:** 0 - 5,000 ppm CO2 **Cost:**

			Cost: $58.95		$22.99
Rain Sensor	Arduino Rain Drop Sensor **Cost:** $9.95	Wind/Rain sensor Argent Systems (Raspberry Pi compatible) Cost: $68.00			
Mesh Network Implementation	MTM-CM5000-MSPWireless Sensor Node IEEE 802.15.4 compliant 2.4GHz Wireless Module **Range:** ~120m(outdoor)	SLTB004A Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT Kit Wireless SoC w/ multi-protocol radio 2.4GHz ceramic chip antenna **Cost:** $19.99	Digi XBee ZigBee Mesh Kit Include three XBee Grove development boards, three XBee ZigBee modules **Cost:** $89.00 (Could buy individ-ual modules too)	MICA2 Basic Kit (MOTE-KIT4x0)	
Solar Energy & Battery	USB / DC / Solar Lithium Ion/Polymer charger - v2 **Output Voltage:** 3.7/4.2V **Cost:** $17.50	Solar Panel/ Voltage Regulator/ Lithium Ion Battery/ 6V DC, 500mA Solar Panel **Output Voltage:** 3.7V **Cost:** $33.32	2 Watt Solar Charger Kit **Output Voltage:** 5V **Cost:** $59.00	Power ESP32 w/ Solar Panels (includes battery level monitoring) **Output Voltage:** 4.2V **Cost:** $26.55	
SIM Cards	Telnyx One-time charge: $1 Monthly fee: $2.00/month Data rate: $0.01/MB	Arduino Sim 90-day 10MB free trial, followed by 5MB at $1.50 per month Limitation: only			

		connects to Arduino IoT Cloud			
Cloud Platform	AWS	Google **Cloud Platform** **IoT Core**	Microsoft Azure **IoT Hub**	Arduino IoT Cloud	
Notification API	Twilio **SMS:** $0.0075/segment https://www.twilio.com/sms/pricing/us **Phone:** $0.0130/min https://www.twilio.com/voice/pricing/us **Email:** Free with 100 emails/day limit https://sendgrid.com/pricing/	Vonage **SMS:** $0.0068/message https://www.vonage.com/communications-apis/sms/pricing/ **Phone:** $0.0139/min **Email:** $0.000843/email	Telnyx **SMS:** $0.0025/segment **Phone:** $0.0070/min https://telnyx.com/pricing/call-control		

Morphological Chart: This chart outlines all the design options considered for both hardware and software components of our device. On the hardware side we looked at various options for a microcontroller as well as several different sensors of interest to our design, such as: humidity/temperature sensor, wind sensor, carbon sensor and rain sensor. We also looked at a couple of different SIM cards to give our microcontroller network service to connect to our web interface. A couple of additional hardware things we looked at were options for implementing solar energy into our device, and options for implementing a mesh network into our device. On the software side of things we considered our options for the cloud platform and notification API.

Decision Table

Microcontroller Decision Table					
Microcontroller	Raspberry Pi 4 Model B 2GB	Arduino Nano 33 IoT	Raspberry Pi Zero W	ESP32 (dual-core or single-core) NodeMCU	ESP8266 NodeMCU
Price	$35	$18.40	$10.00	~$10	~$5
Compute Power	2GB Ram Quad Core ARM x64, 1.5 GHZ	256KB CPU Flash memory 32KB SRAM	512MB Ram Broadcom 1GHZ processor	520 KiB SRAM 1 or 2 core Xtensa LX6, 240Mhz	32-bit RISC CPU Xtensa LX106 - 80Mhz
Networking	2.4 GHz, 5.0 GHz WiFi Bluetooth 5.0 BLE Gigabit Ethernet	2.4GHz WiFi Bluetooth 4.2 BLE	2.4GHz WiFi Bluetooth 4.1 BLE	2.4GHz WiFi Bluetooth 4.2 BLE	2.4GHz WiFi
Power Consumption	>= 5V * 500mA = 2.5W Typical *No Deep Sleep*	>= 3.3V * ~20mA = ~0.1W Typical *Supports Deep Sleep*	>= 5V * ~200mA = 1W Typical *No Deep Sleep*	>= 3.3V * ~50mA = ~0.2W Typical *Supports Deep Sleep*	>= 3.3V * ~30mA = ~0.1W Typical *Supports Deep Sleep*
Operating Temperatures	0-50 C, 32-122 F		0-50 C, 32-122 F	–40-125°C, -40-257 F	–40-125°C, -40-257 F
Sensor / IO compatibility	40 pin GPIO header Up to 6x UART Up to 6x I2C Up to 5x SPI 1x SDIO interface 1x DPI 1x PCM Up to 2x PWM channels Up to 3x GPCLK outputs	14 pins GPIO Up to 1x UART Up to 1x I2C Up to 1x SPI 8 Analog Inputs 1 Analog Output	40 pin GPIO Header Up to 6x UART Up to 6x I2C Up to 5x SPI Up to 11x PWM	34 pins GPIO 18 ADC Channels Up to 3x SPI Up to 3x UART Up to 2x I2C Up to 16 PWM Channels Up to 2 DAC Up to 2x I2S Up to 10x Capacitive Sensing IO	16 GPIO Pins 1x ADC 1x UART 1x SPI 1x i2C

