Temperature Control System

Smart IoT solution for home / office automation.

Summary of the Project

An IoT-based smart room system built using the ESP32 microcontroller. This project enables real-time monitoring and control of room temperature and humidity via a mobile dashboard using MQTT protocol. It supports both manual and automatic fan control based on a user-defined temperature threshold.

Features

- Real-time Temperature & Humidity Monitoring (DHT11 sensor)
- Cooling Fan Control (Auto & Manual Modes)
- Mobile Dashboard Integration (MQTT-compatible apps like MQTT Dash or IoT MQTT Panel)
- Automatic operation based on a configurable temperature limit
- Communication via Wi-Fi and MQTT (using public broker)

01

Hardware Elements

Components List

List of the hardware components used in the system.

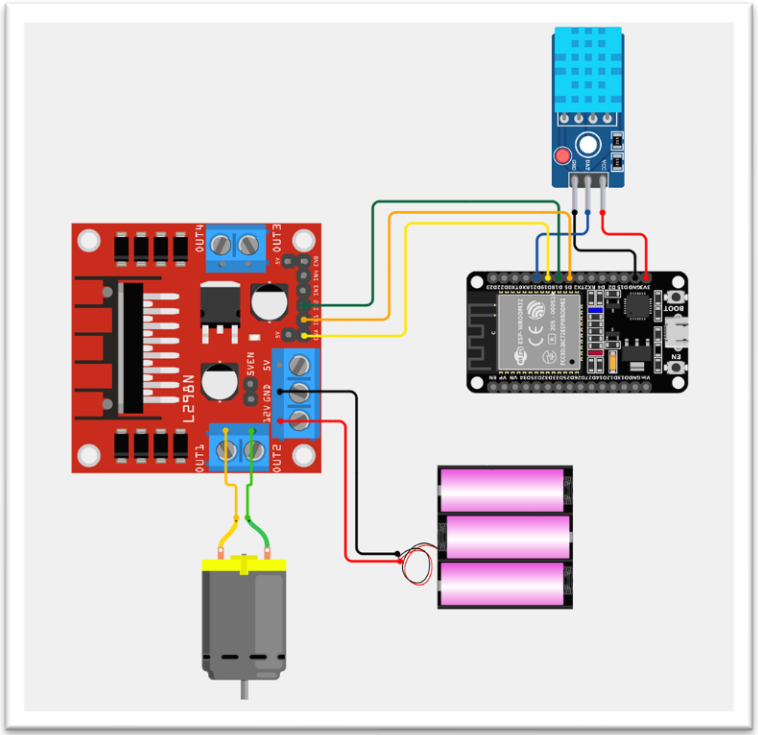
ESP32 Dev Module	The microcontroller that processes the input data and generates the output signals.
DHT11 Sensor	Temperature and humidity sensor.
DC Motor	Attached to a fan to cool the area.
H-Bridge Driver Module	Regulates the PWM and direction of the motor.
Power Supply (Solar Panel)	Powers the motor with 12V DC.
Voltage Regulator (LM317)	Protects the motor from potential overvoltage from the solar panel.

For more info:
https://github.com/Ahmed-Husseini/

LinkedIn Profile:
https://www.linkedin.com/in/ahmed107/

Hardware Connections

Pin 3.3V	DHT11_VCC
Pin GND	DHT11_GND
Pin D21	DHT11_Data_Pin
Pin D19	Driver_EN (PWM)
Pin D18	Driver_IN1
Pin D5	Driver_IN2
Motor_Pin1	Driver_OUT1
Motor_Pin2	Driver_OUT2
Battery_+	Driver_12V
Battery_-	Driver_GND



02

Software and Codes

Code {Initializing Variables}

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// WiFi credentials
#define ssid "HOME 2"
#define password "AAE@2021"

// Pin definitions
#define DHTPIN 21
#define FanPIN 19
#define CPIN 18
#define ACPIN 5

// DHT sensor setup
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// PWM configuration
#define PWM_FREQ 25000
#define PWM_RESOLUTION 8  //
int ledChannel;
```

```
// MQTT broker settings
const char broker[] = "broker.emqx.io";
const int port = 1883;

// Variables for time and environment readings
long long last_time = 0;
float Temperature = 0;
float Humidity = 0;

// Control flags and values
bool ON = false;
bool Auto = false;
int Speed = 0;
bool Direction = false;
float MaxTemp = 0;
float MaxHum = 0;
bool Alarm = false;

// WiFi and MQTT client objects
WiFiClient wificlient;
PubSubClient client(wificlient);
```

Code {Callback Function}

