Portable Weather Station Documentation

Autors: Daniel Jose Centeno Gonzalez & Lorena Garcia Plaza

Introducction

A portable weather station offers portability, accurate data, and enhanced safety. It can be easily transported to different locations, providing precise and localized forecasts. This helps in decision-making for various activities and sectors. The station's portability enables monitoring of essential parameters, alerting users to potential weather hazards in advance. With measurements of temperature, humidity, thermal sensation, wind speed, wind direction, UV index, rain fall and then you can process the data and analyze it. In this documentation, you will find everything about this project.

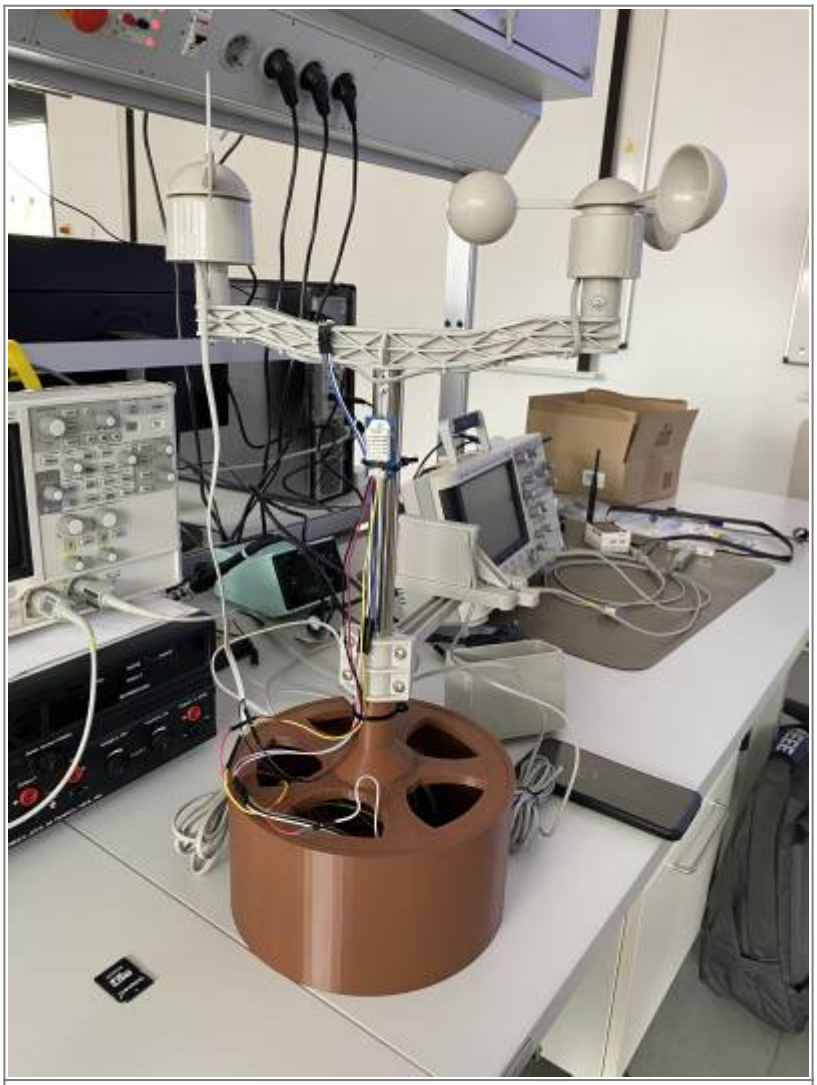

Fig.: HSRW Portable Weather Station

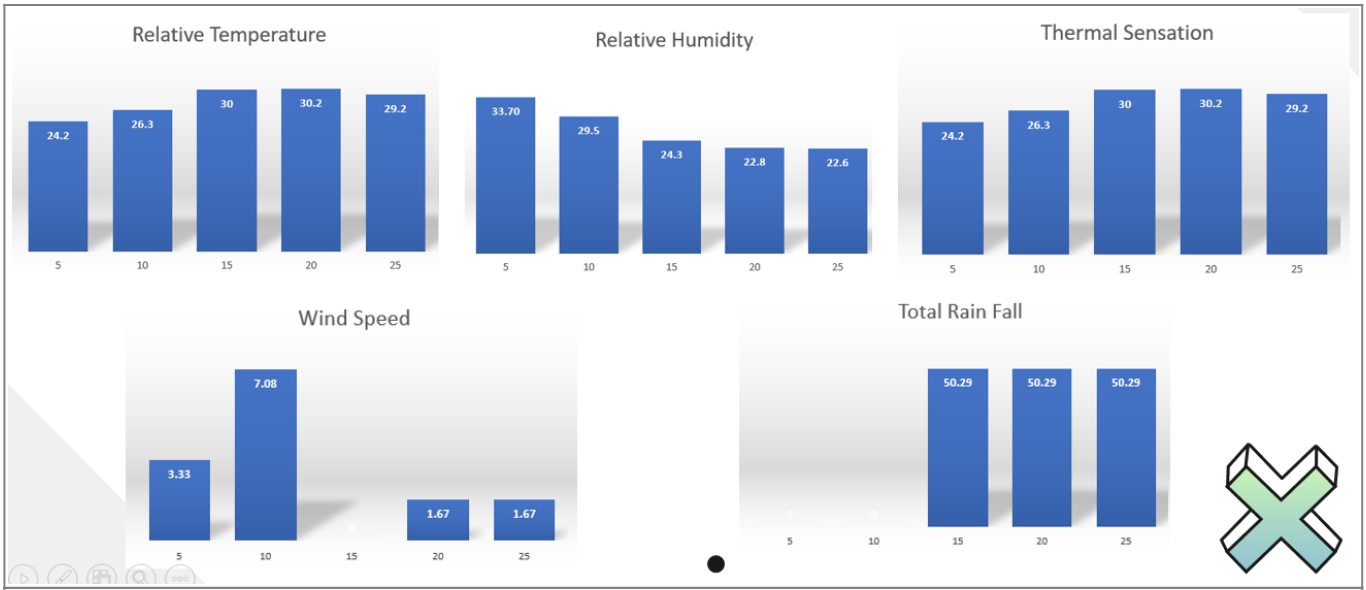

Fig.: Portable Weather Station Data Visualization

Weather Station Sensors

Measurement	Sensor	Measurement unit	Datasheet
Temperature	DHT22	ºC	DHT22
Humidity	DHT22	RH	DHT22
Thermal sensation	DHT22	ºC	DHT22
Wind Direction	Sparkfun Weather station kit (SEN-08942)	Cardinal Points	SEN-08942
Wind Speed	Sparkfun Weather station kit (SEN-08942)	m/s	SEN-08942
Rain Gauge	Sparkfun Weather station kit (SEN-08942)	mm	SEN-08942
UV index sensor	P1918 UV sensor	1-11	P1918 UV sensor