Microcontroller Decision Table: Various microcontrollers and their components, like price, power, ect., were considered when making our final decision. In order to successfully achieve our desired project, we need a microcontroller that can withstand

outside temperatures as well as be able to connect with our sensors. In the end, we chose two options that met our requirements, the ESP32 SIM800L and the T-SIM7000G ESP32. We chose the former due to its affordability, although it only works on 2G networks. The later network microcontroller is able to work on 4G networks, which makes it more expensive, but both are compatible with our project.

Design Implementation Cost Decision Table			
Option	1	2	3
Microcontroller	ESP32 **Cost:** $15.00	Arduino Nano 33 IoT **Cost:** $18.40	Raspberry Pi Model B 2GB **Cost:** $35.00
Temp/Humidity Sensor	TI HDC2021 Low-power humidity and temperature digital sensor **Effective measurement range:** $(-\ 40°C) - (+\ 125°C)$ 0 - 100 RH% **Cost:** $2.67	Adafruit AHT20 Temperature & Humidity Sensor (pro- reaches a lil higher temp ranges ~185 F) **Effective measurement range:** $(-\ 40°C) - (+\ 85°C)$ 0 - 100 RH% **Cost:** $4.50	DHT22 Temperature/Humidity Sensor **Effective measurement range:** $(-\ 40°C) - (80°C)$ 0 - 100 RH% **Cost:** $9.95
~~***Wind Sensor***~~	~~Wind Sensor Rev. C~~ ~~**Effective measurement range:**~~ ~~0 - 26.8224m/s~~ ~~**Cost:**~~ ~~$17.00~~	~~Anemometer Adafruit 1733~~ ~~**Effective measurement range:**~~ ~~0.5-50m/s~~ ~~**Cost:**~~ ~~$44.95~~	~~Wind/Rain sensor Argent Systems (Raspberry Pi compatible)~~ ~~Cost:~~ ~~$68.00~~
Rain Sensor	Arduino Rain Drop Sensor **Cost:** $9.95	Arduino Rain Drop Sensor **Cost:** $9.95	Rain Sensor implemented from Wind Sensor above
Carbon Sensor	NDIR CO2 Sensor MH-Z14A PQM NDIR Infrared Carbon Dioxide Sensor Module **Effective measurement range:** 0 - 5,000 ppm CO2 **Cost:** $22.99	Carbon Dioxide CO2 Sensor (Arduino & Raspberry Pi Compatible) **Effective measurement range:** 0 - 10,000 ppm CO2 **Cost:** $49.97	Senseair K30 10,000ppm CO2 sensor (pro - cheap) **Effective measurement range:** 0 - 10,000 ppm CO2 **Cost:** $85.00

~~Mesh Network Implementation~~	~~SLTB004A Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT Kit~~ ~~Wireless SoC w/ multi-protocol radio~~ ~~2.4GHz ceramic chip antenna~~ ~~**Cost:**~~ ~~$19.99/unit ($60.00/3units)~~	~~SLTB004A Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT Kit~~ ~~Wireless SoC w/ multi-protocol radio~~ ~~2.4GHz ceramic chip antenna~~ ~~**Cost:**~~ ~~$19.99/unit ($60.00/3units)~~	~~Digi XBee ZigBee Mesh Kit~~ ~~Include three XBee Grove development boards, three XBee ZigBee modules, three Micro-USB cables~~ ~~**Cost:**~~ ~~$89.00~~ ~~(Could buy individual modules too)~~
4g/LTE Modem			
Solar Power & Battery	Solar Panel/ Voltage Regulator/ Lithium Ion Battery/ 6V DC, 500mA Solar Panel (3.5W) **Output Voltage:** 3.7V **Cost:** $33.32	Solar Panel/ Voltage Regulator/ Lithium Ion Battery/ 6V DC, 500mA Solar Panel (3.5W) **Output Voltage:** 3.7V **Cost:** $33.32	2 Watt Solar Charger Kit **Output Voltage:** 5V **Cost:** $59.00
$ Total	~~**$160.93**~~ **$100.93**	~~**$221.09**~~ **$161.09**	~~**$345.95**~~ **$256.95**
Score	((125/3 + 27/3 + 5000/3)/2 + 3.5/2) / 1.6093 = 534.65	((85/3 + 50/3 + 10000/3)/2 + 3.5/2) / 2.2109 = 764.81	((85/3 + 25/3 + 10000/3)/2 + 2/2) / 3.4595 = 487.35

