# DOCUMENTATATION

## DMP PROJECT: WEATHER STATION WITH FIRE ALAM

STUDENT NAME: PELLE ANDREI
GROUP: 30434

# TABLE OF CONTENTS

# 1. Introduction

The goal of this project is to design, implement and test a weather station with a fire alarm. As an improvement to the original design, I added wireless connectivity with a Bluetooth module and designed an app that shows the weather data from the sensors in real time.

The user of the weather station should be able to use a **remote control** to:

- See the temperature on the LCD screen
- See the humidity on the LCD screen
- See the light intensity on the LCD screen
- See the time on the LCD screen
- See the IR sensor value and if fire is present on the LCD screen
- When a flame is present, the fire alarm (buzzer and red LED) should be activated in order to warn the user of the danger
- See all of the above in a desktop or mobile app

Such a project requires a lot of care because many connections are necessary and one bad connection will make debugging  very harder. Also, one has to decide on the code structure and how to tie it all together as the different sensors require different libraries, headers and information is decoded and interpreted in different ways.

# 2. Components

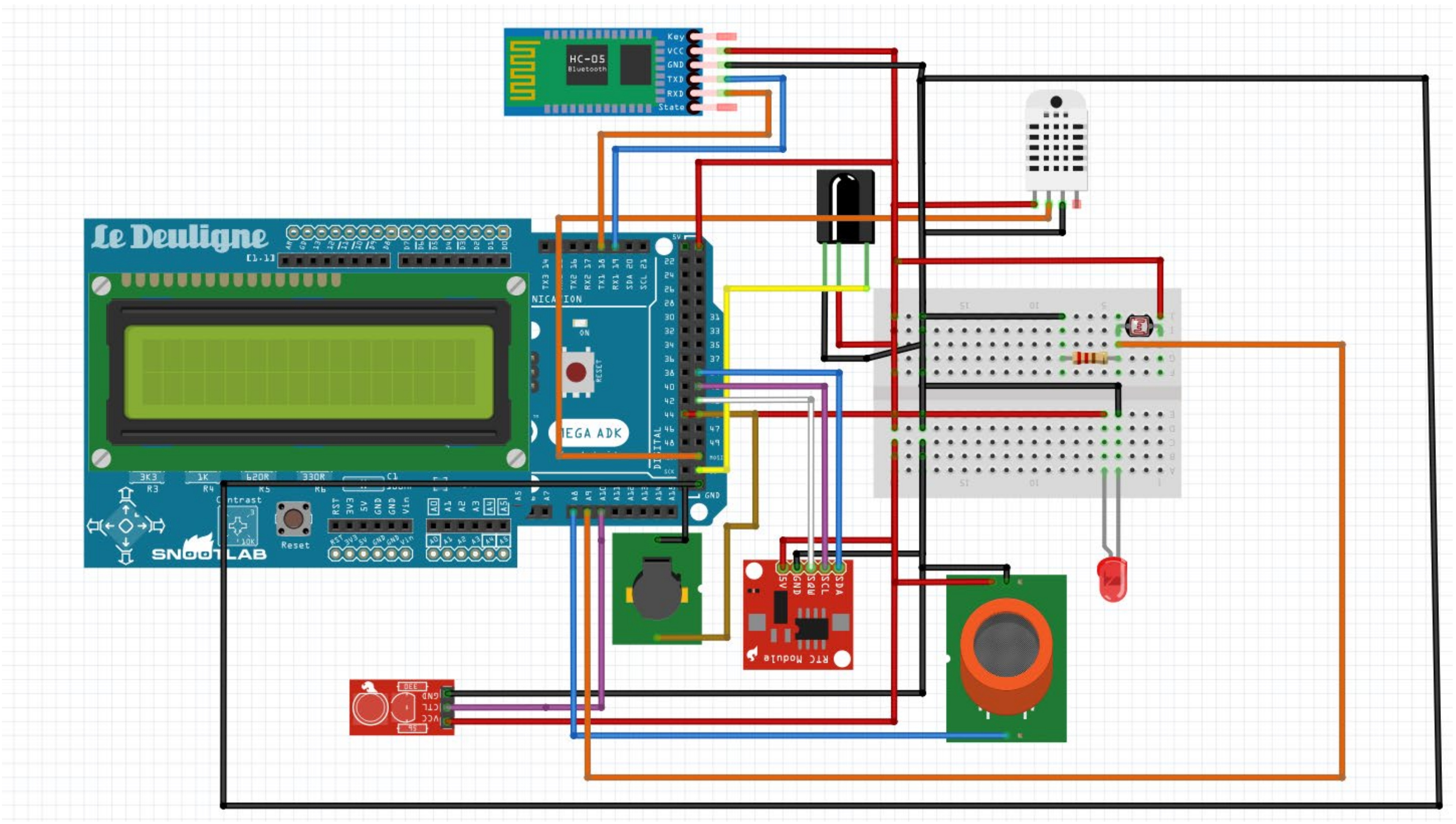
The following components were used:

