```
# WSMainSend.py
# Ben Arnett
# 05/12/2025

import time
import board
import busio
import analogio
import adafruit_bme680
import digitalio
from digitalio import DigitalInOut, Direction, Pull
from adafruit_pm25.i2c import PM25_I2C


reset_pin = None

# Setup IO pin to turn on and off PMSA03i
pwr = digitalio.DigitalInOut(board.GP15)
pwr.direction = digitalio.Direction.OUTPUT
pwr.value = True


# Create library object, use 'slow' 100KHz frequency!
i2c = busio.I2C(board.GP9, board.GP8,
frequency=100000)

# Connect to a PMSA003i over I2C
pms = PM25_I2C(i2c, reset_pin)
print("Found PMSA003i")

# Connect to BME688 over I2C
# note: BME688 uses the bme680 library
sensor =
adafruit_bme680.Adafruit_BME680_I2C(i2c)
sensor.seaLevelhPa = 1011
print("Found BME688")

# Init UART bus for RYLR993 lite
uart = busio.UART(board.GP0, board.GP1,
baudrate=9600)
print("UART bus enabled")

# PhotoTransistor setup
pt = analogio.AnalogIn(board.GP26)
print("Photo Transistor Found")

# Address of Rylr module to send to
address = 6

# Function to send LoRa messages with the RYLR
def rylr_send(message):
    length = len(message)
    send = 'AT+SEND={},{},{}\r\n'.format(address,
length, message)
    uart.write(send.encode("ascii"))
    print('Data sent to {}'.format(address))

# Function to apply AQI correction calculations
def correct_pm25(pm25, humidity, temperature):
    # correction calculations from ChatGPT
    # Humidity correction (EPA formula)
    humidity_correction = 1 + 0.487 * (2.718 ** (0.059
* humidity))
    pm25_corrected = pm25 / humidity_correction

    # Temperature correction
    temp_correction_factor = 1 - 0.02 * (temperature -
25)
    pm25_corrected *= temp_correction_factor

    return pm25_corrected

# Get Light sensor voltage
def ptvoltage(adcin):
    steps = 0
    i = 0
    # get average over 4 seconds
    while i < 20:
        steps += adcin.value
        time.sleep(0.2)
        i += 1

    steps = steps/20

    # Pico ADC is 12 bits, so need to scale to 16 bits
to work with CircuitPython's API
    return (steps * 3.3) / 65536

# Main Loop
while True:
    # Turn on PMSA, give 30s to warmup
    pwr.value = True
    print("Turned on PMSA, 30s warmup")
    time.sleep(30)
    print("Reading Sensors")

    try:
        aqdata = pms.read()
        # print(aqdata)
    except RuntimeError:
        print("Unable to read from sensor, retrying...")
        continue

    # get raw PM 2.5 concentration and apply
corrections

    pms_data = pms.read()
    pm25_raw = pms_data["pm25 standard"]
    pm25_corrected = correct_pm25(pm25_raw,
humidity, temperature)

    # get BME688 sensor data
    humidity = sensor.relative_humidity
    temperature = sensor.temperature
    pressure = sensor.pressure
    altitude = sensor.altitude

    # get PhotoTransistor Voltage
    ptvolt = ptvoltage(pt)

    # Apply significant figures
    pm25_corrected = round(pm25_corrected)
    humidity = "{:2.1f}".format(humidity)
    temperature = round(temperature) # 3 Digits
    pressure = round(pressure) #
    altitude = round(altitude)
    ptvolt = "{:1.2f}".format(ptvolt) # 3 dig. 4 char

    # format to be parse-able
    rylrmsg =
'AQI:{0};T:{1};RH:{2};hPa:{3};Alt:{4};L:{5}'.format(pm2
5_corrected, temperature, humidity, pressure,
altitude, ptvolt)
    rylr_send(rylrmsg)
    print(rylrmsg)

    # Turn off PMSA
    pwr.value = False
    print("Turned off PMSA, light sleeping")
    # Light sleep till next reading
    time.sleep(866) # Sleep 14m 26s (15 mins
between data transmissions)
                              # ^ account for 30s PMSA on time
& 4s light volt reading
```

```
# ModelComplete.py
# Ben Arnett
# 05/12/2025

# UART = GP 0,1
# I2C = SDA-GP8, SCL-GP9
# Led PWM = GP5
# Servo PWM = GP28

import board
import busio
import digitalio
import time
import pwmio
from lcd1602 import LCD1602
from adafruit_motor import servo

# Initialize Devices
rylr = busio.UART(board.GP0, board.GP1,
baudrate=9600)
i2c0 = busio.I2C(board.GP9, board.GP8,
frequency=100000)

lcd = LCD1602(i2c0)
lcd.clear()
lcd.write('  Initializing')

whtled = pwmio.PWMOut(board.GP3,
frequency=5000, duty_cycle=0)
servo_a_pin = pwmio.PWMOut(board.GP28,
frequency=50)
servo_a = servo.Servo(servo_a_pin,
min_pulse=1000, max_pulse=2000)

# 'Breathe' leds once, modulate servo to verify
function
for i in range(100):
        if i < 50:
            whtled.duty_cycle = int(i * 2 * 65535 / 100)  #
Up
        else:
            whtled.duty_cycle = 65535 - int((i - 50) * 2 *
65535 / 100)  # Down
        time.sleep(0.025)

whtled.duty_cycle = 0

servo_a.angle = 0
time.sleep(0.5)
servo_a.angle = 20
time.sleep(0.5)
servo_a.angle = 40
