UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Temperature and humidity monitoring

Project Documentation

Grad Laurentiu-Calin
Group 30435

Computer Science
December, 2024

Contents

1 Introduction: Purpose and Motivation

I chose this project because I was familiar with temperature measuring from another assignment. This topic seems interesting to me because there is a wide variety of sensors on the market with different specifications that can provide input data for a computation pipeline. I have used such kind of input to create a temperature histogram before (a great visual tool to detect average temperatures over a period of time and other statistics-related values), and now I want to implement a "feedback" circuit that has three main parts:

- *INPUT* some temperature sensor that acts as input source.
- *OUTPUT* different kinds of visual indicators such as LEDs and displays or transducers such as a fan.
- *CORE LOGIC* Arduino (C++) code that is to be uploaded on a Uno Board to manage the input and output.

Multiple solutions are available on the web, however I chose to let my creativity and skills decide how this job was developing. I tried to use multiple types of components to get the best out of this assignment. My purpose was to get a grasp on how to control the things that are most likely to be used in an Arduino project.

2 Bibliographic Research

There are multiple implementation and design project ideas related to greenhouses, temperature or humidity monitoring in different media that are available on-line, some interesting ones that I have encountered are: [1], [2], [3]. However, I chose to implement this in my own way, without taking inspiration from any of these. Consequently, my research focused on searching, finding, purchasing and working with sensors, development boards, electrical circuits and diagrams.

What my project intends to be is a translator of outside temperature to people. One or more sensors measure the temperature and relative humidity that trigger changes across output devices. Three such modules should be used:

- an LCD screen that displays a message to the user paired with an RGB LED based on the recorded temperature.
  - ("Brrr!", BLUE) when the temperature is below 5°C
  - ("Spring-like!", YELLOW]) when the temperature is between 5°C and 20°C
  - ("Just right!", GREEN) when the temperature is between 20°C and 28°C
  - ("It's too hot!", RED) when the temperature is between 28°C and 32°C
  - ("Overheat!", MAGENTA) when the temperature is above 32°C
- an 8-LED module that lights up progressively based on the recorded relative humidity.
  - first LED if the relative humidity is below 5%

  - first two LEDs if the relative humidity is between 5% and 14.3%
  - first three LEDs if the relative humidity is between 14.3% and 28.6%
  - first four LEDs if the relative humidity is between 28.6% and 42.9%
  - first five LEDs if the relative humidity is between 42.9% and 57.2%
  - first six LEDs if the relative humidity is between 57.2% and 71.5%
  - first seven LEDs if the relative humidity is between 71.5% and 85.8%
  - first eight LEDs if the relative humidity above 85.8%
- a fan that has four power levels based on the temeprature
  - *TURNED OFF* if the temperature is below 20°C
  - *LEVEL 1 - below half power* when the temperature is between 20°C and 28°C
  - *LEVEL 2 - above half power, but not at full power* when the temperature is between 28°C and 32°C
  - *LEVEL 3 - full power* when the temperature goes above 32°C
- buzzer *-optional-*

A wide variety of temperature or temperature & humidity sensors is available on the market. I set my eyes on three of them:

- HR202 → humidity sensor.
- DHT11 → temperature sensor.
- DHT22 → temperature & humidity sensor.

The core differences between the above mentioned sensors are summarized in Table 1:

From 1 it is clear that the most reliable out of those three is the *DHT22*. Consequently, I chose this sensor to be the input source of my project.

As far as output devices are concerned, I managed to find a small 8-LED printed circuit board, an I2C compatible LCD and a 5V fan. In addition to these, a common cathode RGB LED and some resistors are required.

Feature	**HR202**	**DHT22**	**DHT11**
Type	Resistive Humidity Sensor	DIgital Humidity and Temperature Sensor	Digital Humidity and Temperature Sensor
Humidity Range	20% to 95% RH	0% to 100% RH	5% to 95% RH
Temperature Range	0°C to 60°C	-40°C to 80°C	-20°C to 60°C
Humidity Accuracy	±15% RH	±2% to ±5% RH	±5% RH
Temperature Accuracy	*Undefined*	±0.5°C	±2°C
Output	Digital	Digital	Digital
Power Supply	3.3V - 5V DC	3V - 6V DC	3V - 5.5V DC
Communication	Analog	One-wire	One-wire
Response time	Not Specified	2 seconds	2 seconds
Common applications	Weather monitoring, Dew detection	Weather stations, Greenhouse monitoring	Home weather stations, Environmental monitoring

