ESP32/BM680

Blink

Código de prueba para encender y apagar un LED conectado a la placa ESP32.

`Blink/Blink.ino`

```
int LED_BUILTIN = 2; // Set LED_BUILTIN pin
void setup() { // Set LED_BUILTIN pin as output
    pinMode (LED_BUILTIN, OUTPUT);
}
void loop() { // Blink LED_BUILTIN
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

BME680

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32.

Las librerías usadas son y se encuentran disponibles en:

- *Adafruit_BME680* (https://github.com/adafruit/Adafruit_BME680)
- *Adafruit_Sensor* (https://github.com/adafruit/Adafruit_Sensor)

Wire y SPI son librerías de Arduino para comunicación I2C y SPI respectivamente.

`BME680/BME680.ino`

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"


#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C

void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.println(F("BME680 async test"));

  if (!bme.begin()) {
    Serial.println(F("Could not find a valid BME680 sensor, check wiring!"));
    while (1);
  }

  // Set up oversampling and filter initialization
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
}
```

```
void loop() {
  // Tell BME680 to begin measurement.
  unsigned long endTime = bme.beginReading();
  if (endTime == 0) {
    Serial.println(F("Failed to begin reading :("));
    return;
  }
  Serial.print(F("Reading started at "));
  Serial.print(millis());
  Serial.print(F(" and will finish at "));
  Serial.println(endTime);

  Serial.println(F("You can do other work during BME680 measurement."));
  delay(50); // This represents parallel work.
  // There's no need to delay() until millis() >= endTime: bme.endReading()
  // takes care of that. It's okay for parallel work to take longer than
  // BME680's measurement time.

  // Obtain measurement results from BME680. Note that this operation isn't
  // instantaneous even if milli() >= endTime due to I2C/SPI latency.
  if (!bme.endReading()) {
    Serial.println(F("Failed to complete reading :("));
    return;
  }
  Serial.print(F("Reading completed at "));
  Serial.println(millis());

  Serial.print(F("Temperature = "));
  Serial.print(bme.temperature);
  Serial.println(F(" *C"));

  Serial.print(F("Pressure = "));
  Serial.print(bme.pressure / 100.0);
  Serial.println(F(" hPa"));

  Serial.print(F("Humidity = "));
  Serial.print(bme.humidity);
  Serial.println(F(" %"));

  Serial.print(F("Gas = "));
  Serial.print(bme.gas_resistance / 1000.0);
  Serial.println(F(" KOhms"));

  Serial.print(F("Approx. Altitude = "));
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(F(" m"));

  Serial.println();
  delay(2000);
}
```

BME680 - OLED

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32 y mostrarlos en una pantalla OLED conectada a la misma placa.

Las librerías usadas son y se encuentran disponibles en:

- *Adafruit_BME680* (https://github.com/adafruit/Adafruit_BME680)
- *Adafruit_GFX* (https://github.com/adafruit/Adafruit-GFX-Library)
- *Adafruit_Sensor* (https://github.com/adafruit/Adafruit_Sensor)
- *Adafruit_SSD1306* (https://github.com/adafruit/Adafruit_SSD1306)

BME680/BME680-OLED.ino

```
/***************************************************************************
  This is a library for the BME680 gas, humidity, temperature & pressure sensor

  Designed specifically to work with the Adafruit BME680 Breakout
  ----> http://www.adafruit.com/products/3660

  These sensors use I2C or SPI to communicate, 2 or 4 pins are required
  to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ***************************************************************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO,  BME_SCK);

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

void setup() {
  Serial.begin(9600);
  Serial.println(F("BME680 test"));

  // by default, we'll generate the high voltage from the 3.3v line internally!
(neat!)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C
(for the 128x32)
  // init done
  display.display();
  delay(100);
  display.clearDisplay();
  display.display();
  display.setTextSize(1);
  display.setTextColor(WHITE);

  if (!bme.begin()) {
    Serial.println("Could not find a valid BME680 sensor, check wiring!");
```

```
    while (1);
  }

  // Set up oversampling and filter initialization
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
}

void loop() {
  display.setCursor(0,0);
  display.clearDisplay();

  if (! bme.performReading()) {
    Serial.println("Failed to perform reading :(");
    return;
  }
  Serial.print("Temperature = "); Serial.print(bme.temperature); Serial.println("
*C");
  display.print("Temperature: "); display.print(bme.temperature); display.println("
*C");

  Serial.print("Pressure = "); Serial.print(bme.pressure / 100.0); Serial.println("
hPa");
  display.print("Pressure: "); display.print(bme.pressure / 100); display.println("
hPa");

  Serial.print("Humidity = "); Serial.print(bme.humidity); Serial.println(" %");
  display.print("Humidity: "); display.print(bme.humidity); display.println(" %");

  Serial.print("Gas = "); Serial.print(bme.gas_resistance / 1000.0); Serial.println("
KOhms");
  display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);
display.println(" KOhms");

  Serial.println();
  display.display();
  delay(2000);
}
```

BME680 - OLED - PocketBase

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32 y mostrarlos en una pantalla OLED conectada a la misma placa. Además, los datos son enviados a una base de datos PocketBase en un servidor remoto.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