time.sleep(0.5)
servo_a.angle = 60
time.sleep(0.5)
servo_a.angle = 70
time.sleep(0.5)

lcd.clear()
lcd.set_cursor(2, 0)
lcd.write('Waiting for')
lcd.set_cursor(2, 1)
lcd.write('Weather Data')
print("Init complete, waiting for data")

recieved = False

# Function to parse new recieved data
# Modified for circuitpython from
https://github.com/TimHanewich/MicroPython-Collec
tion/blob/master/REYAX-RYLR998/
def parse_sensor_data(data: str) -> dict:
    # Extract the weather data from entire recieved
byte data
    result = {}
    address:int = None # the address of the
transmitter it came from
    length:int = None # the length (number of bytes)
of the data payload
    msg:bytes = None # the payload data itself
    RSSI:int = None # Received signal strength
indicator
    SNR:int = None # Signal-to-noise ratio

    try:
            # find landmarkers that will help with parsing
            i_equal:int = data.find("=")
            i_comma1:int = data.find(",")
            i_comma2:int = data.find(",", i_comma1 + 1)
            i_comma4:int = data.rfind(",") # search from
end
            i_comma3:int = data.rfind(",", 0,
i_comma4-1) # search for a comma from right,
starting at 0 and ending at the last comma (or right
before it)
            # i_linebreak:int = data.find("\r\n")

            # extract
            ReceivedMessage = data
            address = int(data[i_equal + 1:i_comma1])
            length = int(data[i_comma1 + 1:i_comma2])
            msg = data[i_comma2 + 1:i_comma3]
            RSSI = int(data[i_comma3 + 1:i_comma4])
            #SNR = int(data[i_comma4 + 1:i_linebreak])
    except Exception as e:
            raise Exception("Unable to parse line '" +
str(data) + "' as a ReceivedMessage! Exception
message: " + str(e))

    # Take the weather data string and turn into a
dictionary
    # Modified from ChatGPT generated code
    pairs = msg.split(";")  # Split the string into
key-value pairs
    for pair in pairs:
        if ":" in pair:
            key, value = pair.split(":")  # Separate key
and value
            # Try to convert value to int or float if
possible
            if value.replace(".", "", 1).isdigit():
                value = float(value) if "." in value else
int(value)
            result[key] = value
    return result # returns a dictionary

# Function to change interior conditions
def interior(d):
    light = d['L']
    # If its pretty bright, open blinds and turn off lights
    if light > 3:
        whtled.duty_cycle = 0
        servo_a.angle = 15
        return

    involt = 3 - light
    lightduty = round(involt * 18000) # inverted scaling
    servangle = round(18 * involt) + 15 # regular
scaling

    print('lightduty = {0} servangle =
{1}'.format(lightduty, servangle))

    whtled.duty_cycle = lightduty
    servo_a.angle = servangle
    return
```

```
# Function to update stats on display
def lcddisplay(d):
    if recieved == False:
        lcd.clear()
        lcd.write('T:  C Light:   %')
        lcd.set_cursor(0, 1)
        lcd.write('AQI:    RH:   %')

    lightperc = round(d['L'] / 3.2 * 100)

    lcd.set_cursor(2, 0)
    if d['T'] < 10:
        lcd.write(" ")
    lcd.write(str(d['T']))
    lcd.set_cursor(12, 0)
    lcd.write(lcdform(lightperc))
    lcd.set_cursor(4, 1)
    lcd.write(lcdform(d['AQI']))
    lcd.set_cursor(11, 1)
    lcd.write(str(lcdform(round(d['RH']))))
    return
    # T:00C Light:00%
    # _AQI:000 RH:00%

# Format data to write properly
def lcdform(number):
    stringout = ""
    if number < 10:
        stringout = stringout + "  "
    elif number < 100:
        stringout = stringout + " "
    stringout = stringout + str(number)
    return stringout

# Main Loop
while True:
    data = rylr.read()

    if data is not None:
        # Make a string with incoming data, then put it
through the parser
        data_string = ''.join([chr(b) for b in data])
        print(data_string, end="")
        parsed = parse_sensor_data(data_string)
        print(parsed)

        # Update display and Interior settings
        lcddisplay(parsed)
        interior(parsed)
        recieved = True

    else:
        if recieved == False:
            time.sleep(0.05)
```

Test Codes:
Reyax RYLR Lora Module Tests:

```
# RYLR Receive test
import board
import busio
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

uart = busio.UART(board.GP0, board.GP1,
baudrate=9600)

while True:
    data = uart.read()

    if data is not None:
        led.value = True
        data_string = ''.join([chr(b) for b in data])
        print(data_string, end="")

        led.value = False

    else:

        print("waiting")