Table 1: Comparison of **HR202**, **DHT22**, and **DHT11** sensors

3 Proposed Solution and Implementation

List of components:

- Arduino-Uno-compatible board and programming cable
- I2C LCD
- RGB LED
- 8-LED module
- DHT22 temperature & humidity sensor
- three 470ω resistors
- one 4k7ω resistor
- one fan
- about 30 (F-M, M-M, F-F) wires

I have chosen to implement my project in a bottom-up manner, meaning that I started with small parts that had nothing to do with each other at first glance. Once all these modules had been designed, implemented and tested, I moved to creating the inter-component logic. The main advantage my approach offered was to let me think in object-oriented style. Thus, each component (printed circuit boards, simple LED-s, temperature or humidity sensors) was wrapped inside an abstract object (class). Being a relatively complex project thanks to the amount of components that have different behaviors, the development process was accelerated because I had access to the high-level abstraction mechanisms provided by C++. By following the *SOLID* principles and some *good practice* industry rules, the final form of the source code appears to me as maintainable, clear to use and improve, and most importantly free of bugs, useless code and redundancy.

Down below is the main.ino core: the *setup* and *loop* methods. The first one creates the context and prepares the components to handle incoming events. Because each module is an object (in the OOP sense), the methods are small and easy to understand.

In the *loop* method I make use of the delta time trick to avoid halting the program with the *delay()* method. This way the resulting program feels smoother.

```
1  void setup()
2  {
3      myLed.setupLed();
4      myDHT22Sensor.setupSensor();
5      ledModule.setupLedModule();
6      LCD::getInstance().setupLCD();
7      myFan.setupFan();
8  }
9
10 void loop()
11 {
12     currentTime = millis();
13     deltaTime += currentTime - previousTime;
14     if(deltaTime > MAX_DELTA_TIME) {
```

```
        float newT = myDHT22Sensor.readTempearture();
        float newH = myDHT22Sensor.readHumidity();
        temperatureEventManager(newT);
        humidityEventManager(newH);
        deltaTime = 0.0f;
    }
    previousTime = currentTime;
}
```

Two important methods are not related to any object: *temperatureEventManager* and *humidityEventManager* whose body is presented below:

```
void temperatureEventManager(float temperature)
{
    uint8_t fanPowerFactor = 0;
    if (temperature <= 5.0f)
    {
        myLed.setColor(BLUE);
        LCD::getInstance().displayMessage("Brrr!");
    }
    else if (temperature <= 20.0f)
    {
        myLed.setColor(GREEN);
        LCD::getInstance().displayMessage("Spring-like!");
    }
        //...
    else {
        fanPowerFactor = 255;
        myLed.setColor(MAGENTA);
        LCD::getInstance().displayMessage("Overheat!");
    }
    myLed.displayColor();
    myFan.powerOn(fanPowerFactor);
}
```

Here, we simulate "pattern matching" to decide how the state of the output devices should change.

```
void humidityEventManager(float humidity)
{
    const int LED_MODULE_SIZE = ledModule.getNumberOfLeds();
    if(humidity > 100.0f)
        for(unsigned i = 0; i < LED_MODULE_SIZE; i++)
            ledModule.turnOffLed(i);

    bool whatToTurnOn[LED_MODULE_SIZE]{false};
    float relativeHumidityThresholds[LED_MODULE_SIZE] = {5.0f,
       14.3f, 28.6f, 42.9f, 57.2f, 71.5f, 85.8f, 100.0f};
    for(unsigned i = 0; i < LED_MODULE_SIZE; i++)
        if(humidity > relativeHumidityThresholds[i])
            whatToTurnOn[i] = true;
    for(unsigned i = 0; i < LED_MODULE_SIZE; i++)
        (whatToTurnOn[i]) ? ledModule.turnOnLed(i) : ledModule.
           turnOffLed(i);
```

```
}
```

Here, we create an array to keep track of our threshold values and another one to store the state that each led in the 8-LED module should have at the end of the iteration.