- Arduino Mega R3 compatible development board – I chose this development because it has many pins and allows me to use an LCD shield and still have enough room for sensors.
- LCD 1602 module – add-on module, no need for wires
- HC-05 Bluetooth module – Bluetooth module that we learned how to use in the lab, I chose it for the familiarity
- Flame sensor module – for fire detection
- Photoresistor
- Real time clock module RTC DS1302 – I chose this module for its simplicity, no need to use the serial interface.
- MQ-135 air quality sensor – used for measuring air quality

- DHT11 – used for measuring the temperature and the humidity
- IR receiver – used to receive commands from the remote control
- Remote
- Red LED
- Buzzer

## 3. Schematic

One can see the schematic for the project below. It is not quite a 1:1 because some parts were not available in the Fritzing library, but it gives a very good overview of the implementation.



The left part of the board is occupied by the LCD shield. Pins A8, A9 and A10 were used to read the analog data from the gas, light and infrared (fire) sensors and then translated in the code to useful data. Digital pin 53 is used to receive data from the IR receiver, digital pin 51 is used to receive data from the DHT sensor. Pins 44 and 45 (both PWM) are used to control the buzzer and the LED. Pins 43, 41 and 39 are used for exchanging data with the real time clock module. The Bluetooth module is connected to the RX1 and TX1 pins  of the Arduino mega board, thus serial interface 1 was used to interface the Bluetooth module.

# 4. Implementation

For the implementation, the program can be thought of a state machine, with states 1 thru 5 showing different data from the sensors to the LCD. Additionally, data is refreshed once every 3 seconds using a timer interrupt. When the data is refreshed, information is also gathered and sent to the Bluetooth module for processing.