        time.sleep(0.05)
```

```
# RYLR send test

import board
import busio
import digitalio
import time

uart = busio.UART(board.GP0, board.GP1,
baudrate=9600)
test = 0
send = "AT+SEND=6,4,test\r\n"

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
ledState = False

while True:
    led.value = True
    uart.write(send.encode("ascii"))
    print("sent")
    time.sleep(2.5)
    led.value = False
    time.sleep(2.5)
```

```
# Rylr993RangeTest.py
# Ben Arnett
# 03/04/25

import time
import board
import busio
import digitalio

# Only use with Rylr993 lite, not 998
rylr = busio.UART(board.GP0, board.GP1,
baudrate=9600)

# address of Rylr on network to send to
addr = 27

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

i = 1

while True:
    led.value = True
    msg = "Test " + str(i)
    length = len(msg)
    send = 'AT+SEND={},{},{}\r\n'.format(addr,
length, msg)

    rylr.write(send.encode("ascii"))
    print("Just sent test " + str(i))
    led.value = False
    time.sleep(5)
    i += 1
```

Weather Station Test code

# BME688 Test from Adafruit

```
import board
import busio
import adafruit_bme680
import time

i2c = busio.I2C(board.GP9, board.GP8)
sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)

sensor.seaLevelhPa = 1011

while True:

   print('Temperature: {} degrees C'.format(sensor.temperature))
   print('Gas: {} ohms'.format(sensor.gas))
   print('Humidity: {}%'.format(sensor.humidity))
   print('Pressure: {}hPa'.format(sensor.pressure))
   print('Altitude: {} meters'.format(sensor.altitude))
   print("---------------------------------")
   time.sleep(1)
```

```
# PMSA003i test from Adafruit
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
Example sketch to connect to PM2.5 sensor with either I2C or UART.
"""

# pylint: disable=unused-import
import time
import board
import busio
import adafruit_bme680
from digitalio import DigitalInOut, Direction, Pull
from adafruit_pm25.i2c import PM25_I2C

reset_pin = None
# If you have a GPIO, its not a bad idea to connect it to the RESET pin
# reset_pin = DigitalInOut(board.G0)
# reset_pin.direction = Direction.OUTPUT
# reset_pin.value = False


# For use with a computer running Windows:
# import serial
# uart = serial.Serial("COM30", baudrate=9600, timeout=1)

# For use with microcontroller board:
# (Connect the sensor TX pin to the board/computer RX pin)
# uart = busio.UART(board.TX, board.RX, baudrate=9600)

# For use with Raspberry Pi/Linux:
# import serial
# uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=0.25)

# For use with USB-to-serial cable:
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=0.25)

# Connect to a PM2.5 sensor over UART
# from adafruit_pm25.uart import PM25_UART
# pm25 = PM25_UART(uart, reset_pin)

# Create library object, use 'slow' 100KHz frequency!
i2c = busio.I2C(board.GP9, board.GP8, frequency=100000)
# Connect to a PM2.5 sensor over I2C
pm25 = PM25_I2C(i2c, reset_pin)

sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)

sensor.seaLevelhPa = 1011

print("Found PM2.5 sensor, reading data...")
```

```
while True:
    time.sleep(1)

    try:
        aqdata = pm25.read()
        # print(aqdata)
    except RuntimeError:
        print("Unable to read from sensor, retrying...")
        continue

    print()
    print("Concentration Units (standard)")
    print("--------------------------------------")
    print(
        "PM 1.0: %d\tPM2.5: %d\tPM10: %d"
        % (aqdata["pm10 standard"], aqdata["pm25 standard"], aqdata["pm100 standard"])
    )
    print("Concentration Units (environmental)")
    print("--------------------------------------")
    print(
        "PM 1.0: %d\tPM2.5: %d\tPM10: %d"
        % (aqdata["pm10 env"], aqdata["pm25 env"], aqdata["pm100 env"])
    )
    print("--------------------------------------")
    print("Particles > 0.3um / 0.1L air:", aqdata["particles 03um"])
    print("Particles > 0.5um / 0.1L air:", aqdata["particles 05um"])
    print("Particles > 1.0um / 0.1L air:", aqdata["particles 10um"])
    print("Particles > 2.5um / 0.1L air:", aqdata["particles 25um"])
    print("Particles > 5.0um / 0.1L air:", aqdata["particles 50um"])
    print("Particles > 10 um / 0.1L air:", aqdata["particles 100um"])
    print("--------------------------------------")

    print('Temperature: {} degrees C'.format(sensor.temperature))
    print('Gas: {} ohms'.format(sensor.gas))
    print('Humidity: {}%'.format(sensor.humidity))
    print('Pressure: {}hPa'.format(sensor.pressure))
    print('Altitude: {} meters'.format(sensor.altitude))
    print("--------------------------------")
```

# AQI with BME688 based correction factors