```
47  // MQTT callback function to handle incoming messages
48  void callback(const char topic[], byte* payload, unsigned int length) {
49    String msg = String((char*)payload).substring(0, length);
50
51    Serial.print("Topic: ");
52    Serial.println(topic);
53    Serial.print("Message: ");
54    Serial.println(msg);
55
56    // Handle different subscribed topics
57    if (strcmp(topic, "/FanController/ON") == 0) {
58      ON = (msg == "ON");
59    }
60
61    else if (strcmp(topic, "/FanController/Auto") == 0) {
62      Auto = (msg == "Auto");
63    }
64
65    else if (strcmp(topic, "/FanController/Speed") == 0) {
66      Speed = msg.toInt();
67    }
68
69    else if (strcmp(topic, "/FanController/Direction") == 0) {
70      Direction = (msg == "Clockwise");
71    }
72
73    else if (strcmp(topic, "/FanController/MaxTemp") == 0) {
74      MaxTemp = msg.toFloat();
75    }
76
77    else if (strcmp(topic, "/FanController/MaxHum") == 0) {
78      MaxHum = msg.toFloat();
79    }
80  }
```

Code {Setup}

```
// Initial setup function
void setup() {
  pinMode(FanPIN, OUTPUT);
  pinMode(CPIN, OUTPUT);
  pinMode(ACPIN, OUTPUT);

  // Attach PWM to fan pin
  ledChannel = ledcAttach(FanPIN, PWM_FREQ, PWM_RESOLUTION);

  dht.begin();  // Initialize DHT sensor

  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("\nConnecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected.");
  Serial.println("IP: " + WiFi.localIP().toString());

  // Connect to MQTT broker
  client.setServer(broker, port);
  client.setCallback(callback);

  Serial.print("\nConnecting to broker");
  while (!client.connect("tyujnb")) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("\nConnected to broker");

  // Subscribe to relevant topics
  client.subscribe("/FanController/ON");
  client.subscribe("/FanController/Auto");
  client.subscribe("/FanController/Speed");
  client.subscribe("/FanController/Direction");
  client.subscribe("/FanController/MaxTemp");
  client.subscribe("/FanController/MaxHum");

  Serial.println("\nReady to publish messages");

  last_time = millis();
}
```

Code {Main Loop}

```
// Main loop function
void loop() {
  client.loop();  // Handle incoming/outgoing MQTT messages

  // Read temperature and humidity every 2 seconds
  if ((millis() - last_time) >= 2000) {
    float temp = dht.readTemperature();
    float hum = dht.readHumidity();
    if (!isnan(temp))
      Temperature = temp;
    if (!isnan(hum))
      Humidity = hum;
    last_time = millis();
  }

  // Publish temperature and humidity values
  String tempStr = String(Temperature, 2);
  String humStr = String(Humidity, 2);
  client.publish("/FanController/Temp", tempStr.c_str());
  client.publish("/FanController/Humidity", humStr.c_str());

  // Determine if alarm should be triggered
  if ((Temperature > MaxTemp) || (Humidity > MaxHum)) {
    client.publish("/FanController/Alarm", "Yes");
    Alarm = true;
  }
  else {
    client.publish("/FanController/Alarm", "No");
    Alarm = false;
  }
```

```
  // PWM and fan speed logic
  int pwmValue = 0;
  int tempRation = 0;
  int humRation = 0;

  if (!Auto) {
    if (ON)
      pwmValue = map(Speed, 0, 100, 0, 255);
    else
      pwmValue = 0;
  } else {
    if (Alarm) {
      if ((Temperature - MaxTemp) > 0)
        tempRation = (int)(((Temperature - MaxTemp) / MaxTemp) * 255);
      if ((Humidity - MaxHum) > 0)
        humRation = (int)(((Humidity - MaxHum) / MaxHum) * 255);
      pwmValue = max(tempRation, humRation);
    }
    else
      pwmValue = 0;
  }

  // Clamp PWM value between 0 and 255
  pwmValue = constrain(pwmValue, 0, 255);