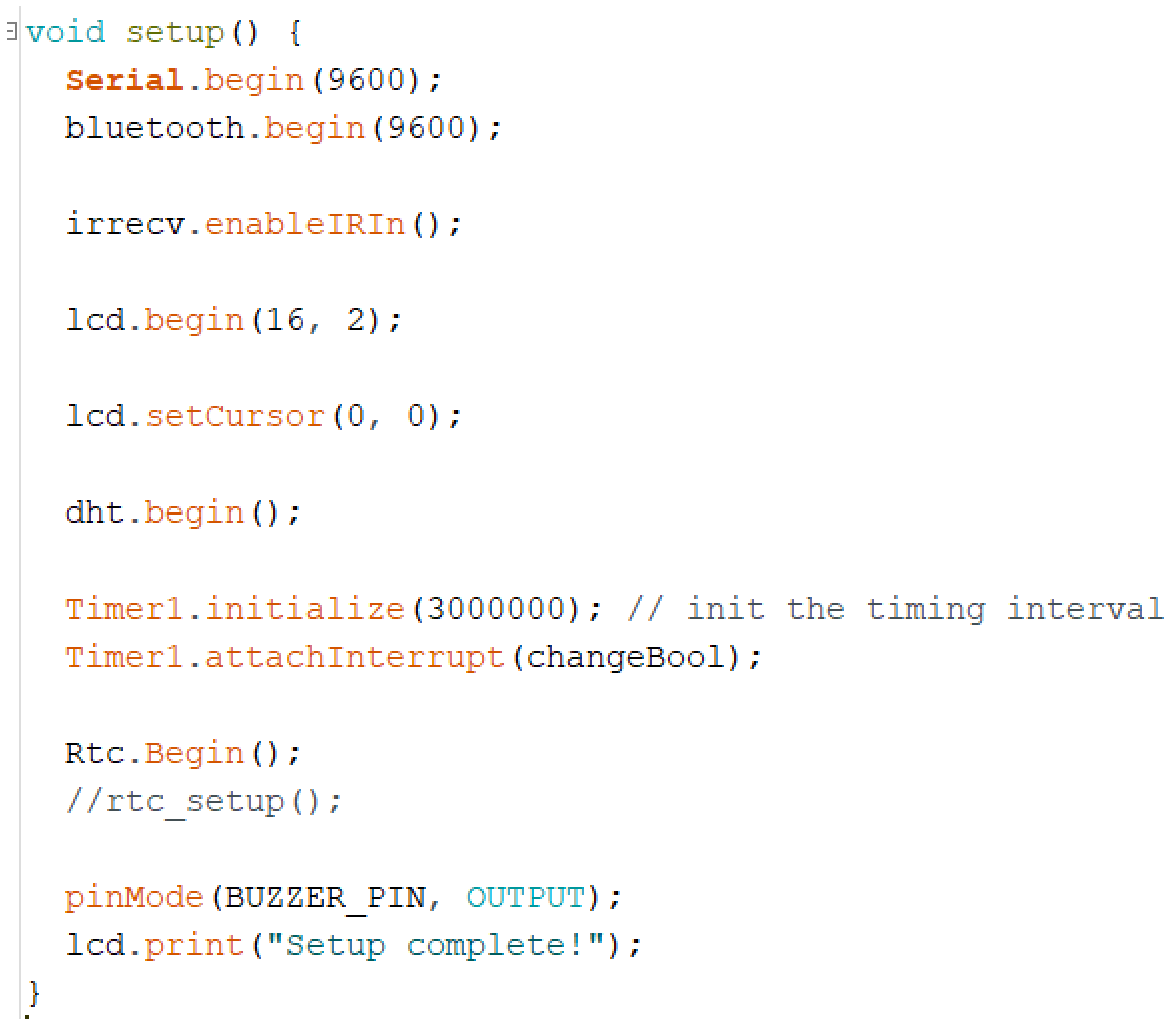
At any point in time, if the flame sensor senses a flame, the fire alarm will start, and the LCD will show an appropriate message. No other info can be seen on the screen at this point, the fire alarm having the highest priority and no state assigned, a special exception if you will.

For the 5 states, one can see the following on the LCD:

1. Temperature and Humidity
2. Date and Time
3. Air Quality
4. Light Intensity
5. Flame sensor readings

The code starts with some includes and defines for better readability. The following libraries and includes were used: “DHT.h”, “IRRemote.h”, “TimerOne.h”, “ThreeWire.h”, “RtcDS1302.h”, “SoftwareSerial.h”, and a global variable for the current state.

In the setup, the sensors and pins are initialized, also the timer is initialized to 3 seconds by passing 3 000 000 to the initialize() function.



```
void setup() {
  Serial.begin(9600);
  bluetooth.begin(9600);

  irrecv.enableIRIn();

  lcd.begin(16, 2);

  lcd.setCursor(0, 0);

  dht.begin();

  Timer1.initialize(3000000); // init the timing interval
  Timer1.attachInterrupt(changeBool);

  Rtc.Begin();
  //rtc_setup();

  pinMode(BUZZER_PIN, OUTPUT);
  lcd.print("Setup complete!");
}
```

The loop() starts with 2 functions, checkFire() and checkIRreceiver().