`BME680/BME680-OLED-PocketBase.ino`

```
/**************************************************************************
    This is a library for the BME680 gas, humidity, temperature & pressure sensor

    Designed specifically to work with the Adafruit BME680 Breakout
    ----> http://www.adafruit.com/products/3660
```

```
    These sensors use I2C or SPI to communicate, 2 or 4 pins are required
    to interface.

    Adafruit invests time and resources providing this open source code,
    please support Adafruit and open-source hardware by purchasing products
    from Adafruit!

    Written by Limor Fried & Kevin Townsend for Adafruit Industries.
    BSD license, all text above must be included in any redistribution
 ***************************************************************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <esp_sleep.h>

#define LED 2

#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

String timeUTC;
bool wifiConnected = true;

const char* ssid = "Hogwarts"; // Network SSID
const char* password = "zV9%E^%tJNd!yaW*"; // Network password

const char* ntpServer = "pool.ntp.org"; // NTP server
const long gmtOffset_sec = 0; // Offset from GMT
const int daylightOffset_sec = 0; // Offset from daylight savings time

void setup() {
    Serial.begin(9600);
    Serial.println("Starting BME680...");

    Serial.println("Starting Wi-Fi..."); // Print a message to the serial monitor
    wifiConnected = connectToWifi(); // Obtain the Wi-Fi connection status

    if (wifiConnected) {
        Serial.println("Connection to Wi-Fi successful"); // Print a message to the
serial monitor
```

```
        Serial.println("Starting time sync..."); // Print a message to the serial
monitor
        configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
        while (!time(nullptr)) {
            delay(1000);
            Serial.println("Waiting for time sync...");
        }
        Serial.println("Time synced");
    } else {
        Serial.println("Connection to Wi-Fi failed"); // Print a message to the
serial monitor
        Serial.println("Proceeding without Wi-Fi..."); // Print a message to the
serial monitor
    }


    // by default, we'll generate the high voltage from the 3.3v line internally!
(neat!)
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C
(for the 128x32)
    // init done
    display.display();
    delay(100);
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);

    if (!bme.begin()) {
        Serial.println("Could not find a valid BME680 sensor, check wiring!");
        while (1); // Freeze the program
    }

    // Set up oversampling and filter initialization
    bme.setTemperatureOversampling(BME680_OS_8X);
    bme.setHumidityOversampling(BME680_OS_2X);
    bme.setPressureOversampling(BME680_OS_4X);
    bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
    bme.setGasHeater(320, 150); // 320*C for 150 ms

    pinMode(LED, OUTPUT); // Set the LED pin as an output

    Serial.println("Setup complete");
}

void loop() {
    digitalWrite(LED, HIGH); // Turn the LED on (Note that LOW is the voltage level

    display.setCursor(0,0);
    display.clearDisplay();

    if (! bme.performReading()) {
        Serial.println("Failed to perform BME680 reading");
        return;
    }
```

```
    printToSerial(); // Print to serial monitor
    printToDisplay(); // Print to OLED display

    if (!wifiConnected) { // If Wi-Fi is not connected, wait 10 minutes and try again
        Serial.println("Retrying in 10 minutes...");
        digitalWrite(LED, LOW); // Turn the LED off by making the voltage HIGH
        delay(10 * 60 * 1000); // Wait 10 minutes

        esp_restart(); // Restart the ESP32
    }

    timeUTC = getUTCTime(); // Get UTC time from NTP server

    sendToPocketBase(); // Send data to PocketBase

    Serial.println("Updating in 10 minutes...");
    digitalWrite(LED, LOW); // Turn the LED off by making the voltage HIGH

    delay(10 * 60 * 1000); // Wait 10 minutes
}

void printToSerial() { // Print to serial monitor
    Serial.print("Temperature = "); Serial.print(bme.temperature); Serial.println("
*C");
    Serial.print("Pressure = "); Serial.print(bme.pressure / (20 * 133.32239));
Serial.println(" inHg");
    Serial.print("Humidity = "); Serial.print(bme.humidity); Serial.println(" %");
    Serial.print("Gas = "); Serial.print(bme.gas_resistance / 1000.0);
Serial.println(" KOhms");
}

void printToDisplay() { // Print to OLED display
    display.setCursor(0,0);
    display.clearDisplay();
    display.print("Temperature: "); display.print(bme.temperature); display.println("
*C");
    display.print("Pressure: "); display.print(bme.pressure / (20 * 133.32239));
display.println(" inHg");
    display.print("Humidity: "); display.print(bme.humidity); display.println(" %");
    display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);
display.println(" KOhms");
    display.display();
}

bool connectToWifi() { // Connect to the Wi-Fi network
    Serial.print("Connecting to ");
    Serial.println(ssid);

    unsigned long startTime = millis(); // Get the current time

    WiFi.begin(ssid, password); // Connect to the network

    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
        delay(500);
        Serial.print(".");
        if (millis() - startTime > 60000) { // If it's been more than 1 minute
```

```
            Serial.println("");
            Serial.println("WiFi connection timed out");
            return false; // Return false
        }
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP()); // Print the local IP address

    return true; // Return true if connection was successful
}

String getUTCTime() { // Get UTC time from NTP server
    struct timeval tv;
    gettimeofday(&tv, nullptr);
    time_t now = tv.tv_sec;
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    char buffer[30];
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.%03ldZ",
        timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
        timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
        tv.tv_usec / 1000);
    return String(buffer);
}

void sendToPocketBase() { // Send data to PocketBase
    Serial.println("Sending data to PocketBase...");

    HTTPClient http;

    http.begin("https://w.arias.pw/api/collections/bme680/records"); // Specify the
URL
    http.addHeader("Content-Type", "application/json"); // Specify content-type
header

    // Create the JSON payload
    String payload = "{\"time\": \"" + timeUTC + "\", \"temperature\": \"" +
bme.temperature + "\", \"pressure\": \"" + bme.pressure / (20 * 133.32239) + "\",
\"humidity\": \"" + bme.humidity + "\", \"gas\": \"" + bme.gas_resistance / 1000.0 +
"\"}";

    int httpCode = http.POST(payload); // Send the request

    if(httpCode == 200) { // Check the returning code
        Serial.println("Data sent to PocketBase successfully");
    } else { // If the code is not 200, something went wrong
        Serial.print("Error sending data to PocketBase, returned code: ");
        Serial.println(httpCode);

        Serial.println("Restarting ESP32...");
        delay(1000); // Wait for the serial output to finish
        esp_restart(); // Restart the ESP32
    }
```

```
    http.end(); // Close connection
}
```

Estación Meteorológica

Blink

Código de prueba para encender y apagar el LED integrado en la placa.

Blink/Blink.ino

```
// Blink a LED on the MicroMod Weather (ESP32) board

int ledPin = 2; // LED is connected to GPIO2

void setup() {
  pinMode(ledPin, OUTPUT); // Set GPIO2 to output mode
  Serial.begin(115200); // Initialize serial port
}

void loop() {
  digitalWrite(ledPin, HIGH); // Turn LED on
  delay(1000); // Wait for 1000 millisecond(s)
  Serial.println("The LED is on."); // Print a message
  digitalWrite(ledPin, LOW); // Turn LED off
  delay(1000); // Wait for 1000 millisecond(s)
}
```

BME280

Código para leer los datos del sensor BME280.

BME280/BME280.ino

```
#include <Wire.h>
#include "SparkFunBME280.h"

BME280 bme280Sensor; // Create BME280 object

float RealFloatPressure;

void setup() {
    Serial.begin(115200); // Initialize serial port
    while (!Serial);  // Wait for user to open serial monitor

    Serial.println("MicroMod Weather Carrier Board - BME280 Example");
    Serial.println();

    Wire.begin(); // Join I2C bus

    bme280Sensor.setReferencePressure(101500); // Set sea level pressure to 101325 Pa
(default)

    if (bme280Sensor.begin() == false) { // Connect to BME280
        Serial.println("BME280 did not respond.");
        while(1); // Freeze
    }

    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
```

```
  Serial.print("Temperature: ");
  Serial.println(bme280Sensor.readTempC(), 2);
  Serial.print("Humidity: ");
  Serial.println(bme280Sensor.readFloatHumidity(), 0);
  Serial.print("Pressure: ");

  RealFloatPressure = bme280Sensor.readFloatPressure() / (20 * 133.32239);
  Serial.println(RealFloatPressure, 2);

  Serial.print("Altitude: ");
  Serial.println(bme280Sensor.readFloatAltitudeMeters(), 1);
  Serial.print("Dewpoint: ");
  Serial.println(bme280Sensor.dewPointC(), 2);

  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

VEML6075

Código para leer los datos del sensor VEML6075.

VEML6075/VEML6075.ino

```
#include <SparkFun_VEML6075_Arduino_Library.h>

VEML6075 veml6075; // Create a VEML6075 object

void setup() {
  Serial.begin(115200);
  while(!Serial); // Wait for user to open serial monitor

  Serial.println("MicroMod Weather Carrier Board - VEML6075 Example");

  Wire.begin(); // Join I2C bus

  if (veml6075.begin() == false) {
    Serial.println("VEML6075 did not respond."); // If the sensor does not respond,
print an error message
    while(1); // Freeze
  }

  pinMode(LED_BUILTIN, OUTPUT);
  Serial.println("UVA, UVB, UV Index"); // Print the header for the data
  Serial.println();

}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    // Print the UVA, UVB, and UV Index values
    Serial.println("UVA: " + String(veml6075.uva()));
    Serial.println("UVB: " + String(veml6075.uvb()));
    Serial.println("UV Index: " + String(veml6075.index()));

    digitalWrite(LED_BUILTIN, LOW);
    delay(1000); // Wait 1 second
}
```

Station

Código para leer los datos de los sensores en conjunto.

`Station/Station.ino`

```
// Station code for the SparkFun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start
// It should print out the weather data to the serial monitor each duration minutes

#include <Wire.h>
#include "SparkFunBME280.h"
#include <SparkFun_VEML6075_Arduino_Library.h>