```
# From ChatGPT, with adjustments made by Ben A.

import time
import board
import busio
import adafruit_bme680  # Works for BME688 too
import adafruit_pm25
from digitalio import DigitalInOut, Direction, Pull
from adafruit_pm25.i2c import PM25_I2C

# Initialize I2C communication and the PMSA003I sensor
i2c = busio.I2C(board.GP9, board.GP8, frequency=100000)
# Connect to a PM2.5 sensor over I2C
pms = PM25_I2C(i2c)

# Initialize BME688 sensor
bme = adafruit_bme680.Adafruit_BME680_I2C(i2c)

def correct_pm25(pm25, humidity, temperature):
    """
    Applies correction factors to PM2.5 based on humidity and temperature.

    Args:
        pm25 (float): Raw PM2.5 measurement (µg/m³)
        humidity (float): Relative humidity (%)
        temperature (float): Temperature (°C)

    Returns:
        float: Corrected PM2.5 value
    """
    # Humidity correction (EPA formula)
    humidity_correction = 1 + 0.487 * (2.718 ** (0.059 * humidity))
    pm25_corrected = pm25 / humidity_correction

    # Temperature correction
    temp_correction_factor = 1 - 0.02 * (temperature - 25)
    pm25_corrected *= temp_correction_factor

    return pm25_corrected

while True:
    # Read sensor data
    pms_data = pms.read()
    pm25_raw = pms_data["pm25 standard"]

    humidity = bme.relative_humidity
    temperature = bme.temperature

    # Apply correction
    pm25_corrected = correct_pm25(pm25_raw, humidity, temperature)

    # Print results
    print(f"PM2.5 Raw: {pm25_raw:.2f} µg/m³")
    print(f"PM2.5 Corrected: {pm25_corrected:.2f} µg/m³")
    print(f"Temperature: {temperature:.2f} °C, Humidity: {humidity:.2f} %")
    print("-" * 40)

    time.sleep(10)
```

# AQI with BME688 based correction factors

```
# From ChatGPT, with adjustments made by Ben A.

import time
import board
import busio
import adafruit_bme680  # Works for BME688 too
import adafruit_pm25
from digitalio import DigitalInOut, Direction, Pull
from adafruit_pm25.i2c import PM25_I2C

# Initialize I2C communication and the PMSA003I sensor
i2c = busio.I2C(board.GP9, board.GP8, frequency=100000)
# Connect to a PM2.5 sensor over I2C
pms = PM25_I2C(i2c)

# Initialize BME688 sensor
bme = adafruit_bme680.Adafruit_BME680_I2C(i2c)

def correct_pm25(pm25, humidity, temperature):
    """
    Applies correction factors to PM2.5 based on humidity and temperature.

    Args:
        pm25 (float): Raw PM2.5 measurement (µg/m³)
        humidity (float): Relative humidity (%)
        temperature (float): Temperature (°C)

    Returns:
        float: Corrected PM2.5 value
    """
    # Humidity correction (EPA formula)
    humidity_correction = 1 + 0.487 * (2.718 ** (0.059 * humidity))
    pm25_corrected = pm25 / humidity_correction

    # Temperature correction
    temp_correction_factor = 1 - 0.02 * (temperature - 25)
    pm25_corrected *= temp_correction_factor

    return pm25_corrected

while True:
    # Read sensor data
    pms_data = pms.read()
    pm25_raw = pms_data["pm25 standard"]

    humidity = bme.relative_humidity
    temperature = bme.temperature
    # Apply correction
    pm25_corrected = correct_pm25(pm25_raw, humidity, temperature)

    # Print results
    print(f"PM2.5 Raw: {pm25_raw:.2f} µg/m³")
    print(f"PM2.5 Corrected: {pm25_corrected:.2f} µg/m³")
    print(f"Temperature: {temperature:.2f} °C, Humidity: {humidity:.2f} %")
    print("-" * 40)

    time.sleep(10)
```

```
# ADC test for photo transistor
# BA 03/11/25

import time
import busio
import board
import analogio

ptr = analogio.AnalogIn(board.GP26)

# Pico ADC is 12 bits, so need to scale to 16 bits to work with CircuitPython's API

def stepstovolt(step):
    return (step * 3.3) / 65536
#                             2^16

while True:
    steps = ptr.value
    voltage = stepstovolt(steps)
    print('Steps: {} Voltage: {:1.3}'.format(steps, voltage))
    time.sleep(0.5)
```

Smart Home Test Code

```
# Servo Test

# from Adafruit

from time import sleep
import board
import pwmio
from adafruit_motor import servo

DEBUG = True

servo_a_pin = pwmio.PWMOut(board.GP18, frequency=50)
servo_a = servo.Servo(servo_a_pin, min_pulse=1000, max_pulse=2000)
def basic_operations():
    if DEBUG: print("Setting angle to 90 degrees.")