Electric diagram

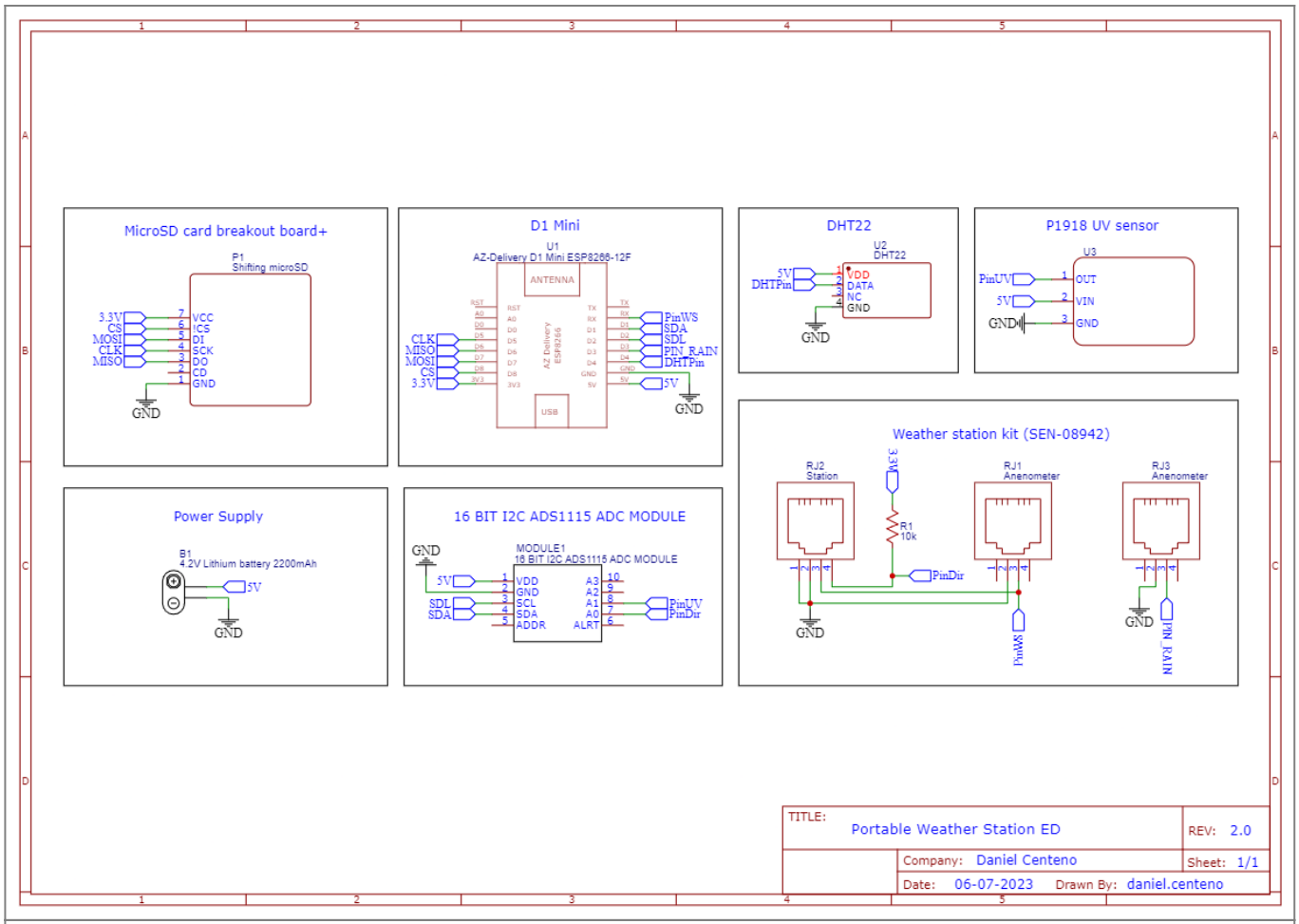

Fig.: HSRW Portable Weather Station Electric diagram, You can download the pdf diagram Here