3.1 The DHT22 Sensor

DHT22 is a temperature and humidity sensor. It is a really popular measuring device because it has a low production cost and is easy to use. Thanks to this popularity a lot of open-source libraries that implement high-level operations already exist: *SimpleDHT, Adafruit Unified Library, etc.*

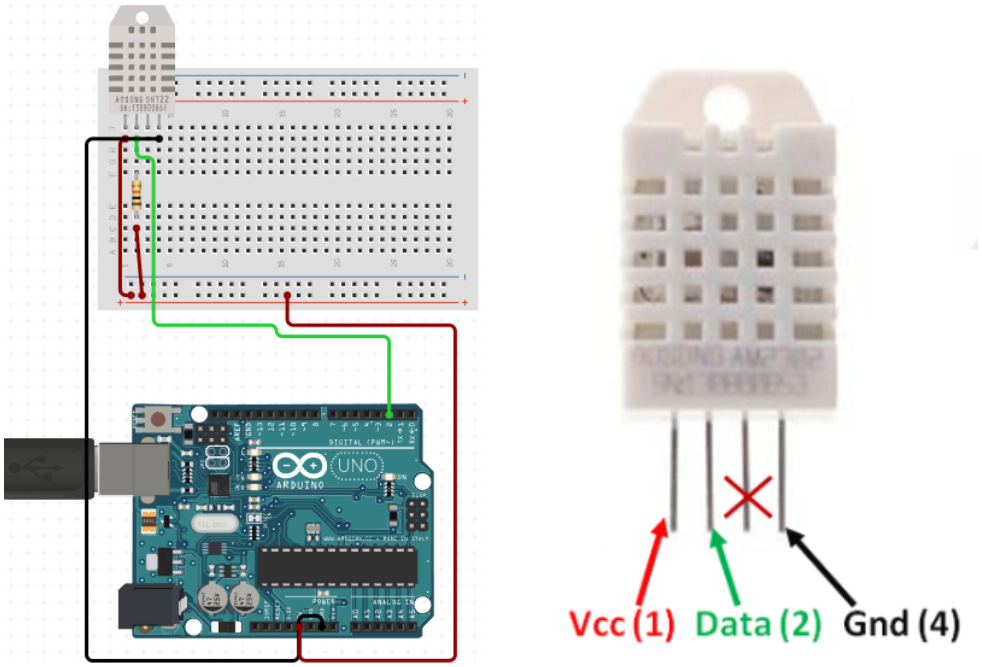

Figure 1: The DHT22 sensor Arduino diagram and a close-up of the component

```
#include <stdint.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

class DHT22Sensor {
    public:
        DHT22Sensor() : dht(12, DHT22) {};
        void setupSensor();
        void sensorLoop();
    private:
        // the pin on the Arduino board that is
        // connected to the data pin of the DHT22
        const int dataPin = 12;
        // sensor wrapper provided by
        // Adafruit_sensor.h
        DHT_Unified dht;
        // how many miliseconds we have to wait
```

```
19          //to read data from the sensor again
20          uint32_t delayMS;
21  };
```

Listing 1: C++ class definition of the DHT22 sensor

3.2 RGB led

A simple common cathode RGB led.