    servo_a.angle = 90
    sleep(3)
    if DEBUG: print("Setting angle to 0 degrees.")
    servo_a.angle = 0
    sleep(3)
    if DEBUG: print("Setting angle to 90 degrees.")
    servo_a.angle = 90
    sleep(3)
    if DEBUG: print("Setting angle to 180 degrees.")
    servo_a.angle = 180
    sleep(3)
while True:
    basic_operations()
```

```
# PWM Led Test
# Ben Arnett
# 04/10/2025

import time
import board
import pwmio

redled = pwmio.PWMOut(board.GP3, frequency=5000, duty_cycle=0)
#grnled = pwmio.PWMOut(board.GP17, frequency=5000, duty_cycle=0)

while True:
    for i in range(100):
        if i < 50:
            redled.duty_cycle = int(i * 2 * 65535 / 100)  # Up
            #grnled.duty_cycle = 65535 - int(i * 2 *
65535 /100) # Down
        else:
            redled.duty_cycle = 65535 - int((i - 50) *
2 * 65535 / 100)  # Down
            #grnled.duty_cycle = int((i-50) * 2 *
65535 / 100) # Up
        time.sleep(0.05)
```

# Receive, parse data test

```
import board
import busio
import digitalio
import time
from lcd1602 import LCD1602

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

uart = busio.UART(board.GP0, board.GP1,
baudrate=9600)
i2c = busio.I2C(board.GP9, board.GP8,
frequency=100000)
lcd = LCD1602(i2c)

#  Modified for circuitpython from
https://github.com/TimHanewich/MicroPython-C
ollection/blob/master/REYAX-RYLR998/

def parse_sensor_data(data: str) -> dict:
    result = {}
    address:int = None # the address of the
transmitter it came from
    length:int = None # the length (number of
bytes) of the data payload
    msg:bytes = None # the payload data itself
    RSSI:int = None # Received signal strength
indicator
    SNR:int = None # Signal-to-noise ratio

    try:

            # find landmarkers that will help with
parsing
            i_equal:int = data.find("=")
            i_comma1:int = data.find(",")
            i_comma2:int = data.find(",", i_comma1
+ 1)
            i_comma4:int = data.rfind(",") # search
from end
            i_comma3:int = data.rfind(",", 0,
i_comma4-1) # search for a comma from right,
starting at 0 and ending at the last comma (or
right before it)
           # i_linebreak:int = data.find("\r\n")

            # extract
            ReceivedMessage = data
            address = int(data[i_equal +
1:i_comma1])
            length = int(data[i_comma1 +
1:i_comma2])
            msg = data[i_comma2 + 1:i_comma3]
            RSSI = int(data[i_comma3 +
1:i_comma4])
            #SNR = int(data[i_comma4 +
1:i_linebreak])
    except Exception as e:
            raise Exception("Unable to parse line '"
+ str(data) + "' as a ReceivedMessage!
Exception message: " + str(e))

# String parsing modified from ChatGPT code

    pairs = msg.split(";")  # Split the string into
key-value pairs
    for pair in pairs:
        if ":" in pair:
            key, value = pair.split(":")  # Separate
key and value
            # Try to convert value to int or float if
possible
            if value.replace(".", "", 1).isdigit():
                value = float(value) if "." in value else
int(value)
            result[key] = value
    return result

# Example usage
#data_string = "Led=1;Temp=77;Light=0.85"
#parsed_data =
parse_sensor_data(data_string)
#print(parsed_data)
```

```
i = 0
recieved = False

while True:
    data = uart.read()

    if data is not None:
        led.value = True
        data_string = ''.join([chr(b) for b in data])
        print(data_string, end="")
        parsed = parse_sensor_data(data_string)
        print(parsed)
        lcd.write("Recieved! \n #: " + str(i))
        led.value = False
        recived = True
        i += 1

    else:
        if recieved == False:
            lcd.clear()
            lcd.write("Waiting...")