Real Time Data Visualization

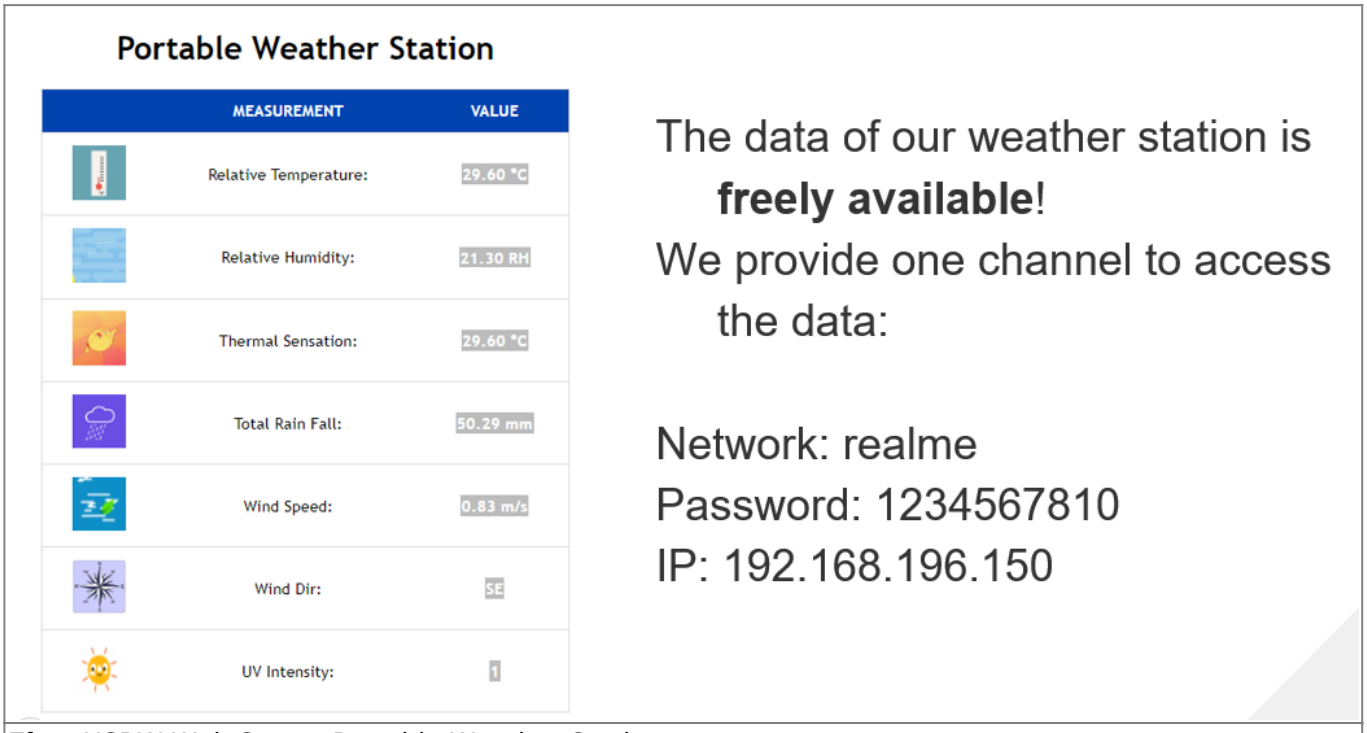

Fig.: HSRW Web Server Portable Weather Station

Logic Code

The logic of the program is divided into 5 important functions of which 4 of them are in charge of carrying out the measurement and calculation of the different values of each sensor and the last remaining one in the storage of the data in the MicroSD. Separating each function in this way we can create a multiprogramming environment which is extremely important in this type of project that uses many resources at the same time. This will also create a list and proper times of each component. For example, data can be stored in the SD while a user connects to the server. These processes will never be interrupted thanks to the task manager created with the millis() function.

Sensor Readings:

read_Temp_Humi(): This function is responsible for measuring temperature and humidity using the DHT22 sensor. First, the humidity and temperature are read in degrees Celsius. If the reading from any of the sensors is invalid, an error message is displayed on the serial port. In addition to obtaining the measurements, the thermal sensation (heat index) is also calculated using the Steadman formula, which takes humidity into account to estimate how the human body perceives temperature.

getRainfall(): The getRainfall() function takes care of measuring the amount of rain that has fallen. For this, a rain gauge is used, which records the number of times the rain causes a mechanism to activate (in this case, the countRain() function). Each time the mechanism is activated, a counter representing the number of times it has rained is incremented. The getRainfall() function takes the value of the counter and converts it to a rain measurement in millimeters, considering a relationship between the activations and the amount of rain.

UV_WindDir(): This function is responsible for obtaining the intensity of ultraviolet (UV) radiation and the direction of the wind. The ADS1115 module is used to read the analog value of the UV radiation sensor and then this value is mapped to a scale of 0 to 11 to represent the percentage of UV intensity. In addition, the ADS1115 module is also used to measure the analog value of the wind direction sensor. The UV_WindDir() function interprets the read value and determines the wind direction based on predefined ranges for different cardinal directions.

countPulse(): The countPulse() function is responsible for measuring the wind speed using an anemometer. The anemometer is designed to generate pulses when driven by the wind. Each time a pulse is detected, a counter representing the number of pulses generated by the anemometer in a specified time interval is incremented. The loop() function then calculates the wind speed from the number of pulses and a scale factor based on the anemometer's calibration.

Data Logging:

SaveFun() Every five minutes, the sensor readings are saved to a CSV file on the SD card. The data includes the timestamp, temperature, humidity, thermal sensation, total rainfall, wind speed, wind direction, and UV intensity.