	Weight	Design 1	Design 2	Design 3
Cost	0.4	7	4	0
Sensor Power	0.3	6	10	9
Solar Power	0.3	7	7	4
Score Total		**6.7**	**6.7**	**3**

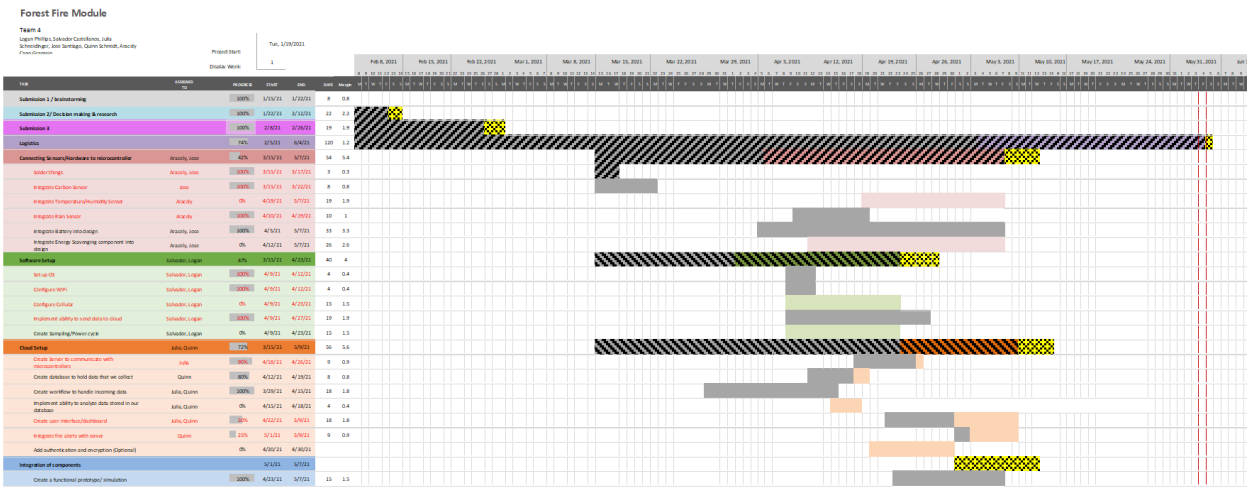
Design Implementation Cost Decision Table: In this design table, we choose three different design implementations that contain one type of microcontroller, one of each sensor, and a battery option. When we first began our design, we planned to have each IoT device connect with each other through a mesh network. We also planned to have four different types of sensors that aid in detecting a fire through environmental monitoring. Lastly, we decided to have energy sourcing as an option to power our device.

After thoughtful consideration, we decided to eliminate the mesh network component for now. If given the opportunity of extra time, we will try to implement a connected network into our project. We also decided not to have a wind sensor, seeing that we had sufficient sensors to detect a fire.

The score of each design was determined based on three main categories, cost, sensor power, and solar. For the cost, we agreed that the IoT device should be as inexpensive as possible, allowing us to build numerous devices. The most expensive design option was number 3, costing roughly $250, which received zero points for cost. In order to factor in the individual cost of each component, we decided to give one point to every item that was less than $25. For the sensor power, we ranked the sensor with the most power a 5, and the sensor with the least power a 3. The highest-ranking was given to the temperature/humidity sensor with the highest range of temperature and the carbon sensor with the most granularity. Lastly, to determine the score of each solar battery, we multiplied the watts by 2 to get a score out of 10 and assigned that number to their respective scores. Adding up the scores, we got the final score totals. There was a tie between design 1 and 2, but ultimately we decided design 1 was the best option for our team.