The checkFire() functions checks the value of the IR sensors and if there is a fire (distance read from sensor under a small value), the lcd is cleared, the pins of the LED and BUZZER are driven to high(buzzer) and a PWM value for LED. A 3 second delay is called and then the value of the sensor is checked again.

checkIRreceiver() will use the built-in functions from the library to decode the data from the remote.  state_changed is set to 1 so the LCD data will be refreshed, and the correct state will be set in the CURRENT_STATE variable.

```
if (irrecv.decode())// Returns 0 if no data ready, 1 if data ready.
{
  long long incoming = irrecv.decodedIRData.decodedRawData;
  if(incoming == FALSE_PRESS) goto CONT;
  lcd.clear();
  lcd.setCursor(0, 0);
  if(incoming == PRESSED_ONE) {
    CURRENT_STATE = 1;
  }
  else if(incoming == PRESSED_TWO) {
    CURRENT_STATE = 2;
  }
  else if(incoming == PRESSED_THREE) {
    CURRENT_STATE = 3;
  }
  else if(incoming == PRESSED_FOUR) {
    CURRENT_STATE = 4;
  }
  else if(incoming == PRESSED_FIVE) {
    CURRENT_STATE = 5;
  }
  else {
    Serial.println("Unknown code from remote:");
    Serial.println(irrecv.decodedIRData.decodedRawData, DEC);
  }
  state_changed = true;
  CONT:
  irrecv.resume(); // Restart the ISR state machine and Receive the next value
```

Further in the loop, we check if the state has changed, then update data from the DHT and clear the lcd. Depending on the state we are in, we print the temp and humidity (in state 1) with a utility function, print the date and time with a utility function (in state 2), check the gas sensor in state 3 and depending on the value we print the quality of the air.

```
 if(state_changed == true){
   state_changed = false;
   checkTempAndHumidity();

   lcd.clear();
   lcd.setCursor(0, 0);
   if(CURRENT_STATE == 1) printTempAndHumidity();
   if(CURRENT_STATE == 2)  {
     RtcDateTime now = Rtc.GetDateTime();
     printDateTime(now);
   }
   if(CURRENT_STATE == 3){
     //lcd.print("Pressed 3");
     int sensorValue = analogRead(GAS_PIN);
     Serial.print("Read from gas pin:");
     Serial.println(sensorValue);
     lcd.print("Gas value:");
     lcd.print(sensorValue);
     lcd.setCursor(0,1);
     if(sensorValue < 200)lcd.print("Good quality!");
     else if(sensorValue < 500) lcd.print("Medium quality!");
     else lcd.print("DANGER!Bad air!");
   }
```

In states 4 and 5, we do something similar to states 2 and 3. We check the sensor values and print the corresponding message.