Raw Code

```
/* Tittle: Portable Weather Station
```

```
 * Autors: Daniel Centeno
 *         Lorena Garcia
 *
 * Version: 2.7.9
 */

//Libraries
#include <Wire.h>
#include <ESP8266WiFi.h>
#include <DHT.h>
#include <Adafruit_ADS1X15.h>
#include <SPI.h>
#include <SD.h>

//*************************************************************************
******
File dataFile;
const int chipSelect = 15;  // Chip select pin connected to SD card breakout
board

//*************************************************************************
******
Adafruit_ADS1115 ads;

//*******************************************************
//Temperature sensor variables

#define DHTPIN 2     // Pin connected to the DHT22 sensor D4
#define DHTTYPE DHT22 // DHT sensor type
DHT dht(DHTPIN, DHTTYPE);

float temperature = 0;
float humidity = 0;
float heatIndex = 0;

//*****************************************************************
//Variables for the UV sensor
int uvPercentage = 0;



//*****************************************************
//All WiFi configuration
// Replace with your network credentials
const char* ssid     = "realme";
const char* password = "1234567810";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;
```

```
// Current time
unsigned long WiFicurrentTime = millis();
// Previous time
unsigned long WiFipreviousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long WiFitimeoutTime = 2000;

//*****************************************************************
//Pines kit Weater Station
#define PIN_RAIN 0      //D3
#define SDA 4           //D2
#define SCL 5           //D1

//*****************************************************************

volatile int numRevsAnemometer = 0;
volatile int numTipsRain = 0;
float totalRain = 0;

//*****************************************************************

unsigned long previousMillis = 0;   // Stores the last time the sensor was
read
const long interval = 1000;         // Read sensor every 1 second
unsigned long currentMillis = 0;    // Variable where will stores the
current millis


//*****************************************************************

void countRain(); // Function prototype for countRain()

void ICACHE_RAM_ATTR countRain() {
   numTipsRain++;
}

//*****************************************************************

unsigned long previousMillis2 = 0;   // Stores the last time the sensor was
read
const long interval2 = 1500;         // Read sensor every 1 second
unsigned long currentMillis2 = 0;    // Variable where will stores the
current millis

//Wind Dir
char WinDirVariable[3];

//*****************************************************************

unsigned long lastWindCheck2 = 0;
```

```
unsigned long currentTime2 = 0;
int Tolerance = 300;
int ValuemV = 0;

//*****************************************************************

unsigned long currentTimeSave = 0;

//*****************************************************************

int anemometerPin = 3;
volatile unsigned long windCount = 0;
unsigned long lastWindCheckWind = 0;
float windSpeed = 0;

void ICACHE_RAM_ATTR countPulse() {
  windCount++;
}

//*****************************************************************

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  dht.begin();
  ads.begin();

  pinMode(PIN_RAIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(PIN_RAIN), countRain, FALLING);

  pinMode(anemometerPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(anemometerPin), countPulse, CHANGE);

  bool status;
  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();

  // Initialize SD card
  if (!SD.begin(chipSelect)) {
```

```
    Serial.println("SD card initialization failed!");
    return;
  }

  // Open the data file in append mode
  dataFile = SD.open("Weather_Station_sensors_data.csv", FILE_WRITE);

  if (dataFile) {
    // Write headers to the file
    dataFile.println("Timestamp, Relative Temperature, Relative Humidity,
Thermal Sensation, Total Rain Fall, Wind Speed, Wind Dir, UV Intensity");
    dataFile.close();
    Serial.println("Data file initialized.");
  } else {
    Serial.println("Error opening data file.");
  }

}

void loop() {
  // put your main code here, to run repeatedly:

  //*********************************************************

  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    WiFicurrentTime = millis();
    WiFipreviousTime = WiFicurrentTime;
    Serial.println("New Client.");          // print a message out in the
serial port
    String currentLine = "";                // make a String to hold
incoming data from the client
    while (client.connected() && WiFicurrentTime - WiFipreviousTime <=