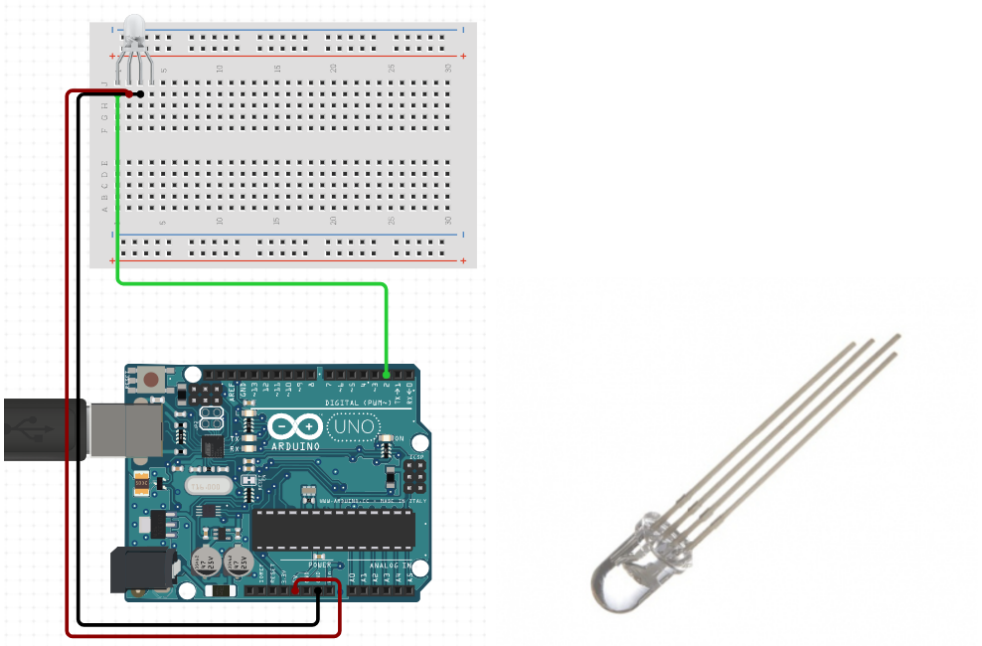

Figure 2: The RGB led Arduino diagram and a close-up of the component

```cpp
#include "Types.hpp"
#include "Arduino.h"

class LED
{
    public:
        explicit LED(Color newColor) : color(newColor) {};
        void setupLed() const;
        int getRedPin() const { return redPin; };
        int getGreenPin() const { return greenPin; };
        int getBluePin() const { return bluePin; };
        Color getColor() const { return color; };
        void setColor(Color newColor) { color = newColor; };
        void displayColor() const;
    private:
        const int redPin{11};   // PWM pin
        const int greenPin{10}; // PWM pin
        const int bluePin{9};   // PWM pin
        Color color;
};
```

Listing 2: C++ class definition of the RGB LED module

3.3 8-LED module

This is a fairly basic printed circuit board that houses 8 LED-s in common anode configuration.

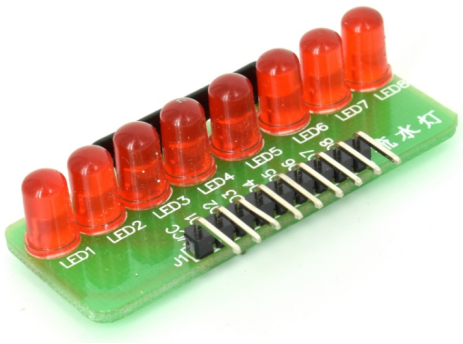

Figure 3: The led module printed circuit board

```
#include "Arduino.h"

class LedModule
{
    public:
        LedModule() {};
        void setupLedModule() const;
        int getLedPin(int index) const;
        void turnOnLed(int index) const;
        void turnOffLed(int index) const;
    private:
        // the 8 pins of the Arduino board that are connected
        // to the 8 individual cathodes of the led module
        const int leds[8]{1, 2, 3, 4, 5, 6, 7, 8};
        const int numberOfLeds{8};
};
```

Listing 3: C++ class definition of the LED module

3.4 Buzzer

Another basic component that emits sounds with respect to a channel that supports PWM.