            print("waiting")
            time.sleep(0.05)

        else:
            time.sleep(0.05)
```

Website host code

- **Written primarily by ChatGPT, with formatting and order changes by me**

```
// WSmDNSGraph
// Ben Arnett
// 05/18/2025

// PRIMARILY WRITTEN BY CHATGPT
// small syntax changes and formatting by B. Arnett

#include <WiFi.h>
#include <time.h>
#include <WebServer.h>
#include "secrets.h"
// Add a secrets.h file to store WiFi login
// otherwise edit params. under setup() > WiFi.begin()
#include <DNSServer.h>
#include <ESPmDNS.h>
#include <Ticker.h>

#define LORA_RX 17
#define LORA_TX 16

String loraBuffer = "";
int lastRSSI = -100;

#define MAX_RECORDS 60  // assuming ~1 data point every 30 seconds

struct WeatherData {
  int aqi;
  float temperature;
  float humidity;
  int pressure;
  int altitude;
  float light;
  float lightperc;
  String timestamp;
  time_t epoch;
  int rssi;
};

String latestData = "";
float latestTemp = 0.0;
const int maxDataPoints = 20;
float temperatureData[maxDataPoints];
int tempIndex = 0;

String getContentType(String filename) {
  if (filename.endsWith(".htm") || filename.endsWith(".html")) return "text/html";
  else if (filename.endsWith(".css")) return "text/css";
  else if (filename.endsWith(".js")) return "application/javascript";
  return "text/plain";
}

WeatherData dataBuffer[MAX_RECORDS];
int bufferSize = 0;
WebServer server(80);
bool isCaptivePortal = false;

DNSServer dnsServer;
const byte DNS_PORT = 53;
const char* localHostname = "loraweather";


String getLocalTimeString() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    return "TIME ERROR";
  }
  char timeStr[20];
  strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &timeinfo);
  return String(timeStr);
}

void readLoRaData() {
  while (Serial1.available()) {
    char c = Serial1.read();
    if (c == '\n') {
      loraBuffer.trim();
      if (loraBuffer.length() > 0) {
        parseLoRaMessage(loraBuffer);
      }
      loraBuffer = "";
    } else {
      loraBuffer += c;
    }
  }
}

void parseLoRaMessage(String message) {
  if (message.startsWith("+RCV=")) {
    message = message.substring(5);
    int comma1 = message.indexOf(',');
    int comma2 = message.indexOf(',', comma1
+ 1);
    int comma3 = message.indexOf(',', comma2
+ 1);
    int comma4 = message.indexOf(',', comma3
+ 1);

    String sender = message.substring(0,
comma1);
    String length = message.substring(comma1
+ 1, comma2);
    String payload = message.substring(comma2
+ 1, comma3);
    String SNR = message.substring(comma3 +
1, comma4);
    String RSSI = message.substring(comma4 +
1);

    lastRSSI = RSSI.toInt();

    Serial.println("=== LoRa Message Received
===");
    Serial.println("From: " + sender);
    Serial.println("Length: " + length);
    Serial.println("Payload: " + payload);
    Serial.println("SNR: " + SNR);
    Serial.println("RSSI: " + RSSI);

Serial.println("========================
=====");

    if (payload.startsWith("AQI:")) {
      String timestamp = getLocalTimeString();
      WeatherData w =
parseWeatherString(payload, timestamp);
      w.rssi = RSSI.toInt();

      struct tm timeinfo;
      getLocalTime(&timeinfo);
      w.epoch = mktime(&timeinfo);

      temperatureData[tempIndex] =
w.temperature;
      tempIndex = (tempIndex + 1) %
maxDataPoints;

      if (bufferSize < MAX_RECORDS) {
        dataBuffer[bufferSize++] = w;
      } else {
        for (int i = 1; i < MAX_RECORDS; i++) {
          dataBuffer[i - 1] = dataBuffer[i];
        }
        dataBuffer[MAX_RECORDS - 1] = w;
      }

      Serial.println("---- Weather Data ----");
      Serial.println("Time: " + w.timestamp);
      Serial.print("AQI: "); Serial.println(w.aqi);
      Serial.print("Temp: ");
Serial.println(w.temperature);
      Serial.print("Humidity: ");
Serial.println(w.humidity);
      Serial.print("Pressure: ");
Serial.println(w.pressure);
      Serial.print("Altitude: ");
Serial.println(w.altitude);
      Serial.print("Light Level: ");
Serial.println(w.light);

      printHourlyAverages();
    }
  } else {
    Serial.println("Other LoRa Response: " +
message);
  }
}

WeatherData parseWeatherString(String input,
String timestamp) {
  WeatherData data;
  input.trim();
  data.timestamp = timestamp;

  int start = 0;
  while (start < input.length()) {
    int sep = input.indexOf(':', start);
    int end = input.indexOf(';', sep);
    if (end == -1) end = input.length();

    String key = input.substring(start, sep);
    String value = input.substring(sep + 1, end);

    if (key == "AQI") data.aqi = value.toInt();
    else if (key == "T") data.temperature =
