Launch Pilot: Smart Vent

ECE 413 Senior Project Development II

Spring 2024

Industry Sponsor: **Cedrec Sumimoto**
Faculty Advisor: **Andrew Greenberg**
Team members: **Meshal Almutairi, Michael Fontaine, Abdulaziz Alateeqi, Fawzan Alfahad, John Michael Mertz**

Table of Contents

Executive summary **3**
- Problem Overview 3
- Project Deliverables 3
- Our Approach 3
- Results, Successes, and Failure 3
- Some challenges we faced 4

Background **4**
- Industry and Scientific Context 4
- Research 4
- System Block Diagram 6

Requirements **7**
Objectives and Deliverables **7**
- Objectives 7
- Deliverables 7
- Changes in Objectives and Deliverables 8

Approach **8**
Design **9**
Testing **25**
Results **25**
Post Mortem **26**
Project Resources **28**
Conclusion **29**
Appendix I: Smart Vent User Manual - Launch Pilot **30**
Appendix II: Getting started with Software Development **38**
Appendix III: Test Plans **40**

Executive summary

Problem Overview

We teamed up with Launch Pilot to make a smart vent system. The main goal was to ensure safe indoor air quality by monitoring CO_2 levels. When the CO_2 level goes above the safe threshold, the system automatically activates to circulate fresh clean air into the room, thus reducing the CO_2 level.

Project Deliverables

- A CO_2 level detection system that activates fans automatically.
- A user-friendly dashboard that is easily accessible remotely
- A display that shows temperature, humidity, and CO_2 ppm.
- Physical buttons to navigate the menu and change certain options

Our Approach

We designed the system using sensors to detect CO_2 levels, temperature, and humidity. The design also includes fans that operate at different speeds to help with fresh air circulations, an LCD to display the sensor readings, and a microcontroller as the brain of this project. We also implemented a web-based dashboard that is accessible via a local network for remote monitoring and adjustments such as changing the CO_2 level threshold, turning the system entirely on or off, and controlling the fan's speed. Additionally, we integrated buttons on the fans to provide direct control for the different settings.

Results, Successes, and Failure

Our system successfully reads carbon dioxide levels and activates the fans accordingly. Moreover, the web-based dashboard and the buttons work great when adjusting different settings. The LCD provides real-time readings from the sensors as

well as any warnings. Users can control the system directly or remotely, which has been a major plus.

Some challenges we faced

- Sensor calibration: The sensors need some time to calibrate which slows the start-up time.
- Switching fans off: we couldn't turn the fans completely off via PWM, we needed to add some sort of a switch to shut them off. We decided to use a MOSFET and send a signal from a GPIO pin to its gate to turn on or off the fans.

Background

Industry and Scientific Context

Indoor air quality is important for health and well-being, especially in places like homes, offices, and schools where people spend a lot of time. High levels of CO_2 can lead to decreased cognitive function, fatigue, and health risks. The commercial demand for smart vent systems is growing, as these systems can improve air quality while also being energy efficient.

Research

Upon researching various open-source projects that could help with our capstone project, we have found the following:

1. **U8g2** - https://github.com/olikraus/u8g2/

This library is licensed under BSD. The library will significantly simplify the user interface design process for our display, enhancing the overall design efficiency.

2. **ESP-DASH** - https://github.com/ayushsharma82/ESP-DASH/

This library is licensed under General Public License v3. The library will help us develop a real-time dashboard for our product. It will allow us to communicate the

sensor measurements and other critical information to the user through wireless connectivity and allow the user to remotely manage the device.

3. **MQUnifiedsensor** - https://github.com/miguel5612/MQSensorsLib

This library is licensed under MIT. The library offers an efficient API abstraction for interacting with the CO_2 sensor. It includes calibration approaches that could help ensure accurate measurement and monitoring.

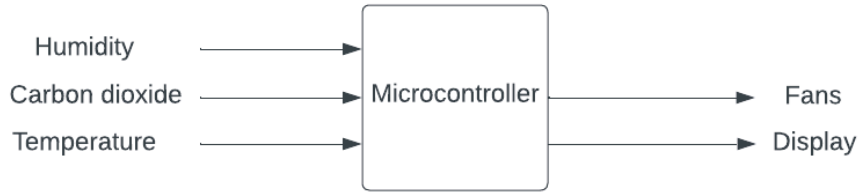
4. **Smart Vents** -

Halfacree, G. (2024, January 25). Tony Brobston's 3D-printed ESP8266 smart vents. https://www.hackster.io/news/tony-brobston-s-3d-printed-esp8266-smart-vents-put-every-vent-in-your-home-under-mqtt-control-b7b85e412e6f

This vent that Tony Brobston created is loosely similar to the Smart Vent project. He is using a duct-installed pressure sensor to open and close the vent damper to automatically control airflow in the room. He is looking to modulate the existing provided room airflow for comfort purposes. Our project is trying to add CFM of air flow movement into the room to flush out elevated levels of dangerous CO_2 gas. Though our end goals are different, the two products do share similarities. Both use external sensors to detect what we are trying to control and then we use an ESP microcontroller to initiate a device to adjust. The vent in this article was more relevant to us in the beginning when we were trying to flush out what this project was going to be. We have since discarded the pressure sensor idea.

System Block Diagram

Level 0:



Level 1:

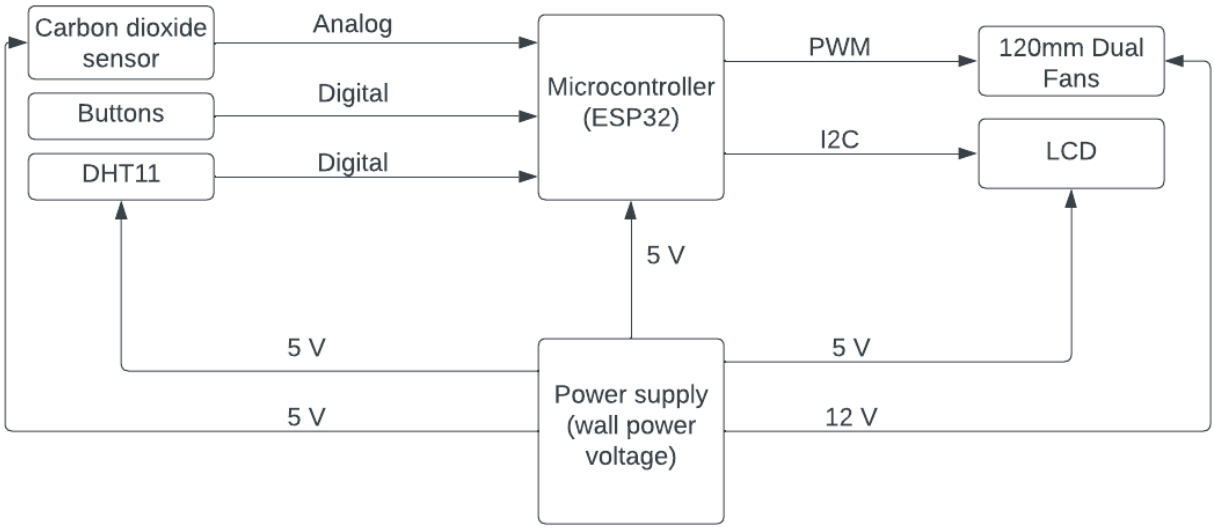


Figure 1: level 0 and 1 block diagrams

Requirements

The device **must**:

- Use sensors to detect CO_2, temperature, and humidity levels for accurate air quality analysis.
- Have a simple Display Screen.
- Use buttons to navigate through the system.
- Use Dual-sized Fans for airflow.
- Must be safe for children and pets with access to the vents.

The device **should**:

- Use Wi-Fi connectivity to offer Smart Home functionality.
- Alert the user using the display (Possibly LED Lights).

The device **may**:

- Use additional Gas Sensors that will trigger an alert for the users.
- Use Bluetooth connectivity to communicate with peripheral devices

Objectives and Deliverables

Objectives

The main objective of our project is to create a smart venting system that monitors and improves indoor air quality by detecting CO_2 levels, temperature, and humidity. Our goal was to make this system user-friendly and safe for children and pets.

Deliverables

- A functional prototype of the smart vent system with CO_2, temperature, and humidity sensors.
- A simple display screen that shows real-time readings of air quality.
- A system controls using physical buttons
- An optional web-based dashboard to monitor and control the system

Changes in Objectives and Deliverables

Initially, the system was designed to only monitor the CO_2 concentration, but then we added a third sensor to measure the temperature and humidity. The idea of a web-based monitor and control was added later on.

Approach

The phases we went through are the following,

Research: We started by doing lots of research about different sensors, the differences between eCO_2 and CO_2 and why they are different, different display screens, and potential libraries that we might use in the project. We also looked at the safe levels of CO_2 in a room, and the standard vent sizes in the United States.

Picking the components: After we did a lot of research, we picked many of different components that could be used in our system and ordered them. We also ordered a product from a competitor to compare and have an idea of how such systems are built.

Testing components: We then moved to testing the different components individually. This phase was the most crucial since we had to replace some components that were not compatible with our microcontroller or were not a good choice.

Design: After that, we moved to the design stage where we began with designing the system layout and our prototype after ensuring that each component was working properly and compatible with the other components.

Integration: In this phase, we faced the most problems. In the beginning, when we integrated everything we did not account that there are some pins in the microcontroller that we can not use. After working around this issue we had to redesign and modify our initial design several times. In the end, we managed to overcome this issue and have the best layout.

User feedback and iteration: After the integration phase we had to demo our integrated layout as a prototype and got lots of helpful feedback. We had to replace some components to reduce the price and improve the safety. We had to use a MOSFET instead of a relay to avoid any sudden voltage surges as well as price reductions. This ultimately led us back to the design and integration phase all again.

History and Surprises: Sensor Sensitivity and Calibration: Early in the testing phase we encountered several issues when it came to calibration and reading spikes with the CO_2 sensor, but we managed to do some research and consult our industry sponsor to overcome this.

Design

We can divide our entire system into two main subsystems, the first one being the basic power subsystem. We decided to not merge it with our main system since we needed to run 12 volts to the fans and 5 volts to run the ESP32 (our microcontroller). This system is pretty straightforward; the wall plug provides 12 volts that can go straight to the fans as well as into the 7805 voltage regulator to convert it to 5 volts to go into the ESP32 which can be seen from Figures 2, 3, and 4. We included two capacitors to smooth the power supply output.

