Professional ESP32 Weather Station

Non-blocking Architecture, WMO-compliant & Remote Management

This project implements an automated weather station based on the **ESP32**, designed for continuous, unattended operation. The software architecture is highly advanced: it utilizes cooperative state machines to avoid code blocking (non-blocking), ensuring that sensor reading, data transmission, and error management occur simultaneously without interruptions.

1. Supported Sensors and Hardware Management

The system supports an array of standard I2C and 1-Wire weather sensors:

- **Temperature and Humidity:** High-precision SHT31 sensor.
- **Atmospheric Pressure (& Backup Temp):** BMP280 sensor.
- **Outdoor/Soil Temperature:** DS18B20 1-Wire sensor.
- **Anemometer, Wind Vane, and Rain Gauge:** Managed via physical pins (Pins 26, 27, 34) with hardware interrupts (ISR) and software debouncing to prevent false readings.

> **Special Anti-Freeze/Condensation Function:** The code includes a control system for the SHT31. If the detected humidity remains fixed above 99% for a prolonged period (indicating condensation on the sensor), the system automatically activates the SHT31's internal heater for 5 seconds to dry it out.

2. Measurement Methodology (WMO Standards & Weather Calculations)

The software does not merely send raw data; it processes it according to rigorous meteorological standards:

- **WMO (World Meteorological Organization) Wind Standard:** A 10-minute (600 seconds) sliding window is maintained, capable of storing up to 700 samples.
- **Dominant Wind Direction Calculation:** The average wind direction is not a simple arithmetic mean. It is calculated by breaking vectors down into sine and cosine components, yielding the true dominant direction.
- **Gust Management:** Wind gusts are calculated as a 3-second moving average within the 10-minute window.
- **Physical Anomaly Filter:** The software discards impossible impulse peaks, such as an instantaneous wind increase of 40 mph or anomalous combined wind and rain impulses (often caused by electrical interference or pole shaking). The physical wind limit is capped at 106 mph.
- **Derived Data:** The microcontroller autonomously calculates *Dew Point*, *Wind Chill*, *Heat Index*, and *Relative Pressure (Sea Level Pressure)* compensated based on the configured altitude.
- **Rainfall:** Hourly and daily accumulations are automatically reset based on civilian time (NTP). Rain data is saved in the EEPROM memory to prevent loss during unexpected reboots.

3. Control Systems, Watchdog, and Error Recovery

Designed for remote sites, the system is "self-healing":

- **Watchdog Timer:** A ticker monitors the main loop to ensure it does not freeze for more than 120 seconds. If a freeze occurs, it saves the current state and forces a reboot.
- **Error Logging (LittleFS):** Reboot triggers (e.g., voltage drops, software crashes) and transmission errors are saved in text files (`reset_log.txt`, `error_log.txt`) within the ESP32's flash memory, enabling remote diagnostics.
- **Sensor Recovery:** If an I2C or 1-Wire sensor fails to read for a set number of attempts, the system tries a forced bus reinitialization (`begin()`) without rebooting the entire microcontroller.
- **Resilient Connectivity:** It supports multiple Wi-Fi networks (`WiFiMulti`) and constantly verifies actual internet access by pinging reliable DNS servers (1.1.1.1 and 8.8.8.8). It features status LEDs to indicate offline operation.

4. Multi-Platform Data Transmission

Using HTTP state machines (`HttpTxState`), the system simultaneously sends data (without halting sensor readings while waiting for server responses) to:

- **Weather Underground:** Updates every 30 seconds.
- **ThingSpeak:** Updates approximately every 3 minutes.
- **Weathercloud:** Updates every 10 minutes.

5. Telegram Notifications

The station acts as a Telegram bot to keep the administrator informed. Message dispatch is handled via a non-blocking Queue. The system notifies about:

- Station startup.
- Internet connection loss (logging the exact time of the outage).
- Internet connection restoration.
- Any sensor hardware errors.

6. Advanced Remote Management (Web Server)

The integrated local Web Server (listening on port `8080`) serves as the diagnostic dashboard and management interface for the station.

6.1 ElegantOTA: Over-The-Air Firmware Updates

ElegantOTA is a library that creates a web interface for uploading new code to the microcontroller via Wi-Fi, completely eliminating the need for a USB cable connection.

- After exporting the compiled binary (`.bin`) from the Arduino IDE or PlatformIO, you simply navigate to `http://<STATION_IP>:8080/update` .
- By dragging and dropping the `.bin` file into the elegant web interface, the ESP32 receives the new firmware, writes it to a secondary memory partition, automatically reboots, and runs the updated code.
- **Advantage:** You can push bug fixes or new features from the comfort of your computer without dismantling the physical installation.

6.2 Real-Time Status Monitoring

By accessing specific routes (e.g., `http://<STATION_IP>:8080/status`), the server returns a web page or JSON file showing the hardware's health:

- **Uptime:** How long the station has been running without interruptions, useful for identifying unexpected reboots.
- **Wi-Fi Signal (RSSI):** Indicates signal strength to diagnose potential cloud transmission delays.
- **Sensor Status:** Shows whether a sensor (e.g., SHT31 or BMP280) is "OK" or "ERROR/DISCONNECTED", helping to spot wiring issues.
- **Free Heap:** Monitors available memory to ensure the code has no memory leaks that could cause future crashes.

6.3 Log Management (Diagnostics and Cleaning)

Because the station uses a Watchdog for automatic reboots, event logs are crucial for understanding *why* a reboot occurred. These events are saved in the non-volatile flash memory using **LittleFS**.

- **Remote Viewing:** By visiting `http://<STATION_IP>:8080/logs`, the ESP32 reads `error_log.txt` and `reset_log.txt` and displays them in the browser. You might see messages like *"14:02 - I2C Error on SHT31 sensor"* or *"16:45 - Forced Watchdog reboot due to main loop block"*.
- **Log Cleaning:** Since the ESP32 flash memory is limited (typically ~4MB total), logs cannot grow indefinitely. A specific address (e.g., `/clearlogs`) can be triggered to empty the text files. The code handles this by opening the file in write mode (`"w"`) to truncate it to zero bytes, or by using `LittleFS.remove()`.
- **Security:** To prevent unauthorized users on the network from deleting your logs, this function can be protected using HTTP Basic Authentication on the Web Server.