value.toFloat();
    else if (key == "RH") data.humidity =
value.toFloat();
    else if (key == "hPa") data.pressure =
value.toInt();
    else if (key == "Alt") data.altitude =
value.toInt();
    else if (key == "L") data.light =
value.toFloat();

    data.lightperc = data.light * 30.77;
    start = end + 1;
  }

  return data;
}

String getAveragesHTML() {
  time_t now;
  struct tm timeinfo;
  getLocalTime(&timeinfo);
  now = mktime(&timeinfo);

  float sumAQI = 0, sumTemp = 0, sumHum = 0,
sumPress = 0, sumAlt = 0, sumLight = 0,
sumLightperc = 0;
  int count = 0;

  for (int i = 0; i < bufferSize; i++) {
    if (difftime(now, dataBuffer[i].epoch) <= 3600)
{
      sumAQI += dataBuffer[i].aqi;
      sumTemp += dataBuffer[i].temperature;
      sumHum += dataBuffer[i].humidity;
      sumPress += dataBuffer[i].pressure;
      sumAlt += dataBuffer[i].altitude;
      sumLight += dataBuffer[i].light;
      sumLightperc += dataBuffer[i].lightperc;
      count++;
    }
  }

  if (count == 0) return "<p>No recent data to
average.</p>";

  String html =
"<table><tr><th>Metric</th><th>Average</th><
/tr>";
  html += "<tr><td>AQI</td><td>" +
String(sumAQI / count, 1) + "</td></tr>";
  html += "<tr><td>Temperature (°C)</td><td>"
+ String(sumTemp / count, 1) + "</td></tr>";
  html += "<tr><td>Humidity (%)</td><td>" +
String(sumHum / count, 1) + "</td></tr>";
  html += "<tr><td>Pressure (hPa)</td><td>" +
String(sumPress / count, 1) + "</td></tr>";
  html += "<tr><td>Altitude (m)</td><td>" +
String(sumAlt / count, 1) + "</td></tr>";
  html += "<tr><td>Light (V)</td><td>" +
String(sumLight / count, 2) + "</td></tr>";
  html += "<tr><td>Light (%)</td><td>" +
String(sumLightperc / count, 1) + "</td></tr>";
  html += "</table>";

  return html;
}

void printHourlyAverages() {
  time_t now;
  struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    now = mktime(&timeinfo);
  } else {
    Serial.println("Failed to get time");
    return;
  }

  float sumAQI = 0, sumTemp = 0, sumHum = 0,
sumPress = 0, sumAlt = 0, sumLight = 0,
sumLightperc = 0;
  int count = 0;

  for (int i = 0; i < bufferSize; i++) {
    if (difftime(now, dataBuffer[i].epoch) <= 3600)
{
      sumAQI += dataBuffer[i].aqi;
      sumTemp += dataBuffer[i].temperature;
      sumHum += dataBuffer[i].humidity;
      sumPress += dataBuffer[i].pressure;
      sumAlt += dataBuffer[i].altitude;
      sumLight += dataBuffer[i].light;
      sumLightperc += dataBuffer[i].lightperc;
      count++;
    }
  }

  if (count == 0) {
    Serial.println("No recent data to average.");
    return;
  }

  Serial.println("---- Hourly Averages ----");
  Serial.printf("Avg AQI: %.1f\n", sumAQI /
count);
  Serial.printf("Avg Temp (°C): %.1f\n",
sumTemp / count);
  Serial.printf("Avg RH (%%): %.1f\n", sumHum
/ count);
  Serial.printf("Avg Pressure (hPa): %.1f\n",
sumPress / count);
  Serial.printf("Avg Altitude (m): %.1f\n", sumAlt
/ count);
  Serial.printf("Avg Light (V): %.2f\n", sumLight /
count);
  Serial.printf("Avg Light (%): %.1f\n",
sumLightperc / count);
}

String getSignalStrengthBar(int rssi) {
  int percentage = constrain(map(rssi, -100, -40,
0, 100), 0, 100);
  String color = (rssi > -70) ? "green" : (rssi >
-90) ? "orange" : "red";
  String html = "<div style='border:1px solid
#ccc;width:100%;height:20px;'><div
style='height:100%;width:" + String(percentage)
+ "%;background-color:" + color +
"'></div></div>";
  return html;
}

void handleRoot() {
```

```
  String html = "<!DOCTYPE
html><html><head><meta charset='UTF-8'>";
  html += "<meta name='viewport'
content='width=device-width,
initial-scale=1.0'>";
  html += "<title>Weather Dashboard</title>";
  html +=
"<style>body{font-family:sans-serif;padding:2e
m;}table{border-collapse:collapse;}td,th{padding
:0.5em;border:1px solid #ccc;}</style>";
  html += "</head><body>";
  html += "<h1>Latest Weather Data</h1>";

  if (bufferSize > 0) {
    WeatherData latest = dataBuffer[bufferSize -
1];
    html +=
"<table><tr><th>Metric</th><th>Value</th></tr
>";
    html += "<tr><td>AQI</td><td>" +
String(latest.aqi) + "</td></tr>";
    html += "<tr><td>Temperature (°C)</td><td>"
+ String(latest.temperature) + "</td></tr>";
    html += "<tr><td>Humidity (%)</td><td>" +
String(latest.humidity) + "</td></tr>";
    html += "<tr><td>Pressure (hPa)</td><td>" +
String(latest.pressure) + "</td></tr>";
    html += "<tr><td>Altitude (m)</td><td>" +
String(latest.altitude) + "</td></tr>";
    html += "<tr><td>Light (V)</td><td>" +
String(latest.light) + "</td></tr>";
    html += "<tr><td>Light (%)</td><td>" +
String(latest.lightperc) + "</td></tr>";
    html += "<tr><td>LoRa RSSI