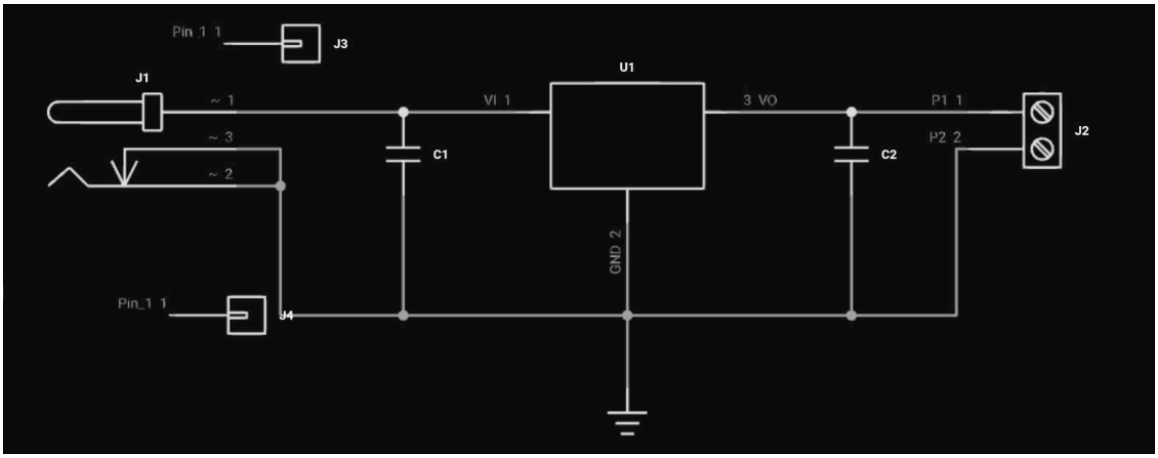


Figure 2: Power circuit schematic

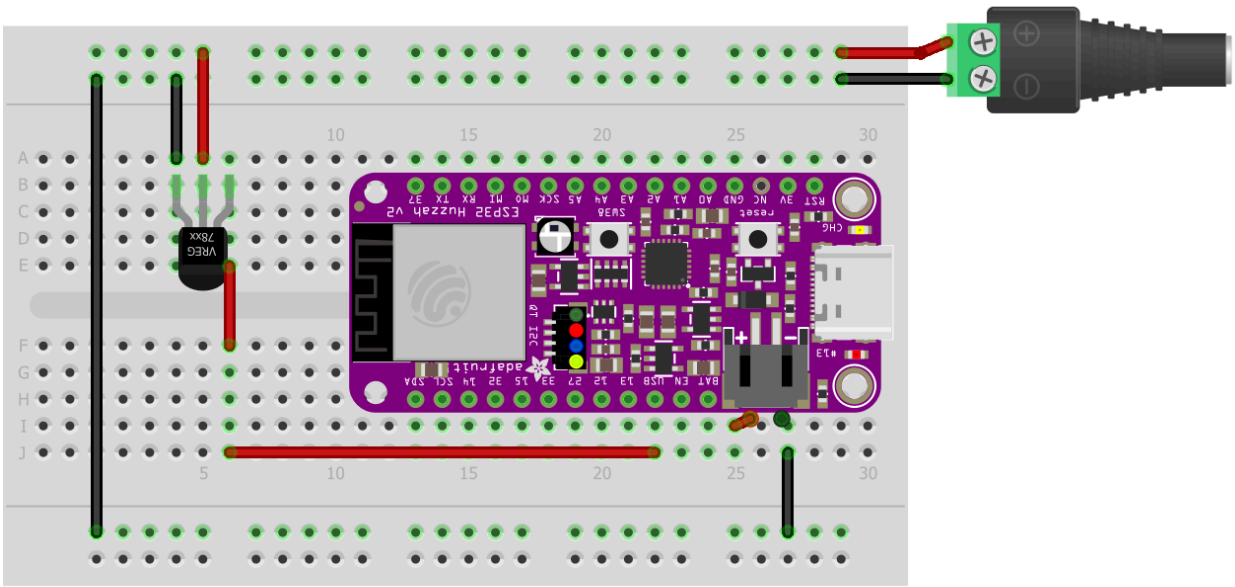


Figure 3: Power circuit layout

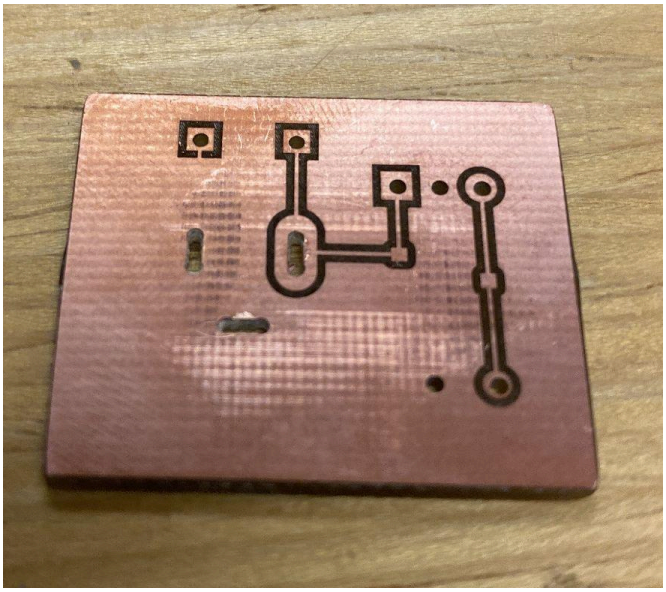


Figure 4: Power circuit printed

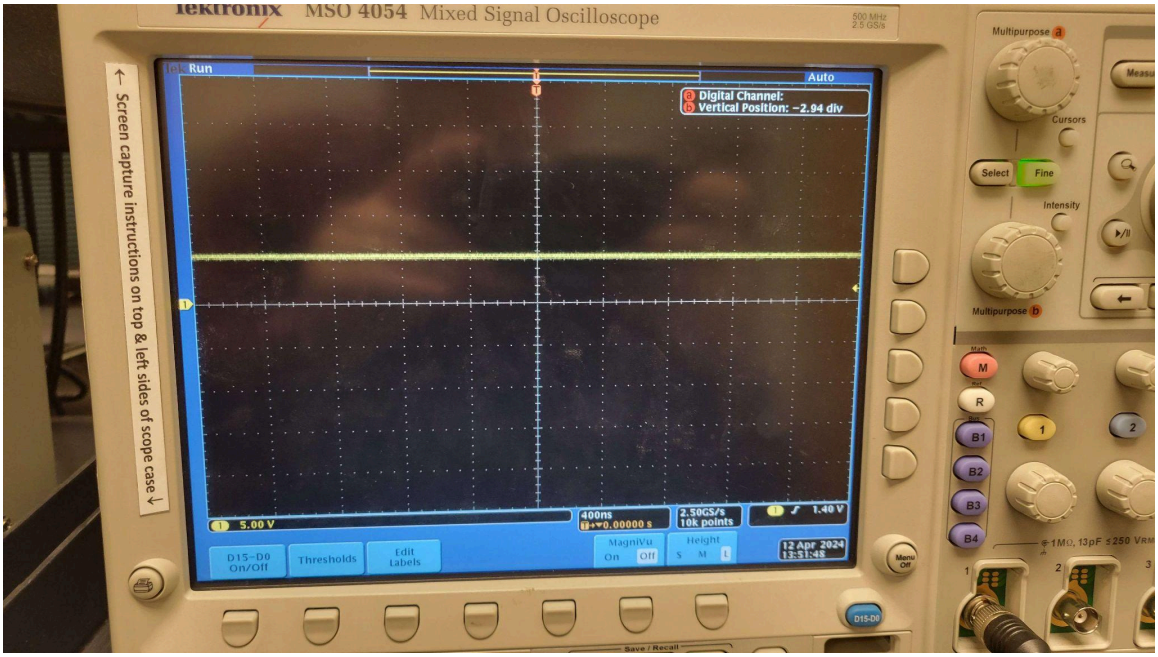


Figure 5: output of the voltage regulator

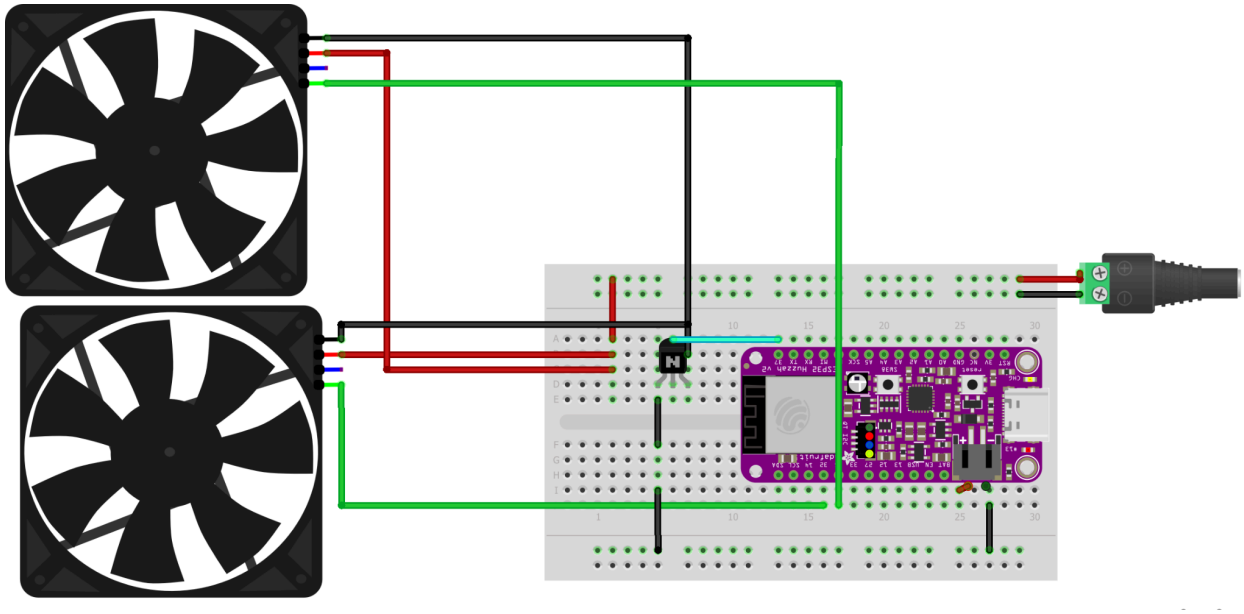


Figure 6: fans layout

Use this code to test the fans.