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

const int duration = 1; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the remaining minutes
and seconds

volatile int windSpeedCount = 0; // Variable to store the number of wind pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y values
float x, y, theta, averageWindDirection, averageWindSpeed; // Variables to store the
x, y, theta, and average wind direction and speed

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
    float uvindex;
};
WeatherData weather;

struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};
WindData wind[duration * 6];


void setup() {
    Serial.begin(115200); // Start the serial monitor
```

```
    while (!Serial); // Wait for user to open serial monitor

    Wire.begin(); // Join the I2C bus

    if (bme280.begin() == false) { // Connect to the BME280
        Serial.println("BME280 did not respond."); // Print an error message if the
BME280 does not respond
        while(1); // Freeze
    }

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond.");
        while(1);
    }

    pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
    pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as an input
    pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
    attachInterrupt(digitalPinToInterrupt(windSpeedSensor), windSpeedIRQ,
FALLING); // Attach the wind speed interrupt
    attachInterrupt(digitalPinToInterrupt(rainSensor), rainIRQ, FALLING); // Attach
the rain interrupt
    interrupts(); // Enable interrupts

    Serial.println("Station complete"); // Print a message to the serial monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED
    rainCount = 0; // Reset the rain count
    for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction array
            wind[i].reading = false;
            wind[i].direction = 0;
            wind[i].speed = 0;
    }
    for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of the
report, taking measurements every 10 seconds
        Serial.print("Taking wind measurements, this process will be finished in:
"); // Print a message to the serial monitor to let the user know the program is
working
        remainingMinutes = duration - ((i / 6) + !!(i % 6));
        remainingSeconds = 60 - ((i % 6) * 10);
        Serial.print(remainingMinutes);
        Serial.print(":");
        if (remainingSeconds == 60) {
            Serial.println("00");
        } else {
            Serial.println(remainingSeconds);
        }

        windSpeedCount = 0; // Reset the wind speed count
        delay(10 * 1000); // Wait for 10 seconds
        wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the wind speed
        wind[i].direction = getWindDirection(); // Calculate the wind direction
        wind[i].reading = true; // Set the reading flag to true
    }
```

```
    weather.rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate the rain
fall in inches/hour
    getAverageWind(averageWindDirection, averageWindSpeed); // Get the average wind
direction and speed
    weather.windDirection = averageWindDirection; // Set the average wind direction
to the weather data
    weather.windSpeed = averageWindSpeed; // Set the average wind speed to the
weather data
    weather.temperature = bme280.readTempC(); // Get the temperature in degrees
Celsius
    weather.humidity = bme280.readFloatHumidity(); // Get the humidity in percent
    weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees Celsius
    weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); // Convert the
pressure from Pascals to inches of mercury

    weather.uva = veml6075.uva(); // Get the UVA value
    weather.uvb = veml6075.uvb(); // Get the current time in UTC
    weather.uvindex = veml6075.index(); // Get the UV index

    printWeather(); // Print the weather data to the serial monitor

    digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}

void printWeather() {
    Serial.print("Temperature: ");
    Serial.print(weather.temperature, 2); // Temperature in degrees Celsius
    Serial.print("   Humidity: ");
    Serial.print(weather.humidity, 2); // Humidity in percent
    Serial.print("   Dewpoint: ");
    Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
    Serial.print("   Pressure: ");
    Serial.print(weather.pressure, 2); // Pressure in inches of mercury
    Serial.print("   Wind Speed: ");
    Serial.print(weather.windSpeed, 2); // Wind speed in knots
    Serial.print("   Wind Direction: ");
    Serial.print(weather.windDirection, 2); // Average wind direction in degrees
    Serial.print("   Rain Fall: ");
    Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
    Serial.print("   UVA: ");
    Serial.print(weather.uva); // UVA value
    Serial.print("   UVB: ");
    Serial.print(weather.uvb); // UVB value
    Serial.print("   UV Index: ");
    Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value from the wind
direction sensor

    if (  10 < reading && reading <=  150) return (288);
    if ( 150 < reading && reading <=  250) return (264);
    if ( 250 < reading && reading <=  400) return (336);
    if ( 400 < reading && reading <=  600) return (312);
```

```
    if ( 650 < reading && reading <=  850) return ( 24);
    if ( 850 < reading && reading <= 1050) return (  0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}

void getAverageWind(float& averageWindDirection, float& averageWindSpeed) {
    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // convert angle to radians
        x = wind[i].speed * cos(theta);
        y = wind[i].speed * sin(theta);
        xSum += x;
        ySum += y;
    }
    averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum / (duration *
6))); // convert radians to degrees
    if (averageWindDirection < 0) averageWindDirection += 360; // convert negative
angles to positive
    averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum / (duration *
6), 2)); // calculate average speed

    xSum = 0;
    ySum = 0;
}

void rainIRQ()
{
    rainCount++;
    // Serial.println("Rain clicked");
}

// Function is called when the magnet in the anemometer is activated
void windSpeedIRQ()
{
    windSpeedCount++;
    // Serial.println("Wind clicked");
}
```

Station - PocketBase

Código para leer los datos de los sensores en conjunto y enviar los datos a la base de datos de PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

Station_PocketBase/Station_PocketBase.ino

```
// Station code for the Sparkfun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start
// It should print out the weather data to the serial monitor each duration minutes
// It should also send the weather data to a PocketBase server every 5 minutes

#include <Wire.h>
#include "SparkFunBME280.h"
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <SparkFun_VEML6075_Arduino_Library.h>

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

const int duration = 4; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the remaining minutes
and seconds

String currentTime; // Variable to store the current time
String minuteString; // Variable to store the minutes from the time
int minuteInt; // Variable to store the minutes as an int

volatile int windSpeedCount = 0; // Variable to store the number of wind pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y values
float x, y, theta, averageWindDirection, averageWindSpeed; // Variables to store the
x, y, theta, and average wind direction and speed

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
    float uvindex;
};
WeatherData weather;

struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};
```

```
WindData wind[duration * 6];

const char* ssid = "Hogwarts"; // Network SSID
const char* password = "zV9%E^%tJNd!yaW*"; // Network password

const char* ntpServer = "pool.ntp.org"; // NTP server
const long gmtOffset_sec = 0; // Offset from GMT
const int daylightOffset_sec = 0; // Offset from daylight savings time

void setup() {
    Serial.begin(115200); // Start the serial monitor
    while (!Serial); // Wait for user to open serial monitor

    connectToWifi(); // Connect to the Wi-Fi network

    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    while (!time(nullptr)) {
        delay(1000);
        Serial.println("Waiting for time sync...");
    }
    Serial.println("Time synced");

    Wire.begin(); // Join the I2C bus

    if (bme280.begin() == false) { // Connect to the BME280
        Serial.println("BME280 did not respond."); // Print an error message if the
BME280 does not respond
        while(1); // Freeze
    }

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond.");
        while(1); // Freeze
    }

    pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
    pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as an input
    pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
    attachInterrupt(digitalPinToInterrupt(windSpeedSensor), windSpeedIRQ,
FALLING); // Attach the wind speed interrupt
    attachInterrupt(digitalPinToInterrupt(rainSensor), rainIRQ, FALLING); // Attach
the rain interrupt
    interrupts(); // Enable interrupts

    Serial.println("Setup complete"); // Print a message to the serial monitor
    Serial.println("Waiting for minute ending in 1 or 6 to start report"); // Print a
message to the serial monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED

    minuteInt = getMinute(); // Get the current minute

    if (minuteInt % 5 == 1) {
        Serial.println("Starting report..."); // Print a message to the serial
```