WiFitimeoutTime) {  // loop while the client's connected
      WiFicurrentTime = millis();
      if (client.available()) {             // if there's bytes to read from
the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial
monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline
character
          // if the current line is blank, you got two newline characters in
a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1
200 OK)
            // and a content-type so the client knows what's coming, then a
```

```
blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the table
            client.println("<style>body { text-align: center; font-family:
\"Trebuchet MS\", Arial;}");
            client.println("table { border-collapse: collapse; width:35%;
margin-left:auto; margin-right:auto; }");
            client.println("th { padding: 12px; background-color: #0043af;
color: white; }");
            client.println("tr { border: 1px solid #ddd; padding: 12px; }");
            client.println("tr:hover { background-color: #bcbcbc; }");
            client.println("td { border: none; padding: 12px; }");
            client.println(".sensor { color:white; font-weight: bold;
background-color: #bcbcbc; padding: 1px; }");

            // Web Page Heading
            client.println("</style></head><body><h1>Portable Weather
Station</h1>");
client.println("<table><tr><th></th><th>MEASUREMENT</th><th>VALUE</th></tr>"
);

            client.println("<td style=\"text-align: center;\"><img
src=\"https://media.tenor.com/rs1yXZWnBOgAAAAM/gif-arts.gif\" alt=\"GIF
Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Relative Temperature: </td><td><span
class=\"sensor\">");
            client.println(temperature);
            client.println(" *C</span></td></tr>");

            client.println("<td style=\"text-align: center;\"><img
src=\"https://media2.giphy.com/media/l41m39GpkjPpbOqqc/giphy.gif\" alt=\"GIF
Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Relative Humidity: </td><td><span
class=\"sensor\">");
            client.println(humidity);
            client.println(" RH</span></td></tr>");

            client.println("<td style=\"text-align: center;\"><img
src=\"https://media1.giphy.com/media/v1.Y2lkPTc5MGI3NjExZTFoejE0eHBlbXhmbHN6
bzdkeGtybm9veWM0M3JueGplbzI1NWNwcSZlcD12MV9naWZzX3NlYXJjaCZjdD1n/J43yvW6uFab
IpBBsPO/giphy.gif\" alt=\"GIF Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Thermal Sensation: </td><td><span
```

```
class=\"sensor\">");
            client.println(heatIndex);
            client.println(" *C</span></td></tr>");

            client.println("<td style=\"text-align: center;\"><img
src=\"https://media.tenor.com/jOWWXvdGaUIAAAAC/rainfall.gif\" alt=\"GIF
Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Total Rain Fall: </td><td><span
class=\"sensor\">");
            client.println(totalRain);
            client.println(" mm</span></td></tr>");

            client.println("<td style=\"text-align: center;\"><img
src=\"https://cdn.dribbble.com/users/1028385/screenshots/2924553/type-tuesda
y-leaf-in-wind.gif\" alt=\"GIF Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Wind Speed: </td><td><span
class=\"sensor\">");
            client.println(windSpeed);
            client.println(" m/s</span></td></tr>");


            client.println("<td style=\"text-align: center;\"><img
src=\"https://www.cliparts101.com/files/140/040F7BE8E61CC2A6CAFCD2451A42A63A
/Wind_rose_icon.png\" alt=\"GIF Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>Wind Dir: </td><td><span
class=\"sensor\">");
            client.println(WinDirVariable);
            client.println("</span></td></tr>");

            client.println("<td style=\"text-align: center;\"><img
src=\"https://cdn.dribbble.com/users/475418/screenshots/3653921/suns.gif\"
alt=\"GIF Image\" width=\"54\" height=\"54\"></td>");
            client.println("<td>UV Intensity: </td><td><span
class=\"sensor\">");
            client.println(uvPercentage);
            client.println("</span></td></tr>");
            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage
return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
```

```
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }


  //*****************************************************

  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    read_Temp_Humi();
  }

  getRainfall();


  unsigned long currentTimeWind = millis();

  if (currentTimeWind - lastWindCheckWind >= 1000) {
    detachInterrupt(digitalPinToInterrupt(anemometerPin)); // Stop
interruptions

    // Calculate the Wind Speed