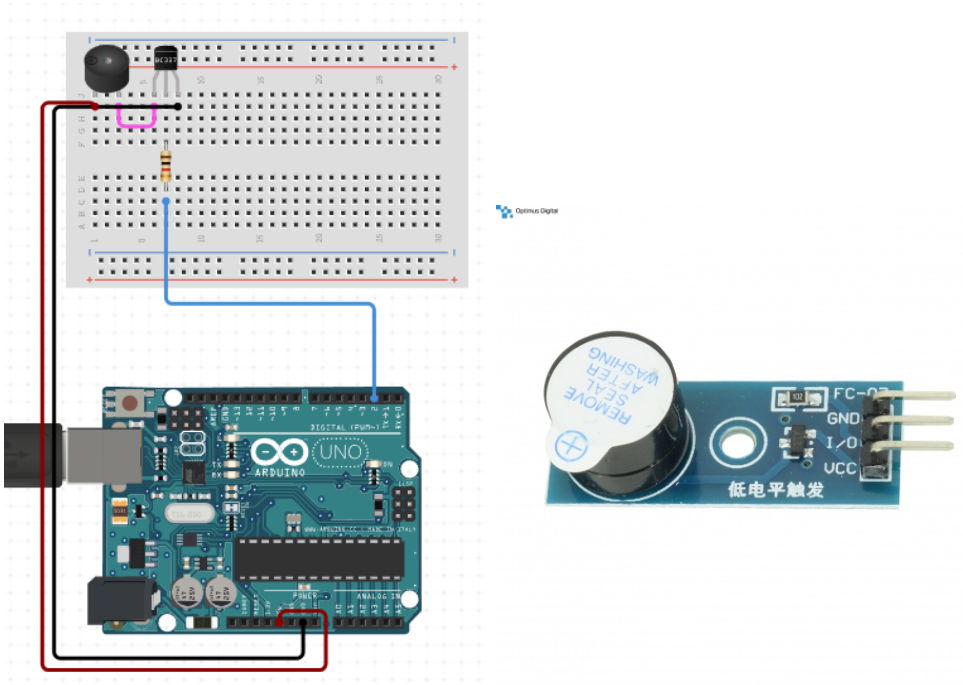

Figure 4: The diagram connection of a simple buzzer and a close-up of the component[1]

```
#include "Arduino.h"
#include "Pitches.hpp"
#include "Types.hpp"

class Buzzer
{
    public:
        Buzzer() {};
        void setupBuzzer() const;
        void turnOn(int noteFrequency, Notes duration);
        void turnOff();
        void buzzerLoop();
    private:
        // PWM Arduino Pin that is connected to the data pin
        // of the buzzer
        const int dataPin{3};
        const float pauseStretch{1.3f};
};
```

3.5 Fan

A 5V fan that is connected to a PWM capable pin and GROUND.

[1] The PCB I use has an integrated transistor and resistance meaning that interfacing with the board is much simpler: 2 power connections (GND and VCC) and a data wire.

Figure 5: The diagram connection of a simple buzzer and a close-up of the component

```
#include <stdint.h>
#include "Arduino.h"

class Fan {
    public:
        void setupFan() const;
        void powerOn(uint8_t howMuch);
    private:
        const int powerPin{ 5 };
};
```