```
monitor
        rainCount = 0; // Reset the rain count
        for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction array
                wind[i].reading = false;
                wind[i].direction = 0;
                wind[i].speed = 0;
        }
        for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of the
report, taking measurements every 10 seconds
            Serial.print("Taking wind measurements, this process will be finished in:
"); // Print a message to the serial monitor to let the user know the program is
working
            remainingMinutes = duration - ((i / 6) + !!(i % 6));
            remainingSeconds = 60 - ((i % 6) * 10);
            Serial.print(remainingMinutes);
            Serial.print(":");
            if (remainingSeconds == 60) {
                Serial.println("00");
            } else {
                Serial.println(remainingSeconds);
            }

            windSpeedCount = 0; // Reset the wind speed count
            delay(10 * 1000); // Wait for a minute
            wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the wind
speed
            wind[i].direction = getWindDirection(); // Calculate the wind direction
            wind[i].reading = true; // Set the reading flag to true
        }
        weather.rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate the
rain fall in inches/hour
        getAverageWind(averageWindDirection, averageWindSpeed); // Get the average
wind direction and speed
        weather.windDirection = averageWindDirection; // Set the average wind
direction to the weather data
        weather.windSpeed = averageWindSpeed; // Set the average wind speed to the
weather data
        weather.temperature = bme280.readTempC(); // Get the temperature in degrees
Celsius
        weather.humidity = bme280.readFloatHumidity(); // Get the humidity in percent
        weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees
Celsius
        weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); // Convert
the pressure from Pascals to inches of mercury

        weather.uva = veml6075.uva(); // Get the UVA value
        weather.uvb = veml6075.uvb(); // Get the current time in UTC
        weather.uvindex = veml6075.index(); // Get the UV index

        weather.time = getUTCTime(); // Get the current time in UTC
        printWeather(); // Print the weather data to the serial monitor
        sendWeatherDataToPocketBase(); // Send the weather data to PocketBase
    }
    delay(1000); // Wait for a second
    digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}
```

```
String getUTCTime() {
    struct timeval tv;
    gettimeofday(&tv, nullptr);
    time_t now = tv.tv_sec;
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    char buffer[30];
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.%03ldZ",
        timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
        timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
        tv.tv_usec / 1000);
    return String(buffer);
}

int getMinute() { // Get the current minute
    currentTime = getUTCTime();
    minuteString = currentTime.substring(14, 16);
    minuteInt = minuteString.toInt();
    return minuteInt;
}

void connectToWifi() {
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password); // Connect to the network

    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP()); // Print the local IP address
}

void printWeather() {
    Serial.print("Time: ");
    Serial.print(weather.time); // Time in UTC
    Serial.print("Z   Temperature: ");
    Serial.print(weather.temperature, 2); // Temperature in degrees Celsius
    Serial.print("   Humidity: ");
    Serial.print(weather.humidity, 2); // Humidity in percent
    Serial.print("   Dewpoint: ");
    Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
    Serial.print("   Pressure: ");
    Serial.print(weather.pressure, 2); // Pressure in inches of mercury
    Serial.print("   Wind Speed: ");
    Serial.print(weather.windSpeed, 2); // Wind speed in knots
    Serial.print("   Wind Direction: ");
    Serial.print(weather.windDirection, 2); // Average wind direction in degrees
    Serial.print("   Rain Fall: ");
    Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
```

```
    Serial.print("   UVA: ");
    Serial.print(weather.uva); // UVA value
    Serial.print("   UVB: ");
    Serial.print(weather.uvb); // UVB value
    Serial.print("   UV Index: ");
    Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value from the wind
direction sensor

    if (  10 < reading && reading <=  150) return (288);
    if ( 150 < reading && reading <=  250) return (264);
    if ( 250 < reading && reading <=  400) return (336);
    if ( 400 < reading && reading <=  600) return (312);
    if ( 650 < reading && reading <=  850) return ( 24);
    if ( 850 < reading && reading <= 1050) return (  0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}

void getAverageWind(float& averageWindDirection, float& averageWindSpeed) {
    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // convert angle to radians
        x = wind[i].speed * cos(theta);
        y = wind[i].speed * sin(theta);
        xSum += x;
        ySum += y;
    }
    averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum / (duration *
6))); // convert radians to degrees
    if (averageWindDirection < 0) averageWindDirection += 360; // convert negative
angles to positive
    averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum / (duration *
6), 2)); // calculate average speed

    xSum = 0;
    ySum = 0;
}

void rainIRQ()
{
    rainCount++;
    // Serial.println("Rain clicked");
}
```

```
// Function is called when the magnet in the anemometer is activated
void windSpeedIRQ()
{
    windSpeedCount++;
    // Serial.println("Wind clicked");
}

void sendWeatherDataToPocketBase() {
    Serial.println("Sending weather data to PocketBase...");

    HTTPClient http;

    // Set the PocketBase endpoint URL
    http.begin("https://w.arias.pw/api/collections/station/records");

    // Set the HTTP headers
    http.addHeader("Content-Type", "application/json");

    // Create the JSON payload
    String payload = "{\"time\":\"" + weather.time +
                     "\",\"temperature\":" + String(weather.temperature) +
                     ",\"humidity\":" + String(weather.humidity) +
                     ",\"dewpoint\":" + String(weather.dewpoint) +
                     ",\"pressure\":" + String(weather.pressure) +
                     ",\"rainFall\":" + String(weather.rainFall) +
                     ",\"windSpeed\":" + String(weather.windSpeed) +
                     ",\"windDirection\":" + String(weather.windDirection) +
                     ",\"uva\":" + String(weather.uva) +
                     ",\"uvb\":" + String(weather.uvb) +
                     ",\"uvindex\":" + String(weather.uvindex) + "}";

    // Send the POST request with the payload
    int httpCode = http.POST(payload);

    // Check if the request was successful
    if(httpCode == 200) {
        Serial.println("Data sent to PocketBase successfully");
    } else {
        Serial.println("Error sending data to PocketBase");
        Serial.print("HTTP code: ");
        Serial.println(httpCode);
    }

    // Free resources
    http.end();
}
```

Station - PocketBase - SD

Código para leer los datos de los sensores en conjunto, enviar los datos a la base de datos de PocketBase y guardarlos en una tarjeta SD.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

`Station_PocketBase_SD/Station_PocketBase_SD.ino`

```
// Station code for the Sparkfun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start

#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include "SparkFunBME280.h"
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <SparkFun_VEML6075_Arduino_Library.h>
#include <esp_sleep.h>

File dataFile; // File to store the weather data

#if defined(ARDUINO_ARCH_APOLLO3)
const int chipSelect = CS;
#else
const int chipSelect = SS;
#endif

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

bool wifiConnected = true; // Variable to store the Wi-Fi connection status

const int duration = 10; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the remaining minutes
and seconds

volatile int windSpeedCount = 0; // Variable to store the number of wind pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y values
float x, y, theta, averageWindDirection, averageWindSpeed, rainFall; // Variables to
store the x, y, theta, average wind direction and speed, and the rain fall

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
    float uvindex;
};
WeatherData weather;
```

```
struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};
WindData wind[duration * 6];

const char* ssid = "Hogwarts"; // Network SSID
const char* password = "zV9%E^%tJNd!yaW*"; // Network password

const char* ntpServer = "pool.ntp.org"; // NTP server
const long gmtOffset_sec = 0; // Offset from GMT
const int daylightOffset_sec = 0; // Offset from daylight savings time

void setup() {
    Serial.begin(115200); // Start the serial monitor
    while (!Serial); // Wait for user to open serial monitor