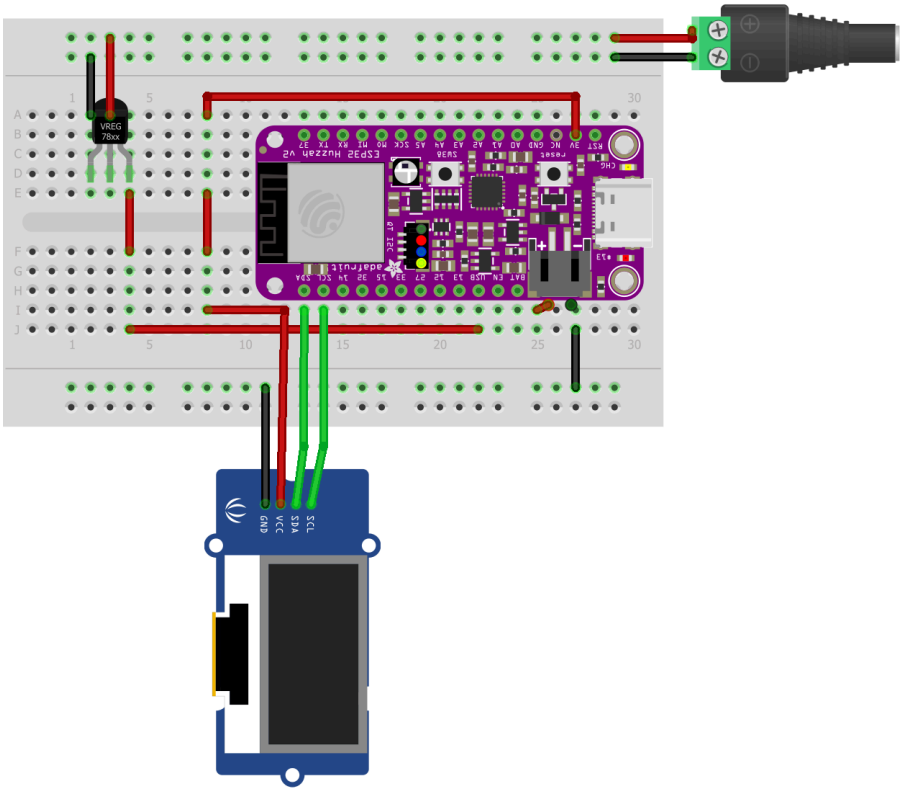


Figure 7: OLED layout

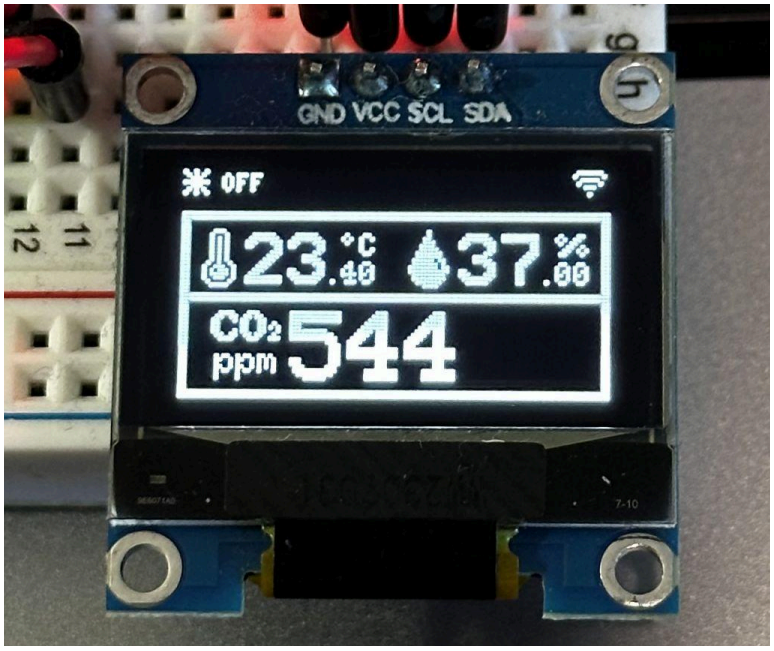


Figure 8: OLED displaying real-time readings

Display [code](#).

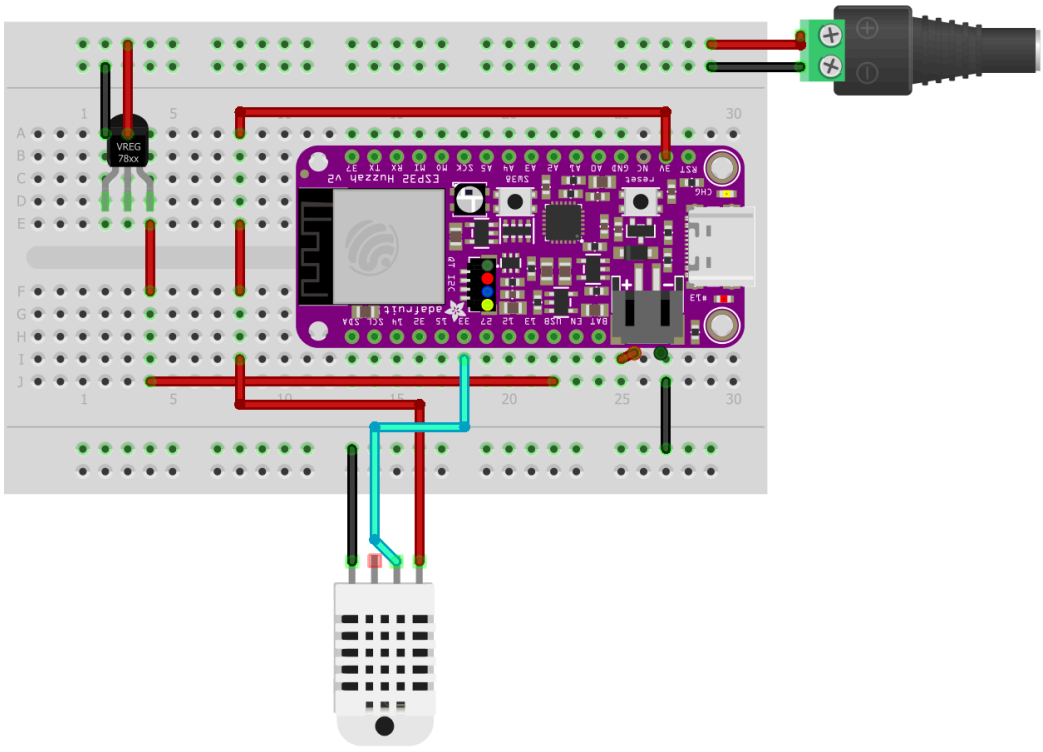


Figure 9: temperature and humidity sensor layout

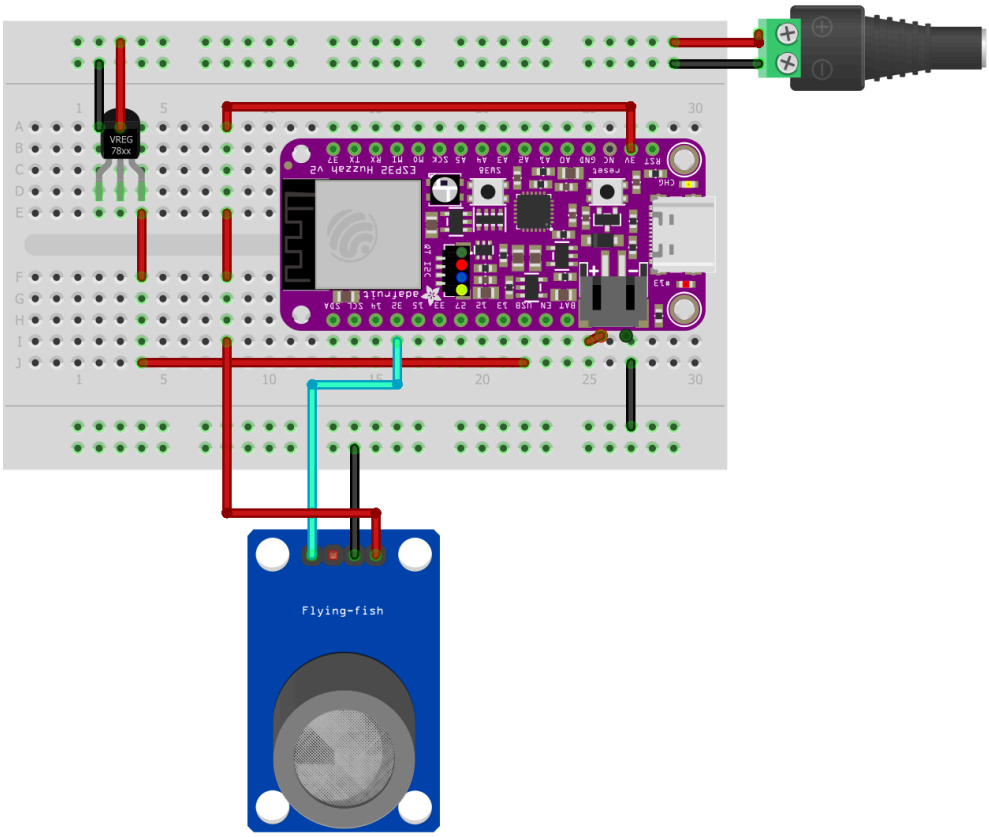


Figure 10: carbon dioxide sensor layout

Code for MQ135.

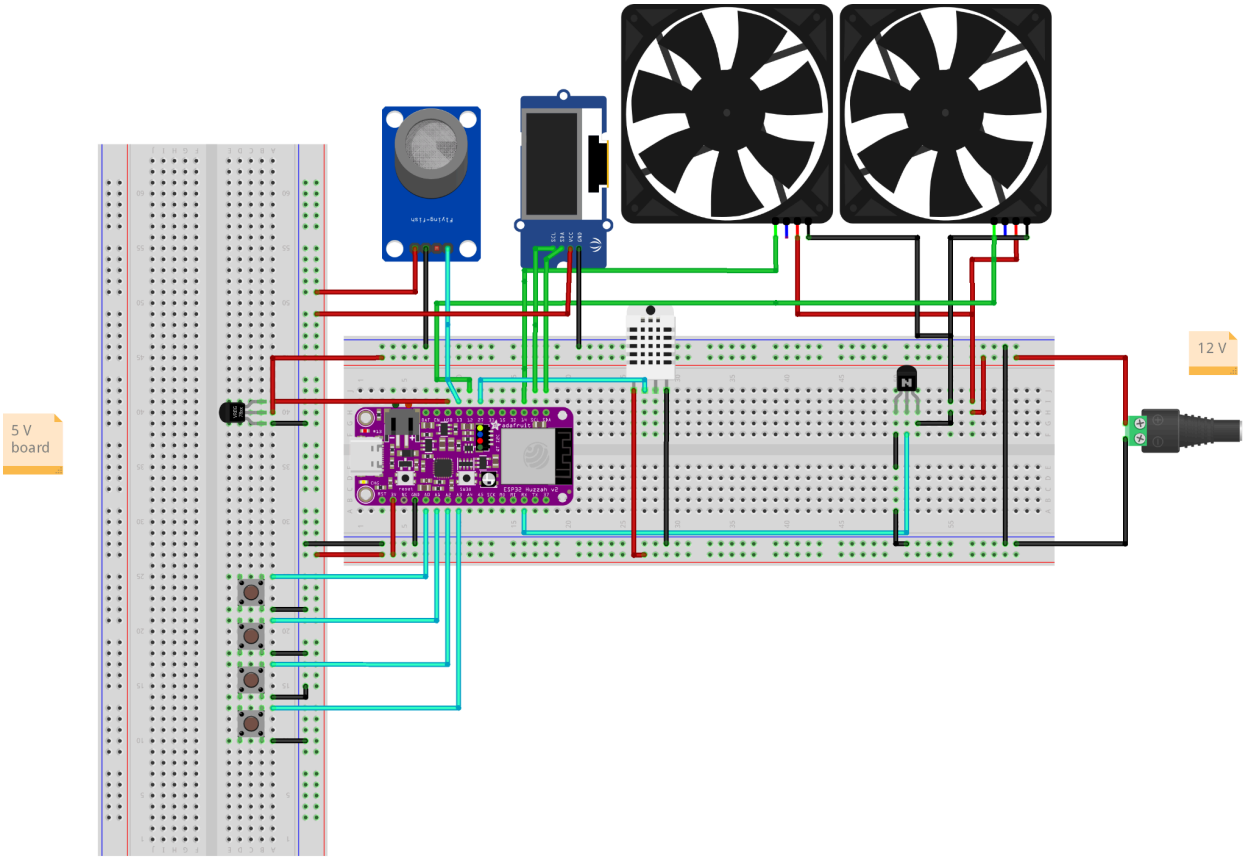


Figure 11: final wiring diagram

The layout is color-coded, so each power wire is represented in red, ground is black, I^2C and PWM are in green, and any signal is in cyan.