(dBm)</td><td>" + String(latest.rssi) +
"</td></tr>";
    html += "<tr><td>Timestamp</td><td>" +
latest.timestamp + "</td></tr>";
    html += "</table>";
  } else {
    html += "<p>No data available.</p>";
  }

  html += "<h2>LoRa Signal Strength</h2>";
  html += "<p>RSSI: " + String(lastRSSI) + "
dBm</p>";
  html += getSignalStrengthBar(lastRSSI);
  html += "<p><small><strong>-70 dBm or higher:</strong> Excellent | <strong>-90 dBm to -70 dBm:</strong> Okay | <strong>-100 dBm to -90 dBm:</strong> Weak</small></p>";

  html += "<h2>Hourly Averages</h2>";
  html += getAveragesHTML();
  html += "<p><em>Page refreshes every 60 seconds</em></p>";
  html += "<script>setTimeout(()=>location.reload(),60000);</script>";
  html += "</body></html>";

  String graphs = R"rawliteral(
      <title>Weather Dashboard</title>
      <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
      <style>
        body { font-family: sans-serif; text-align: center; margin: 20px; }
        canvas { max-width: 600px; margin: auto; }
      </style>
    </head>
    <body>
      <h1>LoRa Weather Dashboard</h1>
      <div id="dataDisplay">Loading latest data...</div>
      <canvas id="tempChart" width="600" height="300"></canvas>

      <script>
        let tempChart;

        async function fetchData() {
          const res = await fetch('/data');
          const json = await res.json();

          document.getElementById('dataDisplay').innerText = json.latest;

          if (!tempChart) {
            const ctx = document.getElementById('tempChart').getContext('2d');
            tempChart = new Chart(ctx, {
              type: 'line',
              data: {
                labels: json.history.map((_, i) => i + 1),
                datasets: [{
                  label: 'Temperature (°C)',
                  data: json.history,
                  borderColor: 'orange',
                  fill: false
                }]
              },
              options: {
                responsive: true,
                scales: {
                  y: { beginAtZero: true }
                }
              }
            });
          } else {
            tempChart.data.labels = json.history.map((_, i) => i + 1);
            tempChart.data.datasets[0].data = json.history;
            tempChart.update();
          }
        }

        setInterval(fetchData, 2000);
        fetchData();
      </script>
    </body>
    </html>
  )rawliteral";
  html += graphs;
  server.send(200, "text/html", html);
}

void handleDataJson() {
  String json = "{";
  json += "\"latest\":\"" + latestData + "\",";
  json += "\"history\":[";
  for (int i = 0; i < maxDataPoints; i++) {
    if (i > 0) json += ",";
    json += String(temperatureData[(tempIndex + i) % maxDataPoints]);
  }
  json += "]}";
  server.send(200, "application/json", json);
}



void setup() {
  Serial.begin(115200);
  Serial1.begin(115200, SERIAL_8N1,
LORA_RX, LORA_TX);
  Serial.println("LoRa Parser Ready");

  // Before WiFi.begin()
  WiFi.mode(WIFI_AP_STA);  // allow AP and
STA mode together
  WiFi.setHostname(localHostname);  // for
mDNS when connected to other networks

  // Start WiFi AP
  WiFi.softAP("WeatherNode", "weather123");

  // Start DNS redirect for captive portal
  dnsServer.start(DNS_PORT, "*",
WiFi.softAPIP());


  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to WiFi");
  unsigned long startAttempt = millis();
  while (WiFi.status() != WL_CONNECTED &&
millis() - startAttempt < 10000) {
    delay(500);
    Serial.print(".");
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi!");
    Serial.print("Web server IP address: ");
    Serial.println(WiFi.localIP());
    if (MDNS.begin(localHostname)) {
      Serial.println("mDNS responder started");
      Serial.print("You can access via http://");
      Serial.print(localHostname);
      Serial.println(".local");
    } else {
      Serial.println("Error setting up MDNS");
    }
    isCaptivePortal = false;
  } else {
    Serial.println("\nWiFi failed. Starting Access
Point...");
    isCaptivePortal = true;
    WiFi.softAP("WeatherNode", "weather1234");
    IPAddress AP_IP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(AP_IP);
  }

  configTzTime("PST8PDT,M3.2.0,M11.1.0",
"pool.ntp.org");

  server.on("/", handleRoot);
  server.on("/data", handleDataJson);
  server.begin();
  Serial.println("Web server started.");

  server.onNotFound([]() {
    server.sendHeader("Location", "/", true);
    server.send(302, "text/plain", "");
  });
}

void loop() {
  readLoRaData();

  server.handleClient();
  dnsServer.processNextRequest();
  if (Serial.available()) {
    Serial1.write(Serial.read());
  }
}
```