    Serial.println("Starting MicroMod Weather Station..."); // Print a message to the
serial monitor

    Serial.println("Starting SD card..."); // Print a message to the serial monitor
    if (!SD.begin(chipSelect)) { // Check if the SD card is present
        Serial.println("SD card initialization failed"); // Print a message to the
serial monitor
        while(1); // Wait for the user to fix the problem
    }
    Serial.println("SD card started"); // Print a message to the serial monitor

    Serial.println("Starting Wi-Fi..."); // Print a message to the serial monitor
    wifiConnected = connectToWifi(); // Obtain the Wi-Fi connection status

    if (wifiConnected) {
        Serial.println("Connection to Wi-Fi successful"); // Print a message to the
serial monitor
        Serial.println("Starting time sync..."); // Print a message to the serial
monitor
        configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
        while (!time(nullptr)) {
            delay(1000);
            Serial.println("Waiting for time sync...");
        }
        Serial.println("Time synced");
    } else {
        Serial.println("Connection to Wi-Fi failed"); // Print a message to the
serial monitor
        Serial.println("Proceeding without Wi-Fi..."); // Print a message to the
serial monitor
    }

    Wire.begin(); // Join the I2C bus

    Serial.println("Starting BME280...");
    if (bme280.begin() == false) { // Connect to the BME280
        Serial.println("BME280 did not respond. Please check your wiring and try
```

```
again"); // Print an error message if the BME280 does not respond
        Serial.println("Restarting..."); // Print a message to the serial monitor
        delay(1000); // Wait for the message to be printed
        esp_restart(); // Reset the ESP32
    } else {
        Serial.println("BME280 started"); // Print a message to the serial monitor
    }

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond.");
        Serial.println("Restarting..."); // Print a message to the serial monitor
        delay(1000); // Wait for the message to be printed
        esp_restart(); // Reset the ESP32
    } else {
        Serial.println("VEML6075 started");
    }

    pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
    pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as an input
    pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
    attachInterrupt(digitalPinToInterrupt(windSpeedSensor), windSpeedIRQ,
FALLING); // Attach the wind speed interrupt
    attachInterrupt(digitalPinToInterrupt(rainSensor), rainIRQ, FALLING); // Attach
the rain interrupt
    interrupts(); // Enable interrupts

    Serial.println("Setup complete"); // Print a message to the serial monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED

    Serial.println("Starting report..."); // Print a message to the serial monitor

    getWindandRainMeasurements(averageWindDirection, averageWindSpeed, rainFall); //
Get the average wind direction and speed

    weather.windDirection = averageWindDirection; // Set the average wind direction
to the weather data
    weather.windSpeed = averageWindSpeed; // Set the average wind speed to the
weather data
    weather.rainFall = rainFall; // Set the rain fall to the weather data
    weather.temperature = bme280.readTempC(); // Get the temperature in degrees
Celsius
    weather.humidity = bme280.readFloatHumidity(); // Get the humidity in percent
    weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees Celsius
    weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); // Convert the
pressure from Pascals to inches of mercury
    weather.uva = veml6075.uva(); // Get the UVA value
    weather.uvb = veml6075.uvb(); // Get the current time in UTC
    weather.uvindex = veml6075.index(); // Get the UV index

    if (!wifiConnected) {
        weather.time = "1970-01-01 00:00:00.000Z"; // Set the time to "No Wi-Fi" if
the ESP32 is not connected to Wi-Fi
        printWeather(); // Print the weather data to the serial monitor
```

```
        writeDataToSDCard(); // Write the weather data to the SD card

        Serial.println("Restarting..."); // Print a message to the serial monitor
        delay(1000); // Wait for the message to be printed
        esp_restart(); // Reset the ESP32
    }

    weather.time = getUTCTime(); // Get the current time in UTC
    printWeather(); // Print the weather data to the serial monitor

    writeDataToSDCard(); // Write the weather data to the SD card

    sendWeatherDataToPocketBase(); // Send the weather data to PocketBase

    Serial.println("Report complete"); // Print a message to the serial monitor to
let the user know the program is working
    digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}

String getUTCTime() {
    struct timeval tv;
    gettimeofday(&tv, nullptr);
    time_t now = tv.tv_sec;
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    char buffer[30];
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.%03ldZ",
        timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
        timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
        tv.tv_usec / 1000);
    return String(buffer);
}

bool connectToWifi() { // Connect to the Wi-Fi network
    Serial.print("Connecting to ");
    Serial.println(ssid);

    unsigned long startTime = millis(); // Get the current time

    WiFi.begin(ssid, password); // Connect to the network

    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
        delay(500);
        Serial.print(".");
        if (millis() - startTime > 60000) { // If it's been more than 1 minute
            Serial.println("");
            Serial.println("WiFi connection timed out");
            return false; // Return false
        }
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP()); // Print the local IP address
```

```
    return true; // Return true if connection was successful
}


void printWeather() {
    Serial.print("Time: ");
    Serial.print(weather.time); // Time in UTC
    Serial.print("   Temperature: ");
    Serial.print(weather.temperature, 2); // Temperature in degrees Celsius
    Serial.print("   Humidity: ");
    Serial.print(weather.humidity, 2); // Humidity in percent
    Serial.print("   Dewpoint: ");
    Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
    Serial.print("   Pressure: ");
    Serial.print(weather.pressure, 2); // Pressure in inches of mercury
    Serial.print("   Wind Speed: ");
    Serial.print(weather.windSpeed, 2); // Wind speed in knots
    Serial.print("   Wind Direction: ");
    Serial.print(weather.windDirection, 2); // Average wind direction in degrees
    Serial.print("   Rain Fall: ");
    Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
    Serial.print("   UVA: ");
    Serial.print(weather.uva); // UVA value
    Serial.print("   UVB: ");
    Serial.print(weather.uvb); // UVB value
    Serial.print("   UV Index: ");
    Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value from the wind
direction sensor

    if (  10 < reading && reading <=  150) return (288);
    if ( 150 < reading && reading <=  250) return (264);
    if ( 250 < reading && reading <=  400) return (336);
    if ( 400 < reading && reading <=  600) return (312);
    if ( 650 < reading && reading <=  850) return ( 24);
    if ( 850 < reading && reading <= 1050) return (  0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}

void getWindandRainMeasurements(float& averageWindDirection, float& averageWindSpeed,
float& rainFall) {
    rainCount = 0; // Reset the rain count
```

```
    for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction array
            wind[i].reading = false;
            wind[i].direction = 0;
            wind[i].speed = 0;
    }
    for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of the
report, taking measurements every 10 seconds
        Serial.print("Taking wind and rain measurements, this process will be
finished in: "); // Print a message to the serial monitor to let the user know the
program is working
        remainingMinutes = duration - ((i / 6) + !!(i % 6));
        remainingSeconds = 60 - ((i % 6) * 10);
        Serial.print(remainingMinutes);
        Serial.print(":");
        if (remainingSeconds == 60) {
            Serial.println("00");
        } else {
            Serial.println(remainingSeconds);
        }

        windSpeedCount = 0; // Reset the wind speed count
        delay(10 * 1000); // Wait for a minute
        wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the wind speed
        wind[i].direction = getWindDirection(); // Calculate the wind direction
        wind[i].reading = true; // Set the reading flag to true
    }
    rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate the rain fall in
inches/hour