We can call this our “main system” since it has every component on it. We started by connecting the fans to the 12 volts and the DHT11 (temperature and humidity) sensor, MQ135 (carbon dioxide sensor), and the LCD needed to be powered by 5 volts. The PWM pins of the fans need to be connected to pins that support PWM to control its speed. We used a MOSFET to completely shut the fans off since we can not send a 0 Hz signal via the PWM, so when we send an off (low) signal to the gate of the transistor it completely cuts power from the fans thus they are fully off. The buttons are used to navigate through the settings from the LCD which need to be connected to the ground so that if it sends a low signal to the microcontroller it senses which button was pushed and does what it needs accordingly. As we can see

from Figure 11 which is the final combined layout, this system is the heart of this project. We had to make sure that the fans were connected to pins that support PWM (Pulse Width Modulation) so we could control the speed of the fans. Also, we can not forget to connect the LCD to pins that support I2C (Inter-Integrated Circuit protocol).

The dashboard was hosted on the ESP32 itself; we used the ESP-DASH library for the website and modified it to fit our needs. As soon as the ESP gets powered up it will cast an access point so the user can connect it to the wifi and host the website. The website has a clean and simple interface that is user-friendly and easy to navigate which can be seen from Figure 12.

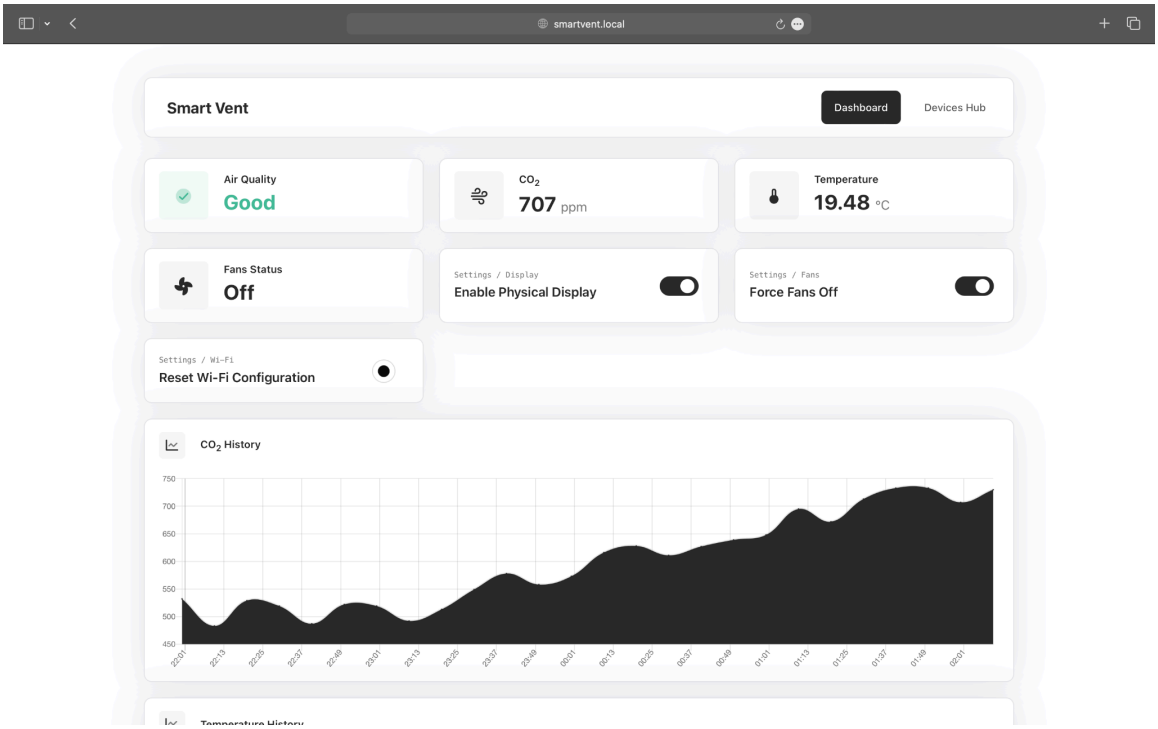


Figure 12: Dashboard

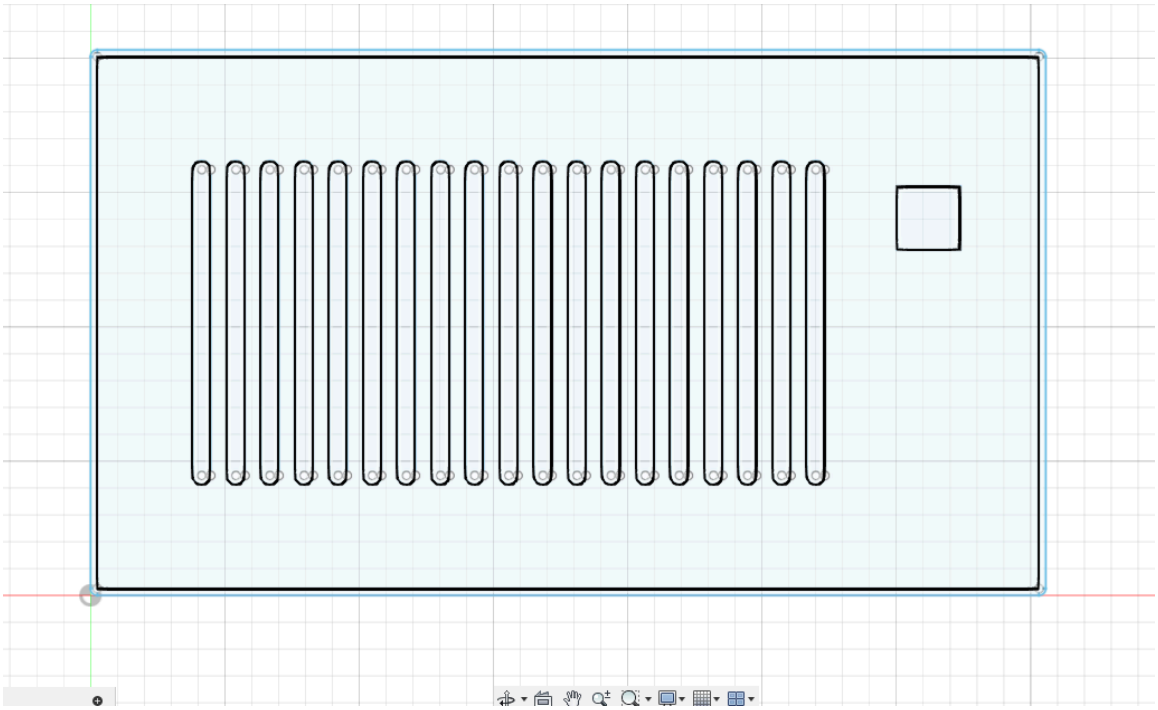


Figure 13: 1st Iteration Grille design

Figure 14: 1st Iteration Printed front face

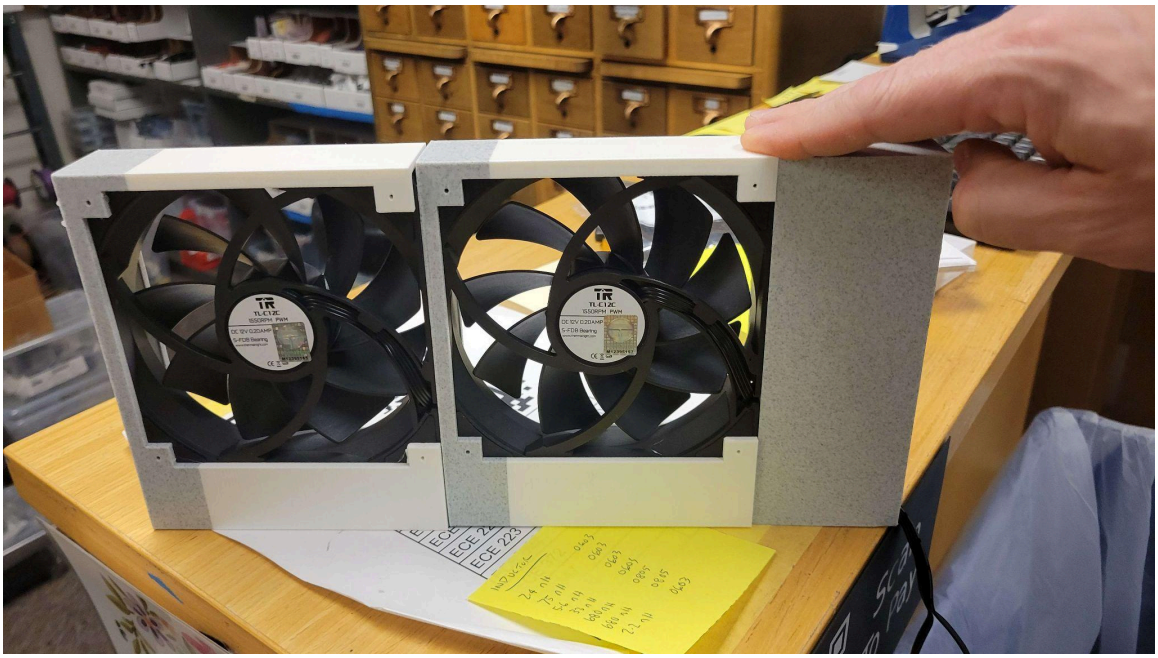


Figure 15: 1st Iteration Fans casing

Figure 16: Final Grille Design

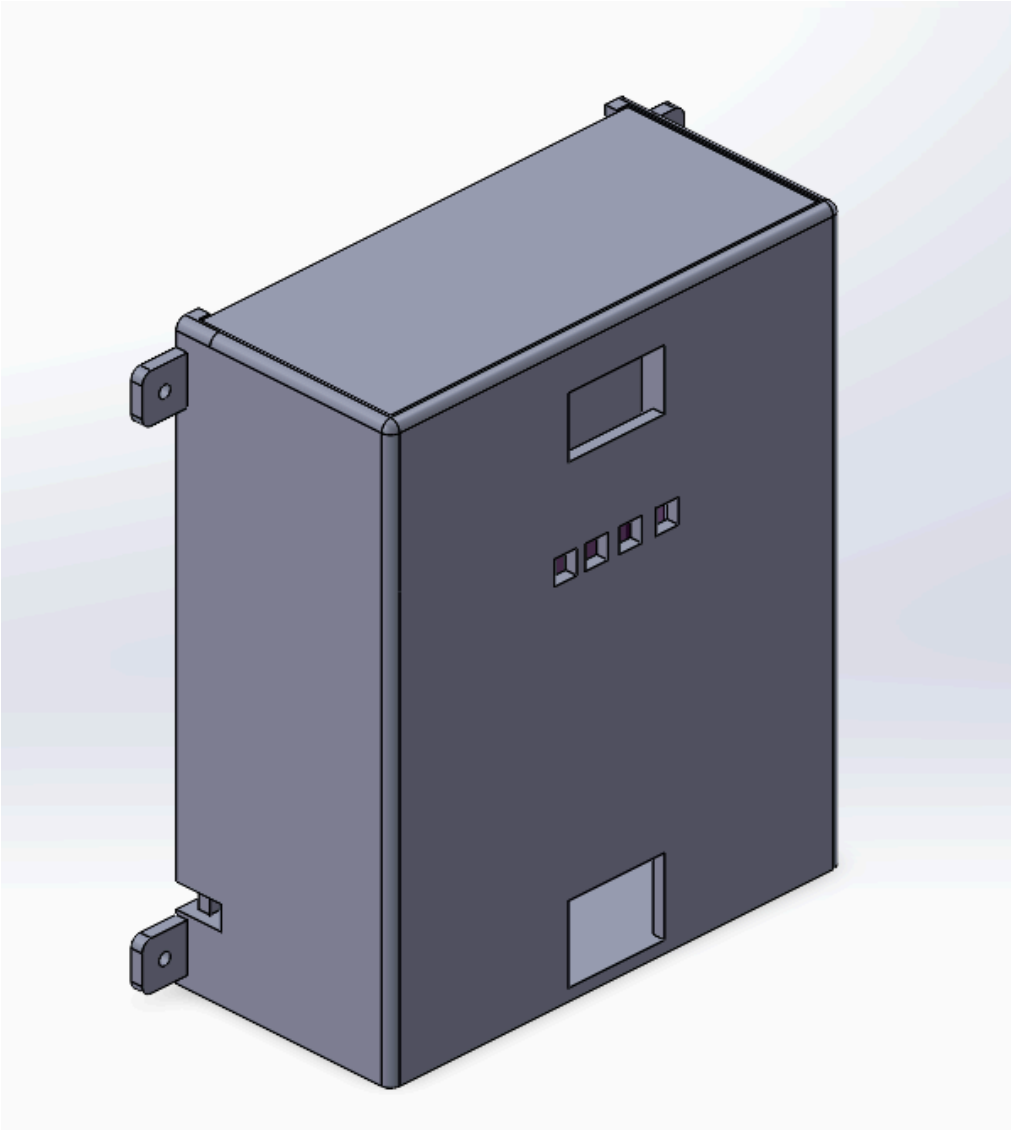


Figure 17: Controls Enclosure (Front)

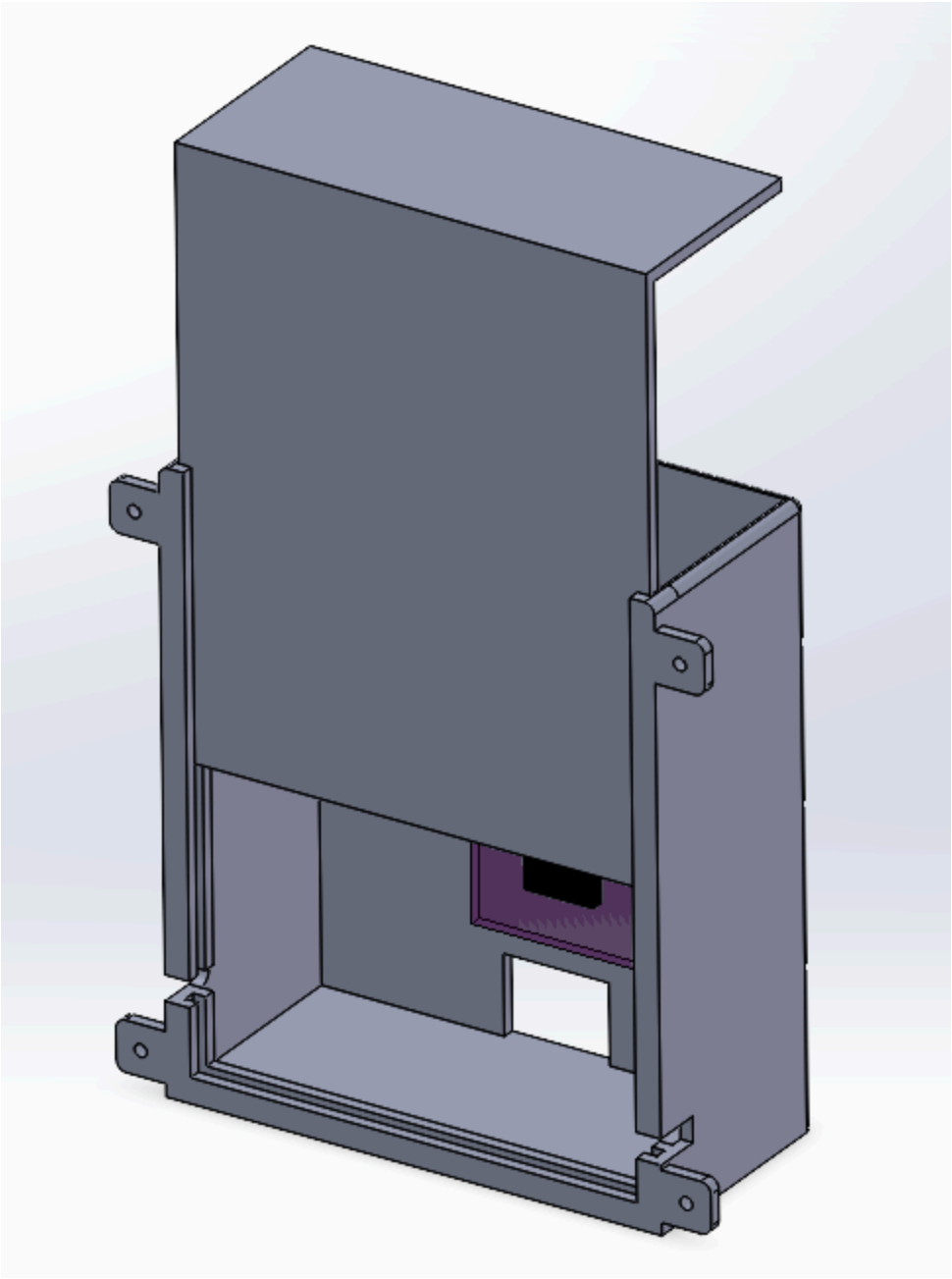


Figure 18: Controls Enclosure (Back)

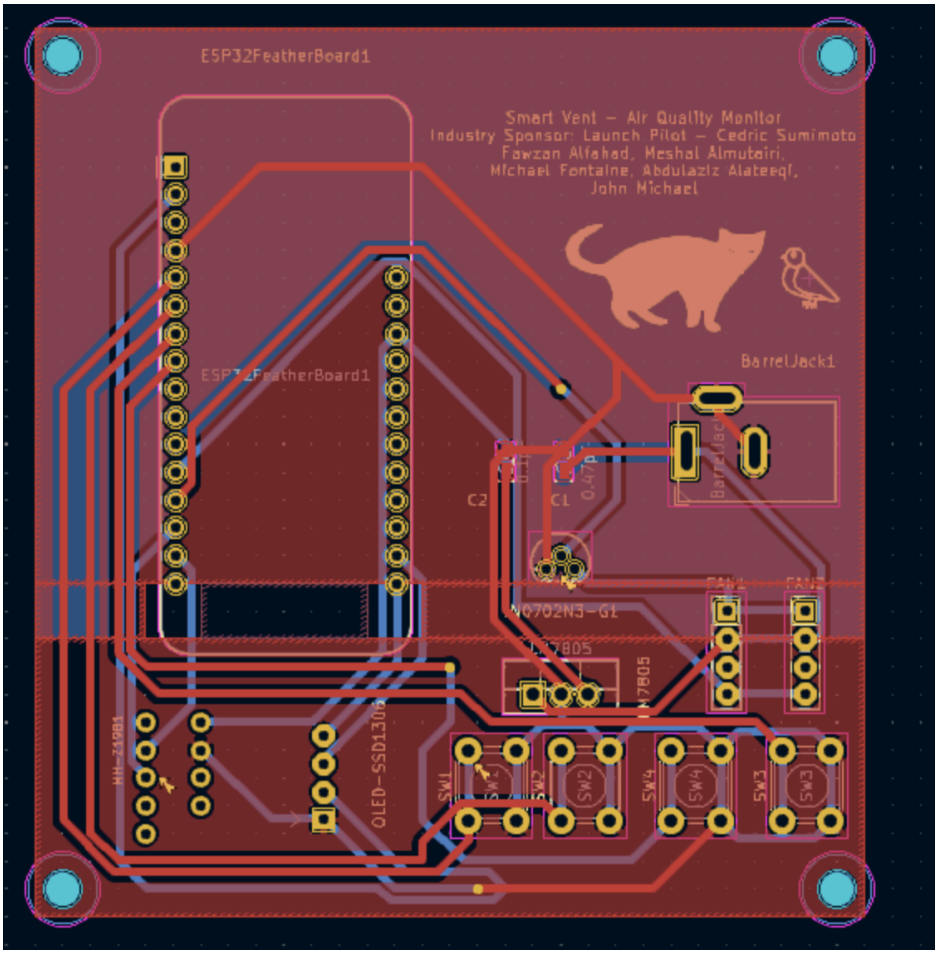


Figure 19: Final iteration PCB

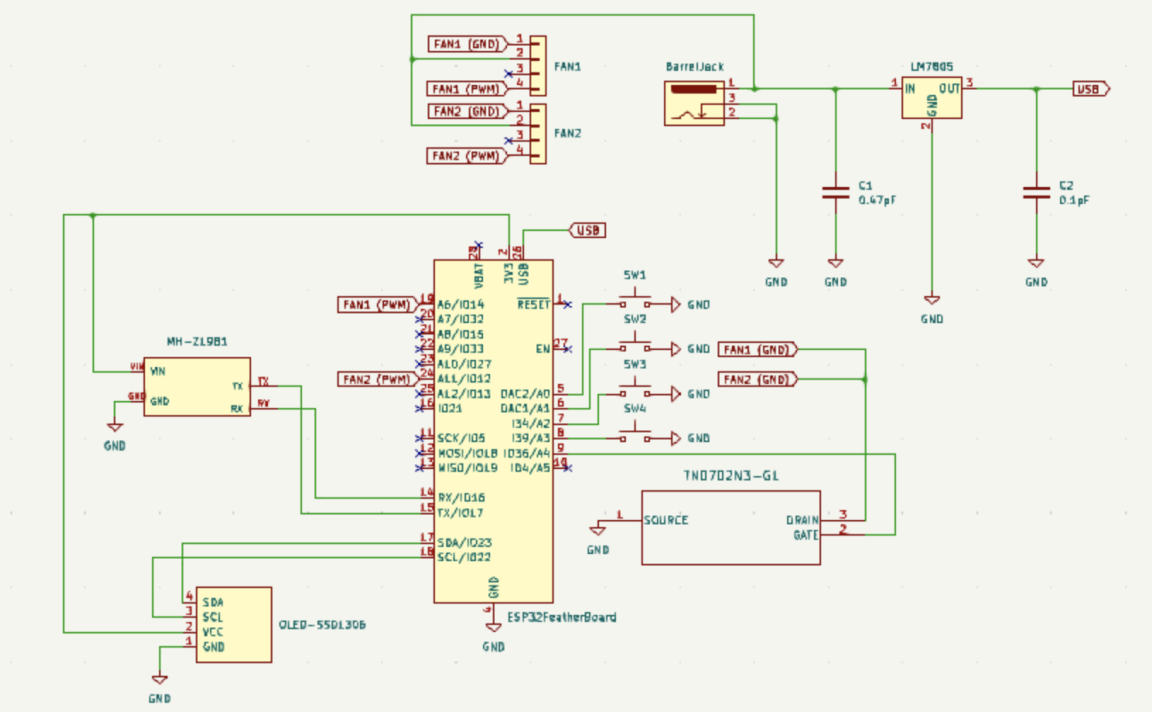


Figure 20: Final schematic

For software, we used two methods to lay out our product design. Initially, we designed all of our high-level layouts using Fritzing. To implement our high-level Fritzing design, we used Flux.ai to build our low-level schematic and PCB design. We chose Flux.ai because it allows multiple people to collaborate on the same PCB layout simultaneously, similar to Google Docs for PCBs.