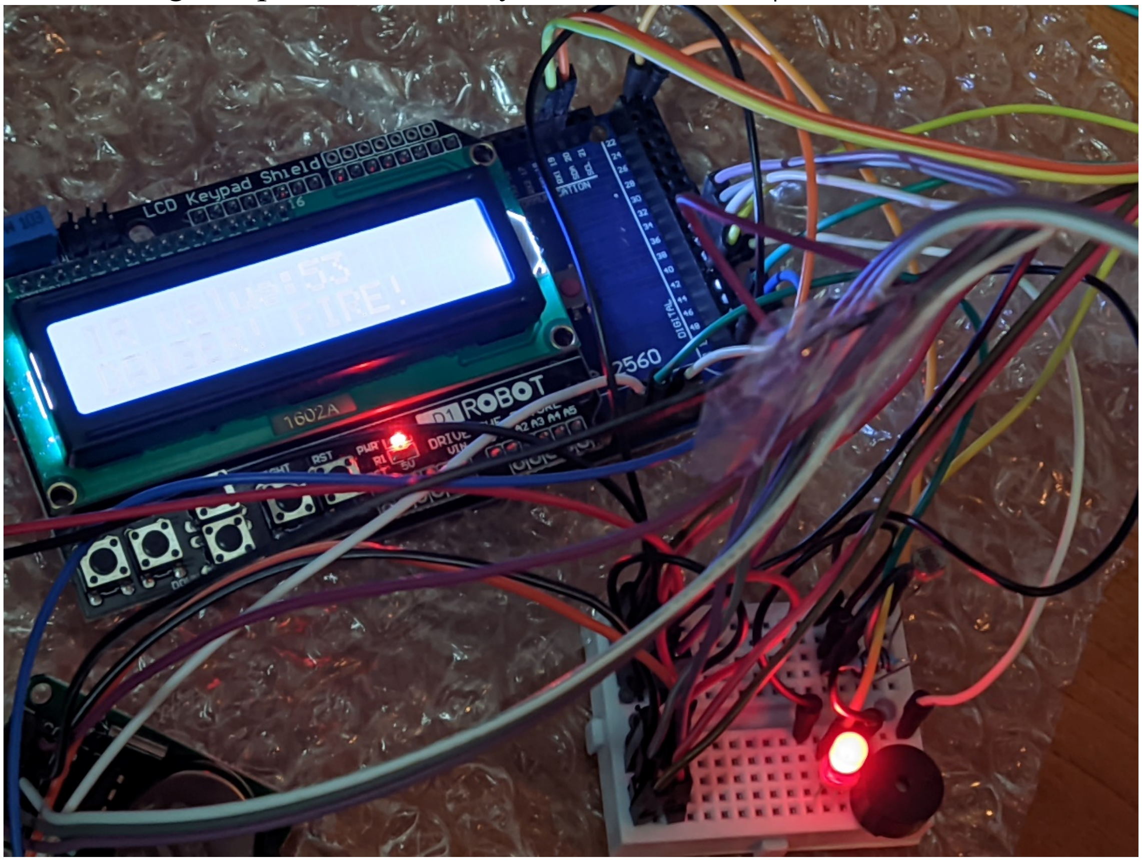
```
 if(CURRENT_STATE == 4){
   int sensorValue = analogRead(LIGHT_PIN);
   Serial.println(sensorValue);

   lcd.print("Light value:");
   lcd.print(sensorValue);
   lcd.setCursor(0,1);
   if(sensorValue < 50) lcd.print("Pitch dark");
   else if(sensorValue <150) lcd.print("Dark");
   else if(sensorValue < 300) lcd.print("Dim");
   else if(sensorValue < 700) lcd.print("Normal light");
   else if (sensorValue <900) lcd.print("Bright");
   else lcd.print("Blinding light");
 }
 if(CURRENT_STATE == 5){
   int sensorValue = analogRead(FIRE_PIN);
   Serial.println(sensorValue);
   lcd.print("IR value:");
   lcd.print(sensorValue);
   lcd.setCursor(0,1);
   if(sensorValue > 500) lcd.print("No fire!");
 }
```

The changeBool() interrupt will set state_changed to true and send the data formatted to the Bluetooth module.