    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // Convert angle to radians
        x = wind[i].speed * cos(theta);
        y = wind[i].speed * sin(theta);
        xSum += x;
        ySum += y;
    }
    averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum / (duration *
6))); // Convert radians to degrees
    if (averageWindDirection < 0) averageWindDirection += 360; // Convert negative
angles to positive
    averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum / (duration *
6), 2)); // Calculate average speed

    xSum = 0;
    ySum = 0;
}

void rainIRQ() // Interrupt called when the magnet in the rain gauge is activated
{
    rainCount++;
}

void windSpeedIRQ() // Interrupt called when the magnet in the anemometer is
activated
{
    windSpeedCount++;
```

```
}

void sendWeatherDataToPocketBase() { // Send the weather data to PocketBase
    Serial.println("Sending weather data to PocketBase...");

    HTTPClient http; // Create an HTTPClient object

    http.begin("https://w.arias.pw/api/collections/station/records"); // Set the
PocketBase endpoint URL

    http.addHeader("Content-Type", "application/json"); // Set the HTTP headers

    // Create the JSON payload
    String payload = "{\"time\":\"" + weather.time +
                     "\",\"temperature\":" + String(weather.temperature) +
                     ",\"humidity\":" + String(weather.humidity) +
                     ",\"dewpoint\":" + String(weather.dewpoint) +
                     ",\"pressure\":" + String(weather.pressure) +
                     ",\"rainFall\":" + String(weather.rainFall) +
                     ",\"windSpeed\":" + String(weather.windSpeed) +
                     ",\"windDirection\":" + String(weather.windDirection) +
                     ",\"uva\":" + String(weather.uva) +
                     ",\"uvb\":" + String(weather.uvb) +
                     ",\"uvindex\":" + String(weather.uvindex) + "}";

    int httpCode = http.POST(payload); // Send the POST request

    if(httpCode == 200) { // Check the returning code
        Serial.println("Data sent to PocketBase successfully");
    } else { // If the code is not 200, something went wrong
        Serial.print("Error sending data to PocketBase, returned code: ");
        Serial.println(httpCode);

        Serial.println("Restarting ESP32...");
        delay(1000); // Wait for the serial output to finish
        esp_restart(); // Restart the ESP32
    }

    http.end(); // Close connection
}

void writeDataToSDCard() { // Write the weather data to the SD card
    Serial.println("Writing weather data to SD card...");

    for (int i = 0; i < 3; i++) { // Try opening the file up to 3 times
        dataFile = SD.open("/data.csv", FILE_APPEND); // Open the data file

        if (dataFile) { // If the file opened successfully, write the data
            dataFile.print(weather.time);
            dataFile.print(",");
            dataFile.print(weather.temperature);
            dataFile.print(",");
            dataFile.print(weather.humidity);
            dataFile.print(",");
            dataFile.print(weather.dewpoint);
            dataFile.print(",");
```

```
            dataFile.print(weather.pressure);
            dataFile.print(",");
            dataFile.print(weather.rainFall);
            dataFile.print(",");
            dataFile.print(weather.windSpeed);
            dataFile.print(",");
            dataFile.print(weather.windDirection);
            dataFile.print(",");
            dataFile.print(weather.uva);
            dataFile.print(",");
            dataFile.print(weather.uvb);
            dataFile.print(",");
            dataFile.println(weather.uvindex);
            dataFile.close(); // Close the file
            Serial.println("Data written to SD card successfully");
            return; // Exit the function after successful write
        } else { // If the file did not open successfully, print an error
            Serial.println("Error opening file on SD card. Retrying...");
            delay(500); // Wait for half a second before retrying
        }
    }

    // If all attempts to open the file have failed, print an error message and
restart the ESP32
    Serial.println("Failed to write data to SD card after multiple attempts.");
    Serial.println("Restarting ESP32...");

    delay(1000); // Wait for the serial output to finish
    esp_restart(); // Restart the ESP32
}
```

Servidor

AccuWeather

Código para obtener los datos de AccuWeather interpretar los datos y enviarlos a PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos deben ser reemplazadas por las correspondientes.

accuweather.py

```
import requests
import json
from datetime import datetime

# Define API endpoint and parameters
location_key = "3570769" # Location key
api_key = "ACCUWEATHERAPIKEY" # AccuWeather API key
pocketbase_api_url = "https://your.domain/api/collections/accuweather/records" #
Pocketbase API endpoint

# Define headers
headers = {
    "Content-Type": "application/json",
}

# Define function to get current time
def execution_time():
```

```
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Print execution time
print(execution_time() + " - Executing script...")

# Make API request
print(execution_time() + " - Making request to AccuWeather...")
response = requests.get("http://dataservice.accuweather.com/currentconditions/v1/" +
location_key + "/historical/24?apikey=" + api_key + "&language=en-
us&details=true&metric=true")
data = json.loads(response.text)
data = json.dumps(data, indent=4)

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Request to AccuWeather successful!")
    # Parse JSON response
    for i in range(0, len(json.loads(data))):
        new_data = {
            "time": datetime.utcfromtimestamp(json.loads(data)[i]
['EpochTime']).strftime('%Y-%m-%d %H:%M:%S.000Z'),
            "temperature": json.loads(data)[i]['Temperature']['Metric']['Value'],
            "realFeelTemperature": json.loads(data)[i]['RealFeelTemperature']
['Metric']['Value'],
            "realFeelTemperatureShade": json.loads(data)[i]
['RealFeelTemperatureShade']['Metric']['Value'],
            "relativeHumidity": json.loads(data)[i]['RelativeHumidity'],
            "indoorRelativeHumidity": json.loads(data)[i]['IndoorRelativeHumidity'],
            "dewPoint": json.loads(data)[i]['DewPoint']['Metric']['Value'],
            "windDirection": json.loads(data)[i]['Wind']['Direction']['Degrees'],
            "windSpeed": json.loads(data)[i]['Wind']['Speed']['Metric']['Value'],
            "uvIndex": json.loads(data)[i]['UVIndex'],
            "visibility": json.loads(data)[i]['Visibility']['Imperial']['Value'],
            "pressure": json.loads(data)[i]['Pressure']['Imperial']['Value'],
            "apparentTemperature": json.loads(data)[i]['ApparentTemperature']
['Metric']['Value'],
            "precipitation": json.loads(data)[i]['Precip1hr']['Metric']['Value'],
        }
        # Send data to PocketBase API
        print(execution_time() + " - Sending data to PocketBase with date and time: "
+ new_data['time'])

        response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

        # Check if response was successful
        if response.status_code == 200:
            print(execution_time() + " - Data sent to PocketBase successfully!")
        else:
            print(execution_time() + " - Request to PocketBase failed! with status
code: " + str(response.status_code))

else:
    print(execution_time() + "Request to AccuWeather failed! with response code: " +
str(response.status_code))
```

AWC

Código para obtener los datos de AWC interpretar los datos y enviarlos a PocketBase.

awc.py

```
# Description: Fetches METAR data from Aviation Weather Center API and posts it to
PocketBase API

# Import libraries
import requests
import csv
import json
from datetime import datetime

# Define constants
API_URL = "https://www.aviationweather.gov/adds/dataserver_current/httpparam" #
Aviation Weather Center API URL
POCKETBASE_API_URL = "https://your.domain/api/collections/awc/records" # PocketBase
API URL

# Define variables
data_type = "metars"
airport_code = "MMLO" # Airport code
hours_before_now = "24" # Number of hours before now to fetch data for
output_format = "csv"

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Define function to fetch CSV data
def fetch_csv_data(api_url, parameters):
    print(execution_time() + " - Making request to Aviation Weather Center...")
    # Make the API request
    response = requests.get(api_url, params=parameters)
    # Check for successful request
    if response.status_code == 200:
        # Decode the CSV content
        content = response.content.decode("utf-8")
        # Parse the CSV data and return it
        csv_data = list(csv.reader(content.splitlines(), delimiter=","))
        return csv_data
    else:
        print(execution_time() + " - Request to Aviation Weather Center failed! with
status code: " + str(response.status_code))