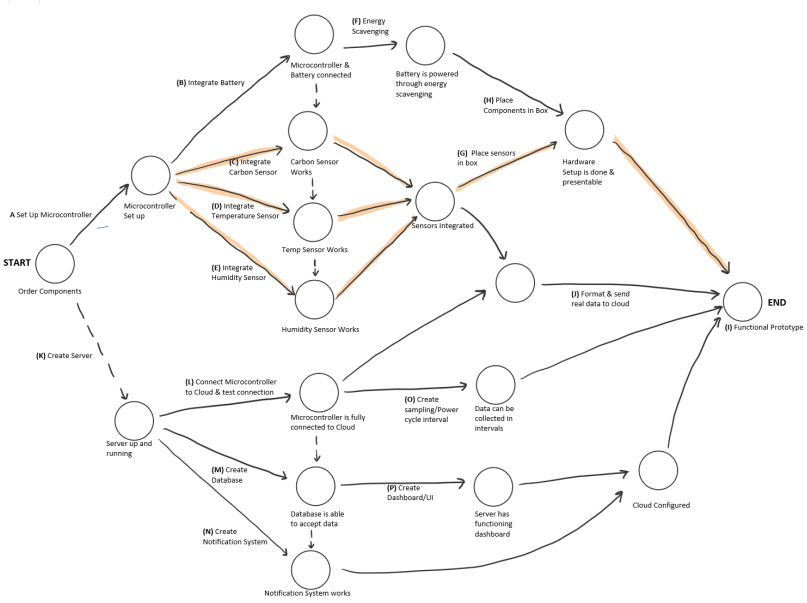
2. Planning

2.1 Gantt Chart



Our Gantt chart was divided into three subsections, where each subsection was split amongst two team members. These subsections included hardware, software, and cloud. Each team member was responsible for updating their task accordingly. The tasks with names in red were determined to be ones that were most important for our functional prototype, so those tasks were prioritized.

2.2 CPM & PERT Analysis



Activity	Duration (Days)	Preceded By
A	2	
B	8	A
C	8	A
D	8	A
E	8	A
F	12	B
G	1	C,D,E
H	1	F,G
I	1	H,J,O,Q
J	8	G,L
K	8	
L	2	K
M	8	K
N	4	K
O	8	L
P	12	M

Critical Path = A -> C, D, E -> G -> End

The critical path was calculated to be following the hardware completion of the project. After listing each task, the team estimated the length of days based on previous experience or estimation.

2.3 Division of Labor and Collaboration

When creating the tasks in the Gantt chart, we began by grouping them into the sections of our design that they related to. These sections were the sensor device software, device hardware, and cloud implementation. We were given the opportunity to decide which tasks interested us the most, splitting ourselves up so that two people worked on each section. The cloud implementation was done by Quinn and Julia, the hardware portion was done by Aracely and Jose, while Logan and Salvador worked on the device's software. Within the individual sections, the group had the option to split up the work or collaborate on each task. For the hardware and cloud implementation, the team members worked collaboratively on each assignment. The software implementation was split between the two members. Communication between each section was crucial to get the whole product working. The team's main mode of communication was through discord and zoom, where we would discuss the team's progress weekly.

3. Test Plan & Results

3.1 Hardware Tests

- CO2 Sensor
    - Purpose: To ensure that the right CO2 ppm levels were being read
    - Method: Attempt to increase the levels of ppm by blowing directly into the sensor and decrease the ppm level by placing the sensor in an outside environment for some controlled time
    - Expected Result: The ppm levels will increase an obvious amount when blown into, but decrease when the sensor is taken outside
    - Actual Result: The ppm level increased to the highest level, 5000ppm when it was blown into. But the sensor would take almost five minutes to return to normal ppm levels, even if it was placed outside
- Temperature and Humidity Sensor
    - Purpose: To make sure the correct temperature and humidity is being outputted from the sensor
    - Method: Increase both the temperature and humidity by placing a damp paper towel near the sensor and using a hair dryer in high heat setting to blow air to the sensor
    - Expected Results: The output of the sensor was supposed to change with respect to the hot air being blown into it. When the hot air is directly pointed towards the sensor, the temperature and humidity should increase. If the air from the hair dryer was set to a lower setting, the temperature should decrease but the humidity should increase. When there are no factors placed on the sensor, the sensor should read room temperature and humidity.
    - Actual Results: When the sensor was placed under different conditions, there was no change in the output. The output remained the same constant numbers throughout the entire tests.
- Rain Sensor
    - Purpose: To make sure the sensor is being triggered properly when in contact with water
    - Method: Place the sensor in water, or lay a wet paper towel above the sensor
    - Expected Results: The sensor would properly detect when it is in contact with water. The sensor should also be able to detect when only a part of it is in water.
    - Actual Results: The sensor would display it was in contact with water when it was only touched with a finger. Therefore the sensitivity needed to be changed in the sensor so it will not detect anything other than water.