  // Write PWM value to fan pin
  ledcWrite(FanPIN , pwmValue);
```

```
  // Set fan direction
  if (Direction) {
    digitalWrite(CPIN, HIGH);
    digitalWrite(ACPIN, LOW);
  }
  else {
    digitalWrite(CPIN, LOW);
    digitalWrite(ACPIN, HIGH);
  }
}
```

03

Communication Protocol

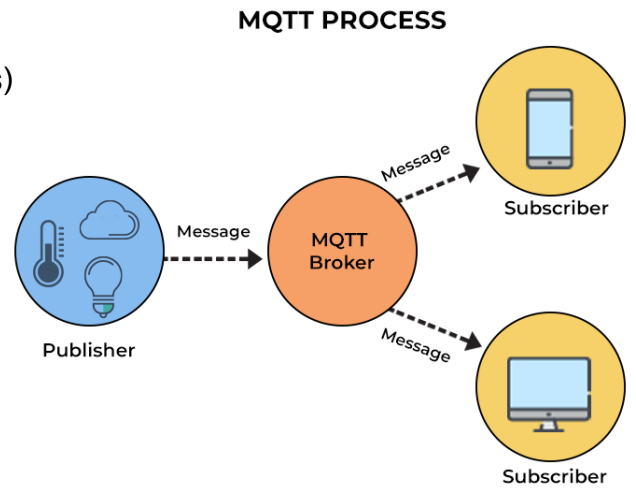
What is MQTT?

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol designed for reliable and efficient communication, especially over constrained networks such as low-bandwidth or high-latency connections. It's widely used in **IoT** applications where devices need to send small amounts of data frequently.

Key Features

- Lightweight and Low Bandwidth Usage
- Low Power Consumption (ideal for battery-powered devices)
- Reliable Delivery with different Quality of Service levels
- Asynchronous Communication
- Runs on TCP/IP

spiceworks



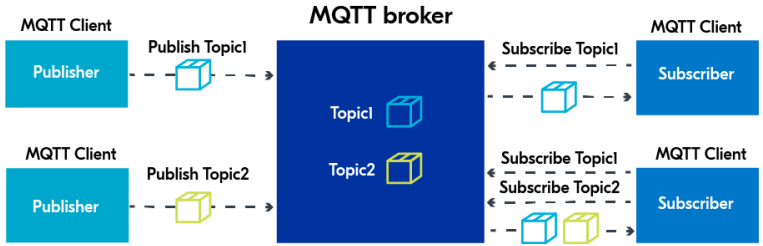
Basic MQTT Concepts

- **Broker**
  - The central server that handles all message routing.
  - Example: Eclipse Mosquitto, HiveMQ, EMQX.

- **Client**
  - Any device or application that connects to the broker (publisher or subscriber).
  - Can be a sensor, mobile app, server, etc.

- **Topic**
  - A UTF-8 string used to filter and categorize messages.
  - Example: `room/sensors/temperature`

- **Publish**
  - When a client sends a message to a specific topic.



- **Subscribe**
  - When a client expresses interest in receiving messages on a specific topic.

How it Works

1. Client A (publisher) connects to the broker
2. Client B (Subscriber) connects to the broker and subscribes to a topic (e.g., `sensors/temperature`)
3. Client A publishes a message to `sensors/temperature`
4. The Broker forwards the message to all clients subscribed to that topic.

QoS Level	Description	Use Case
0	At most once (fire and forget)	Non-critical data
1	At least once (can be duplicated)	Basic reliability
2	Exactly once (no duplicates)	Critical transactions (rarely used)

Used MQTT Topics

1. FanController/Temp
2. FanController/Humidity
3. FanController/ON
4. FanController/Auto
5. FanController/Speed
6. FanController/Direction
7. FanController/MaxTemp
8. FanController/MaxHum
9. FanController/Alarm

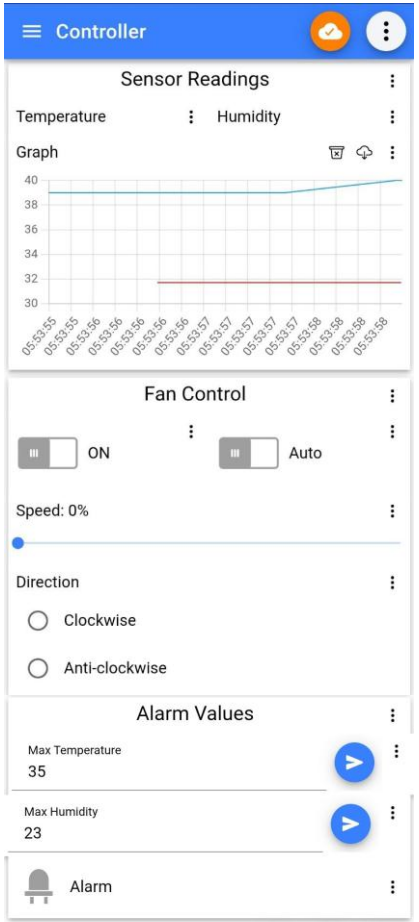
- ESP32 Publishes on this topic
- ESP32 is subscribed to this topic

04

Application Dashboard

IoT MQTT Panel

1. Temperature and humidity readings from the sensor appear at the top digital display and also on the graph.
2. ON/OFF switch to manually control the fan.
3. Auto/Manual switch to automate the process.
4. Speed bar controls the fan speed at manual mode only.
5. Directions radio buttons control the spinning direction of the fan.
6. Max Temperature and Max Humidity are the limits that when exceeded, the alarm will start.
7. If the alarm starts, and Auto mode is activated, the fan will start automatically with a speed relevant to the difference.



Thank You