However, it lacked the necessary components for our project, and when you try to build a component from scratch, you can't find the necessary tools. Therefore, we switched to KICAD. In KICAD, you can find every component you need from the library or the internet. However, if it's from the internet, you need to know how to install the libraries in your KICAD. KICAD is a great tool for debugging and testing, which verifies your schematic for any errors. Honestly, all you need is the schematic. Once the schematic is ready and error-free, you can easily work on the PCB design. However, before working on the PCB design, make sure to include every necessary footprint of each component to avoid complications. In the PCB design, lines will

appear to guide the user through the connection process, which helps a lot. Once every pin is connected, the dimensions of the PCB and measurements must be taken into consideration. For our project, we faced some issues with the dimensions of the PCB because we didn't connect the pins of each component in the schematic in an extremely organized way. Thus, our PCB is a little bit bigger than we expected it to be. We could have made our PCB smaller, but we're still satisfied with what we have.

For programming the microcontroller, we used Visual Studio Code, which is easy to use and user-friendly. Additionally, we used SOLIDWORKS to design the front face of the vent.

Figure 21: Completed Smart Vent

Software Discussion

Our system is developed in C++, and the software architecture is organized into six primary classes, each serving distinct functions:

1. **OLED Class**: Manages the display on a 128x64 OLED screen, including both the primary interface that shows sensor readings and various options menus.
2. **webserver Class**: Facilitates the internet connectivity of the device. It establishes an access point to allow user interaction for Wi-Fi configuration.

Additionally, it supports host election and communication across multiple devices, enhancing scalability and interaction.

3. **mhz19b Class**: Operates the NDIR CO_2 sensor, updating the sensor readings at set intervals. This class ensures real-time monitoring and data accuracy.
4. **Fan Class**: Controls the fan operations, adjusting speeds based on the CO_2 levels detected by the sensor or user-defined settings for manual override.
5. **dashboard Class**: Utilizes the ESP-Dash library to create a local dashboard that visually represents the device's status and sensor data, enabling user-friendly interaction.
6. **buttons Class**: Handles input from physical button presses, with logic to perform actions based on the current device state and user inputs.

This modular design enhances the system's extensibility and maintainability, allowing each component to be developed and debugged independently while ensuring cohesive operation.

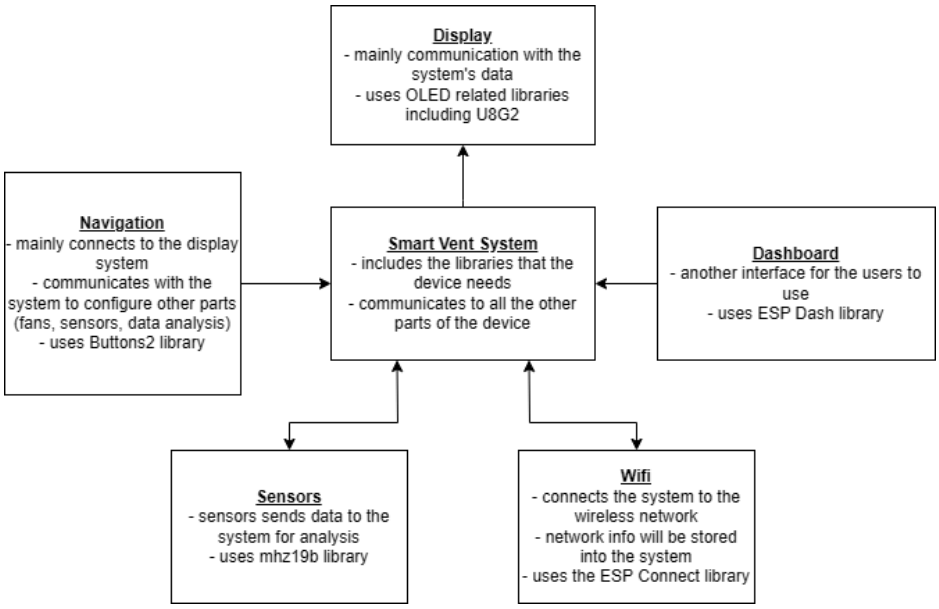


Figure 22: System diagram

Testing

We decided to do a bottom-up test first so that when we integrate the components and put the system all together just like in the figure.

The first part of the test is to check whether our device works the way it is built - voltage, readings, displays, and buttons. These were done mostly separately by each member of the team: Michael with power, Meshal with the carbon dioxide sensor, Aziz with the fans, John Michael with the navigation, and Fawzan with the initial sensor readings. These were done successfully as each single part is significant to the success of the device.

Everything went as planned and according to the requirements. Overall, the test was successful.

Results

- Using MPQ135 - Replaced with MHZ19B

Our main purpose was to detect the Carbon Dioxide with a sensor to display to our customers. We started with the MPQ135 and got it working after hard troubleshooting with calibrating the sensor. During the troubleshooting phase, our team decided to keep looking for better and more efficient sensors for mass production. The MPQ135 works great; however, the calibration time for the sensor to cook is over 12 hours, which is not mass production efficient. After a couple of weeks of research, our team found the MHZ19B. The sensor has better calibration than the MPQ135 with a calibration time of 3 minutes.

- Smart Device - WIFI + Dashboard

Our device can connect to a local wireless network and create its own access point for its dashboard feature. The dashboard is an online tool the user can use to configure and see the status of the vent from their phone or computer.

- Master/Slave - Multi-Device Purpose

With the use of the wireless connection and dashboard feature, multiple Smart Vents can connect to the same WIFI and the same access point to be configured to display their own status.

- 12-volt system - For the fans.

An issue that must be solved was to run both 12-volt fans with a 3.3-5 volt system (ESP32). We had to use a voltage regulator that takes in 12 volts of power to power the fans and lowers it to 3-5 volts for the processor.

- Flexible Mount - OLED Screen

One of our main targets was to keep the product adaptable to the user's comfort. It is nice to see a screen without connecting to a browser just to see a simple number. Our team decided to have the display isolated from both the vent and the power plug due to the location of both when using the product. Thus allowing the user to place where they would want to see numbers.

Post Mortem

What Worked?

Sensor Integration: The integration of CO_2 and temperature sensors worked really well, providing accurate real-time data for air quality.

User Interface: The display screen and physical buttons for navigating the system were user-friendly and intuitive.

Fan Operation: The 120mm dual-sized fans effectively circulated air, maintaining healthy CO_2 levels in the room.

Web Dashboard: The web dashboard worked really well, it displayed real time readings and provided the user with remote control.

What Was Great?

User Feedback: Incorporating the feedback into our design iterations improved the system's usability by a lot.

Power Saving Features: Ensuring the system's display gets dimmer if the user is not using the buttons.

Optional Connectivity: Adding a Wi-Fi option provided additional flexibility and smart home integration.

What Didn't Work?

Sensor Calibration: The calibration process for the first CO_2 sensor (MQ135) required lots of manual work and the process was complicated, which will be an issue for mass production.

What Sucked?

Initial Dashboard Design: The first version of our dashboard was not as intuitive as we hoped, we had to change the design.

Troubleshooting Time: We spent a lot of time troubleshooting pin issues, which delayed other aspects of the project since we had to redesign the layout multiple times.

What Would You Do Differently?

Sensor Research: Do more in-depth research on sensor calibration and maintenance requirements.

What Do You Wish You Would Have Known Back in December?

Gas Sensors Calibration: we wish we had known that gas sensors usually take a lot of time and effort to calibrate if you are working with a small budget.

User Preferences: Early insights into user preferences and menu options for such systems.

What Would You Do If You Had More Time?

Additional Sensors: Adding more types of gas sensors (e.g., VOCs, radon) would provide a more comprehensive air quality analysis.

Additional Features: Implementing features such as notification systems to alert users if the air quality is not safe.

What do you think the next team should do?

User Experience Enhancements: Further improve the user interface based on additional user feedback.

Expand Sensor Capabilities: Integrate additional sensors to provide a broader range of air quality data.

Project Resources

We used GitHub for almost everything in this project, we had our repository setup so that we could push and pull our code and version control link to our repository. We also used different libraries for our sensors and dashboard, for the DHT12 (temperature and humidity sensor) we used DHT12 sensor library. For our MQ135 (Carbon Dioxide sensor) we used the MQ135 library. Our dashboard utilizes the ESP-DASH library is extremely user-friendly and easy to adjust.

For time management and setting goals/milestones, we used Asana. When it came to communication we mainly used Discord to meet and send messages about our tasks and what we had to do. We established a private discord server for our team to use.

List of tools:

- Visual Studio Code (version 1.88)
- Arduino IDE (version 2.3.2)

- Fritzing (version 1.0.1)
- Flux.ai
- KiCad (version 7.0.0)
- SOLIDWORKS
- Other resources:
    - Google Drive
    - GitHub
    - BOM link.

Our Budget was $500 and the sponsor was willing to increase it if we needed it, but luckily it was enough for us.

Conclusion

Our smart ventilation system project aimed to enhance indoor air quality by using sensors and user-friendly controls. We successfully developed a product that detects CO_2 levels and temperature, automatically activating 120mm fans to maintain a healthy environment. Our system includes a clear OLED display screen and physical buttons for straightforward navigation.

We have also integrated Wi-Fi connectivity to provide flexibility in monitoring and controlling the system, making it easier for smart home integration. Having multiple iterations in this project helped us a lot, the feedback that we received from our industry sponsor and faculty advisor along with the continuous testing ensured that we delivered a functional and effective product.

Appendix I: Smart Vent User Manual - Launch Pilot

User Manual Table of Contents

Product Introduction..... 2

Device Overview..... 3

Quick Start..... 3

Built-in Display..... 4

How to Connect to Wi-Fi..... 5

Customer Support..... 10

Product Introduction

The Smart Vent is designed to monitor and improve air quality in its surroundings. It features a sensor that detects carbon dioxide levels and temperature. When carbon dioxide levels exceed the set limits, the Smart Vent's fans will activate.

Device Overview

- Sensor: Measures the CO_2 levels and ambient temperature in the surrounding area.
- Display Screen: Displays the CO_2, temperature,
- Buttons: Navigate through the display options.
- Built-in Wi-Fi: Provides access to remote dashboard
- Power plug: Connects to the device barrel jack.