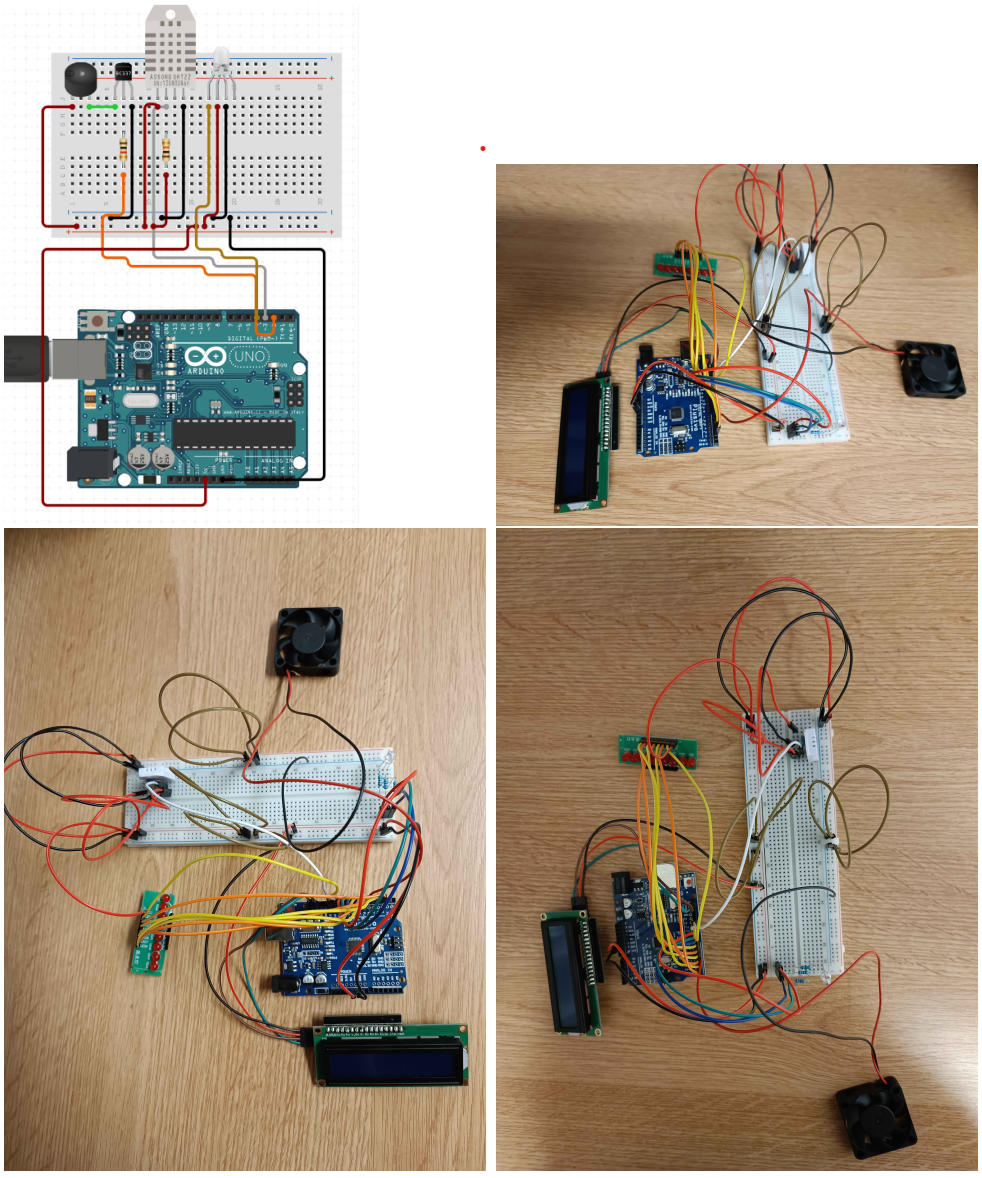

Figure 6: Arduino diagram for different phases of the project

4 Testing and Validation

Problems Encountered: During the implementation of this project, several challenges were faced:

- **Sensor Calibration**: Ensuring accurate readings from the temperature and humidity sensors required careful calibration and troubleshooting.
- **Power Management**: Managing power distribution to the various output devices (LEDs, fan, LCD) to avoid overloading the Arduino board.
- **Signal Interference**: Minimizing electrical noise and interference that affected the reliability of sensor readings and output signals.
- **Component Integration**: Ensuring compatibility and seamless communication between different components in the circuit.

Behavior Comparison: Below are tables comparing the behavior of the project when environmental conditions were altered.

Temperature (°C)	LED Status	Fan Level	LCD Message
Below 5	BLUE	Off	"Brrr!"
5 - 20	YELLOW	Off	"Spring-like!"
20 - 28	GREEN	Level 1	"Just right!"
28 - 32	RED	Level 2	"It's too hot!"
Above 32	MAGENTA	Level 3	"Overheat!"

Table 2: Temperature-Dependent Behavior

Relative Humidity (%)	LEDs Lit
Below 5	1
5 - 14.3	2
14.3 - 28.6	3
28.6 - 42.9	4
42.9 - 57.2	5
57.2 - 71.5	6
71.5 - 85.8	7
Above 85.8	8

Table 3: Humidity-Dependent LED Behavior

5 Conclusion

Purpose Fulfillment: This project successfully met its objective of providing a comprehensive feedback circuit that uses temperature and humidity sensors to trigger various output devices. The integration of visual indicators (LEDs and LCD screens), a fan with multiple power levels, and optional buzzers demonstrated a practical application of sensor data in managing environmental conditions. The approach of allowing creativity to guide component selection and circuit design resulted in a unique and functional solution.

Potential Benefits for Others: The solution can be beneficial for various applications, such as greenhouse management, home automation, and climate control systems. By providing real-time feedback on temperature and humidity, users can make informed decisions to maintain optimal conditions. This project can serve as a learning tool for students and hobbyists interested in Arduino programming and electronics.

Practical Improvements:

- *Enhanced Sensor Accuracy*: Integrate more precise sensors to improve the reliability of the data collected.
- *Wireless Communication*: Add modules like Wi-Fi or Bluetooth to enable remote monitoring and control.
- *Energy Efficiency*: Implement energy-saving measures, such as low-power modes or solar-powered components, to enhance the sustainability of the project.
- *Expanded Functionality*: Include additional sensors for other environmental parameters (e.g., light, CO2 levels) to create a more comprehensive monitoring system.
- *User Interface*: Develop a user-friendly interface, possibly with a mobile app, to display data and control the system remotely.

References

[1] aleixj. Automated greenhouse with arduino. `https://projecthub.arduino.cc/aleixj/automated-greenhouse-with-arduino-974338`, December 2020. Accessed: 2024-11-20.

[2] ejshea. Displaying temperature and humidity on an lcd. `https://projecthub.arduino.cc/ejshea/displaying-temperature-and-humidity-on-an-lcd-91bc36`, July 2019. Accessed: 2024-11-20.

[3] Sgarro and christian26. Greenhouse. `https://projecthub.arduino.cc/Sgarro/greenhouse-f6fc53`, December 2019. Accessed: 2024-11-20.