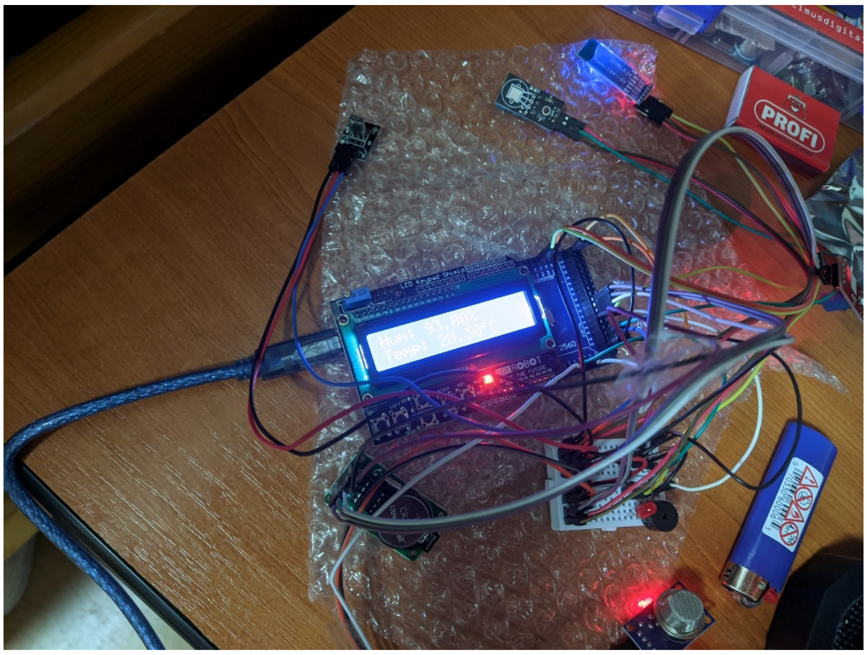
```
  snprintf_P(messageToSend,
             countof(messageToSend),
PSTR("Humidity=%s%% \nTemperature=%s°C \nDate: %02u/%02u/%04u \nTime:
%02u:%02u:%02u \nGas sensor value: %d -> %s\nLight sensor value: %d ->
%s\nFire sensor value: %d -> %s\n\n"),
             hum_temp,
             str_temp,
             dt.Day(),
             dt.Month(),
             dt.Year(),
             dt.Hour(),
             dt.Minute(),
             dt.Second(),
             gasSensorValue,
             gasString,
             lightSensorValue,
             lightString,
             fireSensorValue,
             fireString);

  bluetooth.print(messageToSend);
```

## 5. Pictures and Testing

In the picture below, one can see the state where a fire has been detected and the LED lights up. The screen says : “IR value: 53 | DANGER! FIRE!”



In the next picture we can see the temperature and humidity displayed (when ‘1’ is sent from the remote)



In this final picture we can see what the app looks like and what reading one can get from the sensors.

The humidity is expressed as a percentage with 2 decimal points.