# Set up the API request parameters
parameters = {
    "dataSource": data_type,
    "requestType": "retrieve",
    "format": output_format,
    "stationString": airport_code,
    "hoursBeforeNow": hours_before_now,
}

# Print execution time
print(execution_time() + " - Executing script...")
```

```
# Call the function to fetch the CSV data
csv_data = fetch_csv_data(API_URL, parameters)

# Check if there are results in the CSV data
if len(csv_data) <= 6:
    print(execution_time() + " - No results found in CSV data received from Aviation
Weather Center")
else:
    print(execution_time() + " - CSV data received from Aviation Weather Center
successfully!")
    # Parse the remaining rows of the CSV data and convert to JSON
    print(execution_time() + " - Parsing CSV data and converting it to JSON...")
    for row in csv_data[6:]: # Skip the first 6 rows
        # Get the headers from the first row of the CSV data
        headers = csv_data[5]
        # Create a dictionary with the desired keys and values
        data = {
            "raw_text": row[headers.index("raw_text")],
            "station_id": airport_code,
            "observation_time":
datetime.strptime(row[headers.index("observation_time")], "%Y-%m-%dT%H:%M:
%SZ").strftime("%Y-%m-%d %H:%M:%S.000Z"),
            "temp_c": float(row[headers.index("temp_c")]),
            "dewpoint_c": float(row[headers.index("dewpoint_c")]),
            "wind_dir_degrees": int(row[headers.index("wind_dir_degrees")]) if
row[headers.index("wind_dir_degrees")] else 0,
            "wind_speed_kt": int(row[headers.index("wind_speed_kt")]) if
row[headers.index("wind_speed_kt")] else 0,
            "altim_in_hg": float(row[headers.index("altim_in_hg")]),
            "corrected": bool(row[headers.index("corrected")]),
            "precip_in": float(row[headers.index("precip_in")]) if
row[headers.index("precip_in")] else 0,
            "metar_type": row[headers.index("metar_type")],
        }

        # Set up the headers for the POST request
        headers = {'Content-Type': 'application/json'}

        # Make the POST request to PocketBase API
        print(execution_time() + " - Sending data to PocketBase with date and time: "
+ data["observation_time"])
        response = requests.post(POCKETBASE_API_URL, data=json.dumps(data),
headers=headers)

        # Check for successful request
        if response.status_code == 200:
            print(execution_time() + " - Data sent to PocketBase successfully!")
        else:
            print(execution_time() + " - Request to PocketBase failed! with status
code: " + str(response.status_code))
```

Open-Meteo

Código para obtener los datos de Open-Meteo interpretar los datos y enviarlos a PocketBase.

`open-meteo.py`

```python
# Description: This script is used to get the weather data from the OpenMeteo API and
send it to Pocketbase

# Import libraries
import requests
import json
from datetime import datetime
from datetime import date
from datetime import timedelta

# Define API endpoint and parameters
url = "https://api.open-meteo.com/v1/forecast" # OpenMeteo API endpoint
pocketbase_api_url = "https://your.domain/api/collections/open_meteo/records" #
Pocketbase API endpoint
headers = {
    "Content-Type": "application/json",
}

# Get previous day
previous_day = (datetime.utcnow() - timedelta(days=1)).strftime("%Y-%m-%d")

params = {
    "latitude": 21.01, # Latitude of the location
    "longitude": -101.49, # Longitude of the location
    "hourly":
"temperature_2m,relativehumidity_2m,dewpoint_2m,apparent_temperature,rain,pressure_msl,surface_pres
    "windspeed_unit": "kn", # Unit of the wind speed
    "precipitation_unit": "inch", # Unit of the precipitation
    "forecast_days": 1,
    "start_date": previous_day, # Start date of the forecast
    "end_date": previous_day # End date of the forecast, for one day, use the same
date as start_date
}

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Print execution time
print(execution_time() + " - Executing script...")

# Make API request
print(execution_time() + " - Making request to open-meteo...")
response = requests.get(url, params=params)

# Check if response was successful
if response.status_code == 200:
    # Print response
    print(execution_time() + " - Response from open-meteo was successful!")

    # Parse JSON response
    data = json.loads(response.text)

    # Structure data
    data = json.dumps(data, indent=4)
```

```
    # Send data to Pocketbase API one hour at a time
    for hour in range(0, len(json.loads(data)['hourly']['time'])):
        new_data = {
            "time": datetime.strptime(json.loads(data)['hourly']['time'][hour], "%Y-
%m-%dT%H:%M").strftime("%Y-%m-%d %H:%M:00.000Z"),
            "temperature_2m": json.loads(data)['hourly']['temperature_2m'][hour],
            "relativehumidity_2m": json.loads(data)['hourly']['relativehumidity_2m']
[hour],
            "dewpoint_2m": json.loads(data)['hourly']['dewpoint_2m'][hour],
            "apparent_temperature": json.loads(data)['hourly']
['apparent_temperature'][hour],
            "rain": json.loads(data)['hourly']['rain'][hour],
            "pressure_msl": json.loads(data)['hourly']['pressure_msl'][hour],
            "surface_pressure": json.loads(data)['hourly']['surface_pressure'][hour],
            "windspeed_10m": json.loads(data)['hourly']['windspeed_10m'][hour],
            "windspeed_80m": json.loads(data)['hourly']['windspeed_80m'][hour],
            "windspeed_120m": json.loads(data)['hourly']['windspeed_120m'][hour],
            "windspeed_180m": json.loads(data)['hourly']['windspeed_180m'][hour],
            "winddirection_10m": json.loads(data)['hourly']['winddirection_10m']
[hour],
            "winddirection_80m": json.loads(data)['hourly']['winddirection_80m']
[hour],
            "winddirection_120m": json.loads(data)['hourly']['winddirection_120m']
[hour],
            "winddirection_180m": json.loads(data)['hourly']['winddirection_180m']
[hour],
            "temperature_80m": json.loads(data)['hourly']['temperature_80m'][hour],
            "temperature_120m": json.loads(data)['hourly']['temperature_120m'][hour],
            "temperature_180m": json.loads(data)['hourly']['temperature_180m'][hour],
            "uv_index": json.loads(data)['hourly']['uv_index'][hour]
        }
        # Send data to Pocketbase API
        print(execution_time() + " - Sending data to PocketBase with date and time: "
+ new_data["time"])

        response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

        # Check if response was successful
        if response.status_code == 200:
            print(execution_time() + " - Request to Pocketbase was successful!")
        else:
            print(execution_time() + " - Request to Pocketbase failed! with status
code: " + str(response.status_code))
else:
    print(execution_time() + " - Request to open-meteo failed! with status code: " +
str(response.status_code))
```

OpenWeatherMap

Código para obtener los datos de OpenWeatherMap interpretar los datos y enviarlos a PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos deben ser reemplazadas por las correspondientes.

`openweathermap.py`

```
# Description: This script will make a request to the OpenWeatherMap API and save the
data to PocketBase
```

```
import requests
import json
from datetime import datetime

# Define API endpoint and parameters
api_key = "OPENWEATHERMAPAPIKEY" # OpenWeatherMap API key
latitude = 21.01 # Latitude of the location
longitude = -101.49 # Longitude of the location
url = "https://api.openweathermap.org/data/2.5/weather?lat=" + str(latitude) +
"&lon=" + str(longitude) + "&appid=" + api_key
pocketbase_api_url = "https://your.domain/api/collections/openweathermap/records" #
Pocketbase API endpoint
headers = {
    "Content-Type": "application/json",
}

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Print execution time
print(execution_time() + " - Executing script...")