    windSpeed = (float)windCount / 2.4; // Sets the scale factor based on
the anemometer calibration
    windCount = 0; // Reset pulse counter
    lastWindCheckWind = currentTimeWind; //Update last check time

    attachInterrupt(digitalPinToInterrupt(anemometerPin), countPulse,
CHANGE); // Re-enable interrupts
  }

  currentMillis2 = millis();
  if (currentMillis2 - previousMillis2 >= interval2) {
    previousMillis2 = currentMillis2;

    UV_WindDir();
  }


  //Uncomment to see the values through the serial port
  //unsigned long currentTime2 = millis();
  //if (currentTime2 - lastWindCheck2 >= 2000) {
  //  lastWindCheck2 = currentTime2;
  //  FunPrint();
  //}
```

```
  // Get current timestamp
  currentTimeSave = millis();

  // Check if 5 minutes have passed
  if (currentTimeSave % (5 * 60 * 1000) == 0) {

    SaveFun();
  }

}

void getRainfall(){
  int tips = numTipsRain;
  numTipsRain = 0;

  float rainfall = tips * 0.2794;
  totalRain += rainfall;
}

void read_Temp_Humi(){

  humidity = dht.readHumidity();    // Read humidity value
  temperature = dht.readTemperature(); // Read temperature value in Celsius

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  heatIndex = calculateHeatIndex(temperature, humidity);
}

float calculateHeatIndex(float temperature, float humidity) {
  // Heat index calculation (Steadman's formula)
  float hi = 0.5 * (temperature + 61.0 + ((temperature - 68.0) * 1.2) +
(humidity * 0.094));

  if (hi > temperature) {
    return hi;
  } else {
    return temperature;
  }
}

void UV_WindDir(){

  int16_t UVSensor, WindDir;

  UVSensor = ads.readADC_SingleEnded(0);
  // Map the analog value to a scale from 0 to 11
```

```
  uvPercentage = map(UVSensor, 0, 25132, 0, 11);

  WindDir = ads.readADC_SingleEnded(1);
  ValuemV = ads.readADC_SingleEnded(1);
  if(WindDir >= (4960-600) && WindDir <= (4960+600)){
    strcpy(WinDirVariable, "W"); //W
  }
  else if(WindDir >= (10860-600) && WindDir <= (10860+600)){
    strcpy(WinDirVariable, "NW"); //NW
  }
  else if(WindDir >= (16693-550) && WindDir <= (16693+550)){
    strcpy(WinDirVariable, "N"); //N
  }
  else if(WindDir >= (15686-350) && WindDir <= (15686+350)){
    strcpy(WinDirVariable, "NE"); //NE
  }
  else if(WindDir >= (13590-600) && WindDir <= (13590+600)){
    strcpy(WinDirVariable, "E"); //E
  }
  else if(WindDir >= (7960-600) && WindDir <= (7960+600)){
    strcpy(WinDirVariable, "SE"); //SE
  }
  else if(WindDir >= (1600-600) && WindDir <= (1600+600)){
    strcpy(WinDirVariable, "S"); //S
  }
  else if(WindDir >= (3170-600) && WindDir <= (3170+600)){
    strcpy(WinDirVariable, "SW"); //SW
  }

}

void FunPrint(){

  //Imprimir Relativity Tem and Hum
  Serial.print("Relative Temperature: "); Serial.print(temperature);
Serial.println("°C");
  Serial.print("Relative Humidity: "); Serial.print(humidity);
Serial.println("RH");

  //Total RainFall
  Serial.print("Total Rain Fall: "); Serial.print(totalRain);
Serial.println("mm");

  //Wind speed
  Serial.print("Wind Speed: "); Serial.print(windSpeed);
Serial.println("km/h");

  //Wind direction
  Serial.print("Wind Dir: "); Serial.print(WinDirVariable);
Serial.println("");
```

```
  Serial.print("UV Intensity: "); Serial.print(uvPercentage);
Serial.print("%"); Serial.println();

}

void SaveFun(){

  dataFile = SD.open("Weather_Station_sensors_data.csv", FILE_WRITE);

  if (dataFile) {
      // Write timestamp and sensor value to the file
      dataFile.print(currentTimeSave);
      dataFile.print(", ");
      dataFile.print(temperature);
      dataFile.print(", ");
      dataFile.print(humidity);
      dataFile.print(", ");
      dataFile.print(heatIndex);
      dataFile.print(", ");
      dataFile.print(totalRain);
      dataFile.print(", ");
      dataFile.print(windSpeed);
      dataFile.print(", ");
      dataFile.print(WinDirVariable);
      dataFile.print(", ");
      dataFile.print(uvPercentage);
      dataFile.println("");  // Add a newline character at the end
      dataFile.close();

      Serial.println("Sensor value saved to file.");
    } else {
      Serial.println("Error opening data file.");
    }
}
```

You can freely download and use the code here –> ultimateprogram_weatherstation.rar

Conclusion

The “Portable Weather Station” is a versatile and portable weather monitoring system that collects and displays essential weather data. The collected data is easily accessible through a web interface, and it also logs the information to an SD card for further analysis and tracking. The station can be used for various applications, including home weather monitoring, agriculture, and educational purposes.