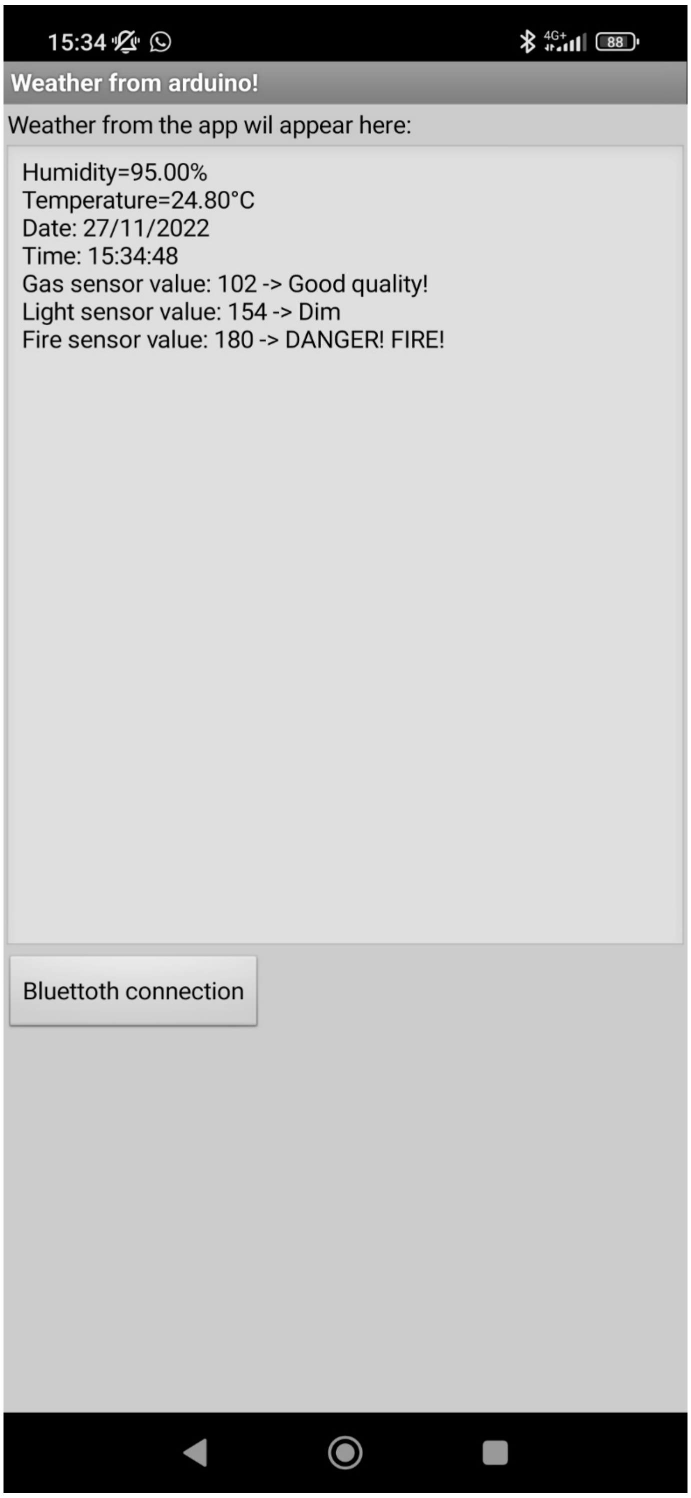
The temperature as a fixed-point with 2 decimal points and the degree symbol.

The date is given in the DD/MM/YYYY format, but this can be changed to any other format desired. The time is given in the 24-hour time format.

The air quality is given in 2 ways, the direct reading from the sensor and an interpretation based on the value read: good, medium, or bad quality air.

The light sensor is given in a similar manner, with more categories such as dim, normal, bright, etc.

The fire sensor value is not really relevant, but the interpretation is. Any value other than a really high one means that there is a fire close to the weather station, and the appropriate message shall be printed.



Weather from the app wil appear here:

```
Humidity=95.00%
Temperature=24.80°C
Date: 27/11/2022
Time: 15:34:48
Gas sensor value: 102 -> Good quality!
Light sensor value: 154 -> Dim
Fire sensor value: 180 -> DANGER! FIRE!
```

## 6. Conclusions and Further Improvements

This project has taught me a lot about how microcontrollers work and how one would implement something useful using a microcontroller and a few sensors. Given the time limit, the budget and the complexity involved, I am quite proud of the result, and I think it has helped me achieve a better understanding of this subject.

Further improvements can be made to the weather station, some of them being:

- Use a better DHT sensor since the DHT11 is old and does not support many reads
- Use a better RTC module, perhaps an I2C/SPI variant to use fewer pins
- Use a development board to solder the modules
- Add more sensors like pressure or wind sensors
- Remake the app in Kotlin and give it a better look and make a similar desktop application.
- Bluetooth connects to a server where it sends data and all the other users (mobile or desktop apps) connect to this API endpoint to get the weather data

## 7. Bibliography

- https://appinventor.mit.edu/ -> used for designing the mobile application
- https://create.arduino.cc/projecthub -> various ideas for the design and implementation of the project
- https://fritzing.org/ -> tool used to design the schematic for the project
- https://users.utcluj.ro/~rdanescu/teaching_pmp.html -> laboratory and lecture notes useful for implementation, especially the use of the TimerOne library