# Make API request
print(execution_time() + " - Making request to OpenWeatherMap...")

response = requests.get(url)

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Request to OpenWeatherMap successful!")
    # Parse JSON response
    data = json.loads(response.text)

    data = json.dumps(data, indent=4)

    new_data = {
        "time": datetime.utcfromtimestamp(json.loads(data)['dt']).strftime('%Y-%m-%d
%H:%M:%S.000Z'),
        "temperature": json.loads(data)['main']['temp'],
        "feels_like": json.loads(data)['main']['feels_like'],
        "pressure": json.loads(data)['main']['pressure'],
        "humidity": json.loads(data)['main']['humidity'],
        "wind_speed": json.loads(data)['wind']['speed'],
        "wind_direction": json.loads(data)['wind']['deg'],
    }

    # Make API request to PocketBase
    print(execution_time() + " - Sending data to PocketBase with date and time: " +
new_data['time'])
    response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

    if response.status_code == 200:
        print(execution_time() + " - Data sent to PocketBase successfully!")
    else:
```

```
        print(execution_time() + " - Request to PocketBase API failed! with status
code: " + str(response.status_code))
else:
    print(execution_time() + " - Request to OpenWeatherMap failed! with status code:
" + str(response.status_code))
```

Recolección

RAW Data

Script para la recolección de datos sin procesar.

raw.py

```
# This script fetches data from PocketBase and writes it to a CSV file

import requests
import csv

POCKETBASE_API_URL = "https://domain.name/api/collections/" # Specify the PocketBase
API URL here
COLLECTION_NAME = "station"  # Specify the collection name here
OUTPUT_CSV_FILE = f"{COLLECTION_NAME}.csv"  # Generate the output file name

# Define function to fetch data from PocketBase using pagination
def fetch_pocketbase_data(api_url):
    page = 1
    per_page = 50
    records = []

    while True:
        params = {
            "page": page,
            "perPage": per_page,
            "sort": "-created"
        }

        response = requests.get(api_url, params=params)

        if response.status_code == 200:
            json_response = response.json()
            records += json_response.get("items", [])

            # Check if there are more pages to fetch
            if json_response["page"] == json_response["totalPages"]:
                break

            # Move to the next page
            page += 1
        else:
            print("Request to PocketBase failed with status code:",
response.status_code)
            break

    return records

# Construct the PocketBase API URL for the specified collection
pocketbase_api_url = f"{POCKETBASE_API_URL}{COLLECTION_NAME}/records"
```

```
# Call the function to fetch data from PocketBase
pocketbase_data = fetch_pocketbase_data(pocketbase_api_url)

# Check if there are results in the PocketBase data
if not pocketbase_data:
    print("No data found in PocketBase")
else:
    print("PocketBase data retrieved successfully!")
    # Extract the keys from the first record to use as CSV headers
    headers = list(pocketbase_data[0].keys())

    # Open the CSV file for writing
    with open(OUTPUT_CSV_FILE, mode="w", newline="") as file:
        writer = csv.DictWriter(file, fieldnames=headers)
        writer.writeheader()

        # Write each record to the CSV file
        for record in pocketbase_data:
            writer.writerow(record)

    print("Data written to CSV file:", OUTPUT_CSV_FILE)
```

Procesamiento y análisis de datos

Reescribir fechas

Script para reescribir las fechas de los datos, de acuerdo al timestamp de creación y no al proporcionado por la estación.

re-date.py

```
# Re-dates time values from given created values.

import os
import csv
from datetime import datetime

def change_date(input_file, output_directory):
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    output_file = os.path.join(output_directory, os.path.basename(input_file))

    correction_count = 0
    rows = []

    with open(input_file, "r") as file:
        reader = csv.DictReader(file)
        headers = reader.fieldnames

        for row in reader:
            time_value = row["time"]
            if time_value.startswith("1970"):
                created_value = row.get("created", "")
                row["time"] = created_value[:24]  # Update time value with created
time
                correction_count += 1
```

```
            rows.append(row)

    with open(output_file, "w", newline="") as outfile:
        writer = csv.DictWriter(outfile, fieldnames=headers)
        writer.writeheader()
        writer.writerows(rows)

    print("Data processing complete for", input_file)
    print("Number of corrections:", correction_count)

input_file = "manual/station-SD.csv"
output_directory = "re-dated"

change_date(input_file, output_directory)
```

Redondeo de fechas

Script para redondear las fechas de los datos, de acuerdo al valor de 10 minutos más cercano.

round.py

```
# Rounds the time values in the CSV files to the nearest 10 minutes.

import os
import csv
from datetime import datetime, timedelta

def round_to_nearest_ten_minutes(time_str):
    time = datetime.strptime(time_str, "%Y-%m-%d %H:%M:%S.%fZ")
    rounded_time = time - timedelta(minutes=time.minute % 10,
                                    seconds=time.second,
                                    microseconds=time.microsecond)
    return rounded_time.strftime("%Y-%m-%d %H:%M")

input_directory = "re-dated"
output_directory = "rounded"

if not os.path.exists(output_directory):
    os.makedirs(output_directory)

for filename in os.listdir(input_directory):
    if filename.endswith(".csv"):
        input_file = os.path.join(input_directory, filename)
        output_file = os.path.join(output_directory, filename)

        with open(input_file, "r") as file:
            reader = csv.DictReader(file)
            headers = reader.fieldnames

            with open(output_file, "w", newline="") as outfile:
                writer = csv.DictWriter(outfile, fieldnames=headers)
                writer.writeheader()

                for row in reader:
                    row["time"] = round_to_nearest_ten_minutes(row["time"])
                    writer.writerow(row)
```

```
        print("Data processing complete for", filename)

print("All files processed successfully.")
```

Interpolación y agrupación

Script para interpolar y agrupar los datos de acuerdo a la variable especificada.

interpolate.py

```
# Combine multiple csv files into a single csv file, interpolating missing data.

import os
import pandas as pd

def combine_csv_files(file_list, variable, initial_time, final_time):
    combined_data = pd.DataFrame(columns=['time', variable])

    for file_path in file_list:
        # Extract filename without extension
        filename = os.path.splitext(os.path.basename(file_path))[0]

        # Read the csv file, parsing 'time' column as datetime
        df = pd.read_csv(file_path, parse_dates=['time'])

        # Check if the variable column exists in the DataFrame
        if variable in df.columns:
            # Filter data based on initial and final time
            df = df[(df['time'] >= initial_time) & (df['time'] <= final_time)]

            # Create a new column with the data from the variable
            df[filename] = df[variable]

            # Append the relevant columns to the combined data
            combined_data = pd.merge(combined_data, df[['time', filename]],
on='time', how='outer')

    # Set 'time' column as the index
    combined_data = combined_data.set_index('time')

    # Group by index (time) and aggregate the values
    combined_data = combined_data.groupby(combined_data.index).mean()

    # Resample and interpolate missing data
    combined_data = combined_data.resample('10T').interpolate(method='time')

    # Save the combined data to a new csv file
    combined_data.to_csv(variable + '.csv')

# Example usage
file_list = ['accuweather.csv', 'awc.csv', 'bme680.csv', 'open_meteo.csv',
'openweathermap.csv', 'station-SD.csv', 'station.csv']
variable = 'windSpeed'
initial_time = '2023-04-30 23:00'
final_time = '2023-05-15 23:50'

combine_csv_files(file_list, variable, initial_time, final_time)
```