Quick Start

I. Mount the Device: Attach the Smart Vent device to the wall near your vent or duct.

II. Power the Device: Plug the power adapter into an electrical outlet.

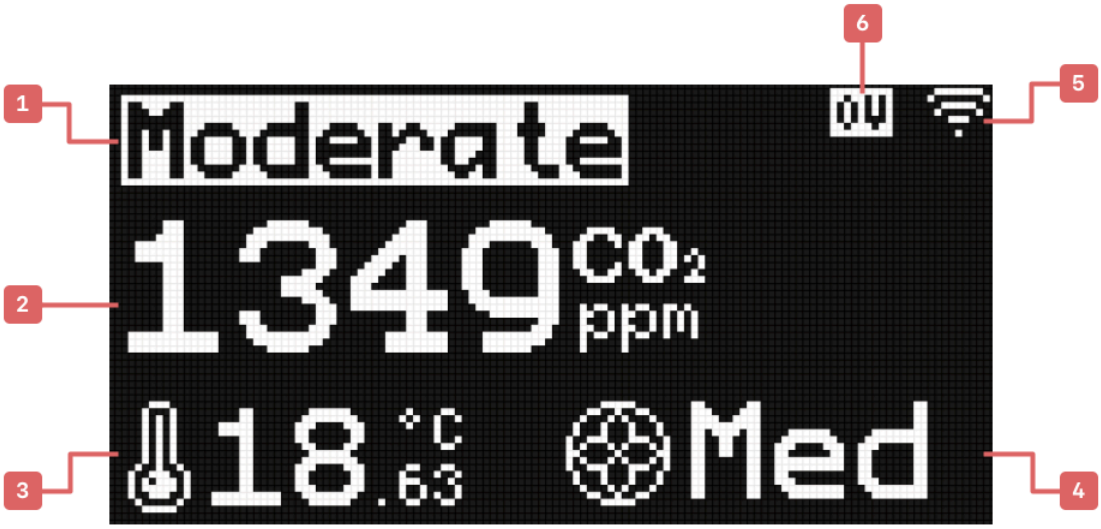
III. Initial Display: The display screen will light up, showing the main interface.

IV. Stabilization Period: Allow the device one to two minutes to stabilize.

V. Reading Display: When the readings begin to appear, your device is correctly set up.

Built-in Display

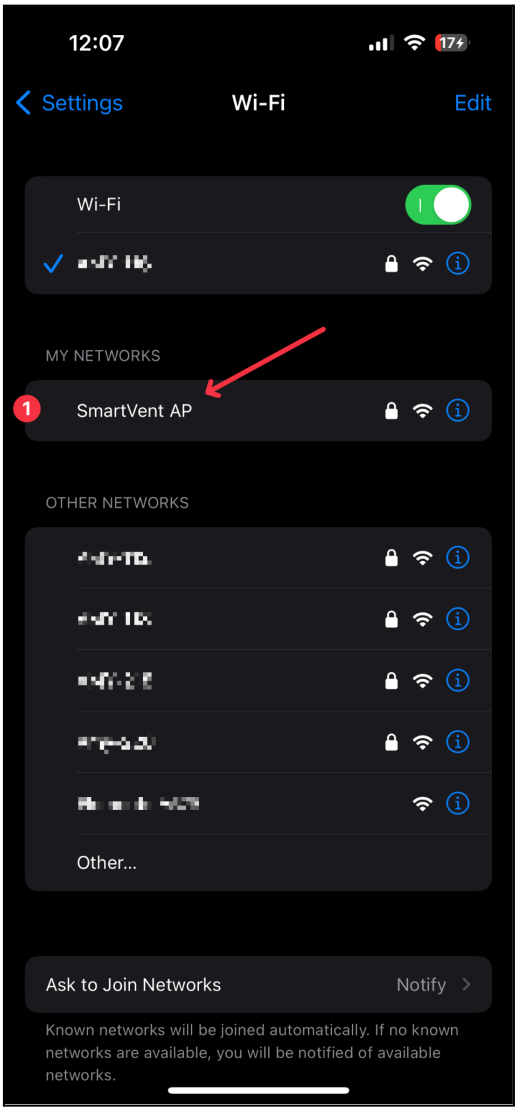
1. Air quality threshold indicator. The following are the threshold levels:
    a. **Good**: below 800 ppm.
    b. **Moderate**: 800 to 1400 ppm.
    c. **Poor**: 1400 to 2000 ppm.
    d. **Unhealthy**: 2000 to 3000 ppm.
    e. **Hazardous**: above 3000 ppm.
2. CO_2 concentration level in ppm (parts per million).
3. Temperature of the air in Celsius or Fahrenheit.
4. Current speed of the fans (Off, Low, Med, High, Max).
5. Wi-Fi connection indicator.
6. Fans override indicator

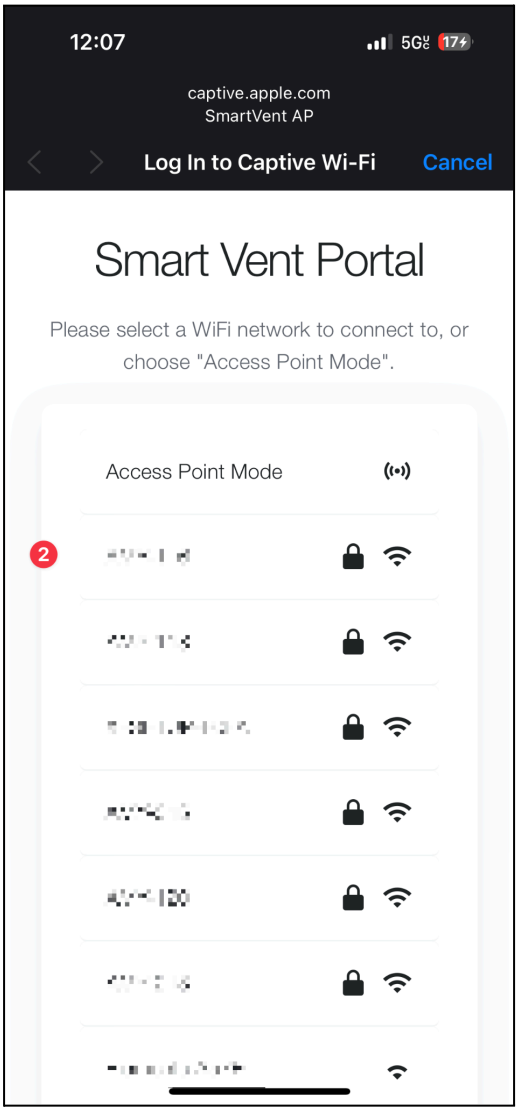


How to Connect to Wi-Fi

1. Once the smart vent is powered on, head to your Wi-Fi settings on a compatible device and connect to the smart vent network under "**SmartVent AP**" SSID using the password "**capstone**"

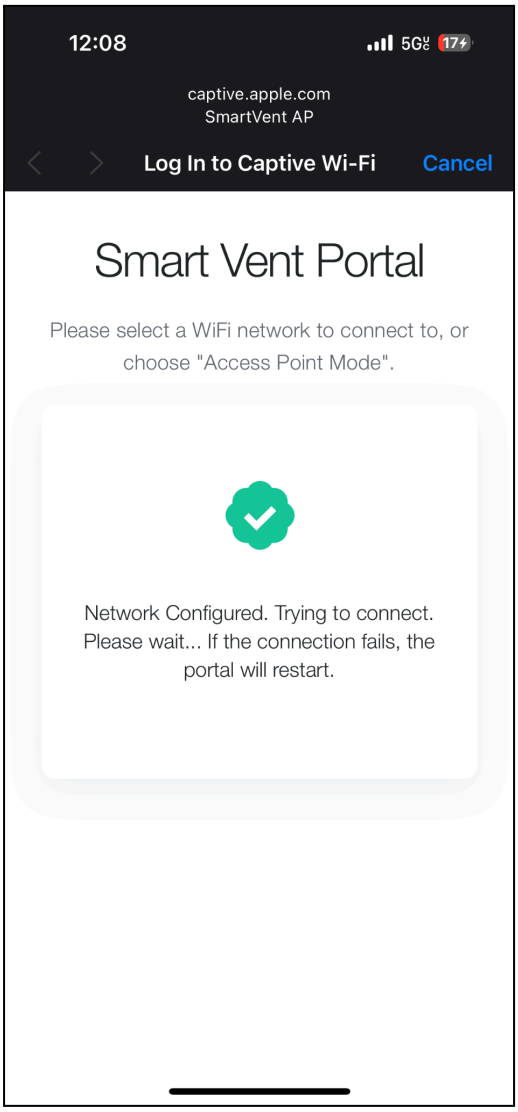
2. Once connected, the smart vent portal should pop up on your device, using the portal select your home Wi-Fi you wish to connect the smart vent to.



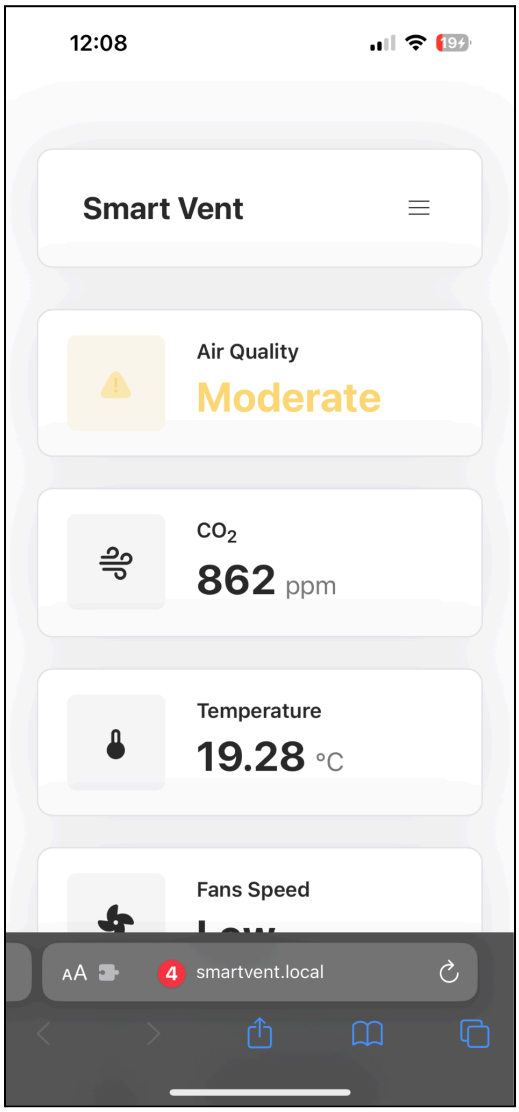


3. You will be prompted to enter your home Wi-Fi password, once filled click on connect to connect the smart vent to your home network.

4. Once connected successfully, your device should automatically disconnect from the smart vent access point and portal will be dismissed.