3.2 Microcontroller Software Tests

- WiFi Connectivity
    - Purpose: To send and receive data and commands and communicate alongside the cloud server using a wireless connection.
    - Method: Create a simple HTML page that goes online when the microcontroller is on.
    - Expected Result: Server will go online once the microcontroller is on showing that it has the ability to connect to WiFi and user will be able to access the web page.
    - Actual Result: Server successfully went online and users were able to access the website. Plugging in the lithium battery did cause the microcontroller to remain on constantly which was unexpected.
- Cellular Connectivity
    - Purpose: To send and receive data and commands and communicate alongside the cloud server using a cellular connection.
    - Method: Ping various web-servers such as Google, Print to serial parameters of cell signal strength
    - Expected Result: Receive response from web-server and have results of signal strength printed to console, verifying that device can connect to 4G LTE and connect to the internet
    - Actual Result: Incompatibility with SIM card prevented connection to internet, serial console output verified connectivity to 4G LTE

3.3 Cloud Tests

- Server API Security Test. (Run on both Database and web application servers)
    - Purpose: To make sure that our web server is protected by an API key.
    - Method: Attempt to make a request with an API key to ensure the server is currently running. If the server is operational, attempt to make the same request without an API key in the HTTPS header.
    - Expected Result: The first request should succeed and return 200, the second should fail and return 403.
    - Actual Result: The first request returns a 200 status code, the second returns 403.
- Data/Location Request Test
    - Purpose: Determine if requests inputting location or sensor data can be correctly handled by the server

    - Method: Using an API key, submit POST requests to /nodes/location and /nodes/data with sample values in the request body.
    - Expected Result: The server's database should contain the new data
    - Actual Result: We tested this with a JSON file instead of a SQL database. The file acquired the new values as expected.
- Fire Detection Logic Test
    - Purpose: Test the accuracy of the system at identifying an imminent fire from sensor data
    - Method:
        i. Use a dataset of sensor readings collected over time in a wilderness environment where some fires have taken place nearby. Each row of sensor readings should be labeled as either occurring or not occurring during a fire.
        ii. Set the server's database to data that occurred in a certain time window within the dataset. Then run the detection method on this data and store the result.
        iii. Repeat ii. for all possible time windows
    - Expected Result: The true-positive and true-negative rates of the fire classifications should be above 90%.
    - Actual Result: This test cannot currently be run since we are unable to find a suitable dataset, and creating data ourselves would be infeasible as it would require burning actual trees.
- End-to-End Test
    - Purpose: Test that the characteristics of a sensor device's environment can be forwarded to the server, and an alert will be triggered only when the environment enters a state associated with a fire. This test also determines if the location and fire risk for each device can be correctly represented by the web UI.
    - Method:
        i. Using a potentiometer to simulate sensors that are not working, create a low-risk environment, which is made up of low heat, high humidity and low CO2.
        ii. Gradually raise the temperature, lower the humidity and increase the CO2 as far as possible. Repeat this test to check that the sensor values can change in any order or at the same time, and correct functionality will still follow.
        iii. Revert the environment to a low-risk one.
    - Expected Result:
        i. The dot related to the sensor node should be green, and the server should not have sent any alerts.