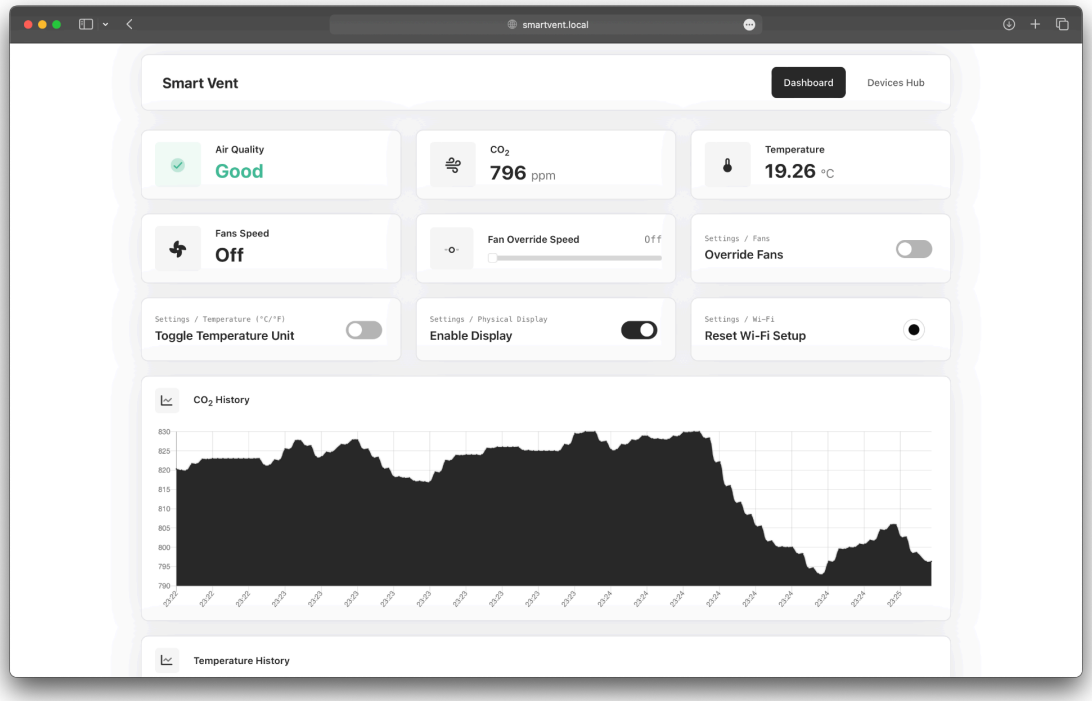


5. Then connect your device to your home Wi-Fi and enter the web address '**smartvent.local**' in your browser to access the remote dashboard.



Wi-Fi Remote Dashboard

The smart vent dashboard can be accessed using 'smartvent.local' web address on your browser if your device is connected to the same Wi-Fi network as the smart vent. The dashboard offers a remote overview of the smart vent and displays the same readings as the display on the smart vent.



Technical Specifications

- Power Input: 12V - 1A via barrel jack
- Connectivity: Wi-Fi (2.4GHz)
- Nondispersive infrared (NDIR) sensor to measure CO_2 concentration
- CO_2 Sensor Range: 400-5000 ppm
- Temperature Range: 0°C to 50°C (32°F to 122°F)

Safety Information

- Indoor Use Only: The Smart Vent is designed for indoor use only.
- Avoid Moisture: Do not expose the device to water or moisture.
- Avoid Extreme Temperatures: Do not put the device in an extremely hot or cold environment (0°C to 50°C).
- Power Safety: Use only the provided power adapter (DC 12V - 1A).

Customer Support

- **Email**: support@smartvent.com
- **Phone**: +1 (800) 123-4567
- **Website**: www.smartvent.com/support

Appendix II: Getting Started with Software Development

To develop, compile, and run this project, you will need the following primary tools: Visual Studio Code v1.89.1 and PlatformIO v3.3.3. Below are the detailed steps to get started:

1. Install Visual Studio Code:
    a. Download Visual Studio Code from this link and select the appropriate version for your operating system.
    b. Follow the installation instructions for your platform.
2. Set Up PlatformIO:
    a. Open Visual Studio Code.
    b. Navigate to the Extensions tab on the sidebar (you can also press Ctrl+Shift+X).
    c. In the Extensions Marketplace, search for "PlatformIO" and install the extension.
3. Clone the Project Repository:
    a. The project's source code is hosted on GitHub. Clone the repository using the following command:

    `git clone https://github.com/mshll/SmartVent.git`

4. Open the Project in Visual Studio Code:
    a. Open Visual Studio Code and select File > Open Folder.
    b. Navigate to the cloned repository folder and open it.
5. Configure PlatformIO:
    a. The project is configured using the platformio.ini file included in the repository. This file specifies the necessary settings for compiling and uploading the code to the ESP32 microcontroller.
    b. Ensure that your ESP32 microcontroller is connected to your computer.
6. Compile and Upload the Code:
    a. In Visual Studio Code, navigate to the PlatformIO tab on the sidebar.

b. Click on the “Build” button to compile the project.
c. Once the build is successful, click on the “Upload” button to flash the compiled code to the connected ESP32 device.

Additional Information

- **PlatformIO**: PlatformIO is an open-source ecosystem for IoT development. It provides a unified interface to work with various development platforms, including Arduino, Espressif, ARM mbed, and more. PlatformIO simplifies the process of setting up development environments, managing libraries, and deploying firmware to microcontrollers.
- **Arduino Requirement**: The project is designed to be used with an ESP32 microcontroller and does not require the Arduino IDE. PlatformIO handles all the necessary dependencies and builds configurations.

Appendix III: Test Plans

Top Down Test Plan v1.0

Date of Test:	2024/04/26
Tester:	The entire team

Purpose

To test the entire integrated system against its requirements by performing a top-down test.

Equipment Needed

- System Equipment
    - 12 volts power supply
    - 2 Protoboards
- Test Equipment
    - Laptop
    - Phone
    - Commercial carbon dioxide sensor

Pre-Test Setup

1. Plug in the wall plug to supply 12 volts.
2. Prepare a phone to connect the esp32 to the wifi
3. Check the power supply connection and the wire connections

Top Down Test Steps

1. Put on the room ventilation
2. Test 1: Have a control near the sensor (vent or plug) (Small Room)
    a. Turn on and set the system
    b. Record every 30 seconds?

3. Test 2: Have the control in the center (Small Room)
    a. Turn and set the system
    b. Record every 30 seconds?
4. Things to look out for:
    a. When the fans turn on/off.
    b. The difference in carbon dioxide level from the device
5. Test 3 & 4: Repeat tests 1 & 2 in a Bigger Room
    a. Expectation: Bigger room harder to see the difference in carbon dioxide level.

Bottom-Up Test Steps

1. Once we turn on the power switch.
    a. ESP32 should turn on as well as the display.
    b. We must see real-time readings from the sensors in the display
    c. Connect the ESP32 to the wifi successfully
    d. Visit the dashboard and control the system successfully
    e. Navigate the system settings.
    f. Control the fans' speed and turn them off completely
    g. Breathe near the sensor and make the fans kick-off

Post-Test Teardown

1. Unplug the power supply
2. Remove it from the wall

Test Author: Michael Fontaine							
	Test Case Name:	Power test		**Test ID #:**			01
	Description:	This test case will test power that goes to the ESP32 as well as the fans.		**Type:**			□ white box
Tester Information							
	Name of Tester:	Michael Fontaine		**Date:**			04/21/2024
	HW/SW Version:	1.0		**Time:**			N/A
	Setup:	Setup the connection as in figure 5					
TEST	**INPUTS**	**EXPECTED OUTPUTS**		**PASS**	**FAIL**	**N/A**	**Comments**
1	12 volts	-Fans turn on -ESP32 turns on		x			The voltage regulator successfully converts the input voltage to 5 volts
	Overall test result:			**x**			

Table 1: Power test

Test Author: Meshal Almutairi				
	Test Case Name:	CO_2 sensor readings	**Test ID #:**	02
	Description:	This test case will test the MQ135 sensor and will display the real-time readings	**Type:**	□ white box
Tester Information				
	Name of Tester:	Meshal Almutairi	**Date:**	04/04/2024
	HW/SW Version:	1.0	**Time:**	N/A
	Setup:	Setup the connection as in figure 9, and use the code below it		

TEST	**INPUTS**	**EXPECTED OUTPUTS**	**PASS**	**FAIL**	**N/A**	**Comments**
1	Normal room carbon dioxide	Normal readings	x			The readings are similar to the commercial sensor
2	Breathe near the sensor	The readings go up	x			
	Overall test result:		**x**			

Table 2: CO_2 sensor test

Test Author: Fawzan				
	Test Case Name:	DHT11 readings	**Test ID #:**	03
	Description:	This test case will test the DHT11 sensor and will display the real-time readings	**Type:**	□ white box
Tester Information				
	Name of Tester:	Fawzan	**Date:**	04/01/2024
	HW/SW Version:	1.0	**Time:**	N/A
	Setup:	Setup the connection as in figure 8, and use the code below it		

TEST	**INPUTS**	**EXPECTED OUTPUTS**	**PASS**	**FAIL**	**N/A**	**Comments**
1	Normal room temperature	Normal temperature readings	x			
2	Use a lighter near the sensor	The readings go up	x			
	Overall test result:		**x**			

Table 3: DHT11 sensor test

Test Author: Meshal Almutairi							
	Test Case Name:	Display and dashboard test		**Test ID #:**			04
	Description:	This test case will test the DHT sensor and will display the real-time readings		**Type:**			□ white box
Tester Information							
	Name of Tester:	Meshal Almutairi		**Date:**			04/15/2024
	HW/SW Version:	1.0		**Time:**			N/A
	Setup:	Setup the connection as in figure 6, and use the code below it					
TEST	**INPUTS**	**EXPECTED OUTPUTS**		**PASS**	**FAIL**	**N/A**	**Comments**
1	5 volts	The display should turn on and display whatever in the code		x			
2	Wifi credentials	The dashboard should be up and running		x			
	Overall test result:			**x**			

Table 4: display and dashboard test

Test Author: Abdulaziz				
	Test Case Name:	Fans speed test	**Test ID #:**	05
	Description:	This test case will ensure that the PWM works and we can control the fans speed	**Type:**	□ white box
Tester Information				
	Name of Tester:	Abdulaziz	**Date:**	04/04/2024
	HW/SW Version:	1.0	**Time:**	N/A
	Setup:	Setup the connection as in figure 5, and use the code below it		

TEST	**INPUTS**	**EXPECTED OUTPUTS**	**PASS**	**FAIL**	**N/A**	**Comments**
1	Send different frequencies	The fans should change their speed every few seconds	x			
2	Sending a signal to the MOSFET	The fans should completely turn off	x			
	Overall test result:		**x**			

Table 5: fans speed test

Test Author: John Michael				
	Test Case Name:	Buttons and navigation test	**Test ID #:**	06
	Description:	This test case will ensure that the buttons work and can navigate the settings	**Type:**	□ white box
Tester Information				
	Name of Tester:	John Michael	**Date:**	04/19/2024
	HW/SW Version:	1.0	**Time:**	N/A
	Setup:	Connect one side of the buttons to ground and the other one to any digital pin on the ESP32		

TEST	**INPUTS**	**EXPECTED OUTPUTS**	**PASS**	**FAIL**	**N/A**	**Comments**
1	Press different buttons	The settings menu should change	x			
2	Control the settings	Different settings should apply	x			
	Overall test result:		**x**			

Table 6: buttons and navigation test