        ii. The dot should turn yellow when the temperature is high and the humidity is low, but the CO2 is not high enough. The dot should turn red and an alert should be sent only once all three values reach dangerous levels. The dot should be located correctly on the map.
    - Actual Result: We tested the system as described except that we only changed the sensor values in one order and did not completely revert them to their previous values. We have not integrated alerts yet into our full system, but we were able to see the dot's color change correctly.
    - Discussion: A real end-to-end test would involve setting up the device in a wildland area and starting a fire to see if an alert would be detected. This obviously is not feasible for safety reasons. Instead, we had to create our own definition of an environment that would likely harbor a fire based on the temperature, humidity and CO2 changes we could safely cause. In order to differentiate a "high" value from a "low" one in the test we conducted, we set the thresholds for temperature, humidity and CO2 to be 50, 50 and 1500, respectively.
- Server Blacklist Test
    - Purpose: If a device starts misbehaving we want the ability to block its API key so that we don't allow a bad actor into our system.
    - Method: Start sending repeated bad requests every minute from an API key and wait to see how long it takes for the server to stop responding.
    - Expected Result: After a few minutes we should start getting 403 Forbidden responses from the server.
    - Actual Result: Untested.
- Database Register Device Test
    - Purpose: Make sure that when a new device is registered we create a new table for it.
    - Method: List the tables available in our database. Check to make sure our new table does not already exist. Register a new device and check the table list again to make sure that it was created.
    - Expected result: The new device table does not exist before running the test, but it does exist after.
    - Actual results: After testing the system we found that the database table is created as expected. However, we did not fully integrate it with the main server the devices will interact with, so it is only partially complete.
- Database Insert Data Test
    - Purpose: Make sure that when data comes into the database it generated the timestamp and entry id properly.

    - Method: Check a table's data and see the most recent entry. Add a new data point to the table and make sure that each of the auto-generated fields are correct.
    - Expected Result: The entry ID of the newest data point should be one higher than the previous entry, and the sample date should be the time you ran the entry function.
    - Actual Result: When we ran the test the entry id and date/time were properly generated. This is only tested locally and not integrated into the main server so it is only partially tested.
- Database Get Most Recent Data Test
    - Purpose: Make sure that when we store data it is still retrievable.
    - Method: Attempt to retrieve data from an existing table. Add a new data point to the table and then run the retrieve again.
    - Expected Result: We should get old data on the first get and new data on the second.
    - Actual Result: When we ran the test we got the expected result. However, this was only tested locally and not integrated into the main server so it is only partially tested.

4. Review

- Aracely Cano-Gramajo - Overall I really enjoyed working on this project and working with the team members. One thing that I would do differently given the opportunity, would be to thoroughly research the sensors before purchasing them. Had this been done before buying all the hardware components, we would have bought the correct sensor for temperature and humidity, and our prototype would have been better. I also would have liked to incorporate more sensors. Aside from that, I think the team did well given the constraint of working remotely.
- Jose Santiago - I really enjoyed seeing a project through the design process, from the drawing board to the prototype. Working in a team can be difficult, but everyone in the team was reliable and did their work diligently. Teamwork was a little more difficult in the remote setting but I appreciate the work everyone put into completing this project. Something I think we could have done better was the planning. The planning was pretty solid, but there were a few major oversights we had. I think we would have benefited a lot from learning about the causes of wildfires as well as the causes of fires in general. Studying and understanding the source of fires could help us design a device with sensors that would be best for detecting fire risk. Another thing we overlooked was sensor compatibility with our microcontroller. If we read a little into the device datasheets before buying

our parts we may have been able to get a temperature and humidity sensor that worked for our design. Overall this was an insightful experience into working in a team to design a device.

- Quinn Schmidt - Working with the team was a really good experience for me. Everyone cared about the project and did their work well. I think we could have benefited from meeting more than once per week especially during the brunt of the research/planning of last quarter and when working towards a functional prototype this quarter.
- Logan Phillips - This project was a great learning experience for me and allowed me to really evaluate how work gets done in a group setting. Having this project take place during COVID was a challenge as all work had to be done remotely, however, there was much learned in the process. If I could do the project again, I would like it to be in person. This would allow for a deeper level of collaboration and understanding between team members. Another change I would make is in the overall design of the device. I would like to take the design further and have a fully fleshed out enclosure and circuit design, allowing for a near-deployment state.
- Julia Schneidinger - It was great working with everyone on the team, and I am glad that I got experience working on a large project in a group setting. If I could do the project differently, I would have done more research on fire event classification methods earlier on in the quarter, so the fire risk logic would be better suited for real-world use. It also might have helped to get more information about how to best fit this system to the needs of fire protection agencies.
- Salvador Castellanos - This is the first engineering project where I had the opportunity to work with a team and I enjoyed the whole experience. The only aspect that I did not enjoy very much was working remotely as it made communication slightly more difficult but this is something that happens in a real life environment as well and it taught us how to adapt to these circumstances. Something I wish we had time to do would have been deploying the devices and seeing how well the device worked out in the forest with a fully working enclosure as well to keep the device safe. Overall, throughout the two quarters this project has allowed me to learn many different concepts and working with the team has been a great experience.