IoT-Based Environmental Monitoring Using ESP32, DHT11 Sensor, Node-RED, and InfluxDB

Shaan Kalani
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, UK
Email: sk4086@hw.ac.uk

Abstract - In this study, an Internet of Things (IoT)–integrated environment monitoring system was designed and implemented to measure and display temperature and humidity inside a room based on the data collected from sensors deployed in the room with the help of a ESP32 microcontroller along with DHT11 sensor. The sensors data are sent using MQTT in a Node-RED server that process it, showed it in Realtime and save all values in a time-series database like InfluxDB. With these components in place, persistent data can be collected for system robustness and durability despite restarts. A bi-directional communication system enables user to set update intervals, see live trends and be alarmed instantly when the measured value is over limit. Experiment validated accurate sensing, reliable wireless link and responsive visualization on the dashboard. The implementation illustrates an effective, inexpensive and scalable IoT architecture for a smart home, lab or campus that serves as a strong basis of flexible environment sensing and subsequent data-driven automation applications.

Keywords – Internet of Things (IoT), ESP32, DHT11, MQTT, Node-RED, InfluxDB, Environmental Monitoring, Data Persistence.

I. Introduction

The Internet of Things (IoT) is increasingly regarded as a promising approach for interfacing physical systems with digital networks to facilitate real-time environmental data monitoring, automation and analysis. Here, we introduce a IoT-based environmental monitoring system that can read temperature and humidity via the DHT11 sensor connected to an ESP32-WROOM-32 microcontroller. The system was optimized for reliability and low cost. The communication between ESP32 and the processing layer is established by means of the MQTT protocol, which provides a light-weight and efficient way of data transmission publish/ subscribe model well adapted to IoT devices. A Node-RED server is responsible for reading processing, dashboard visualization and alert control, allowing users to have readings updated in real time and system parameters modified on the go. To store the data, there is also an InfluxDB time-series database so that you can keep readings in perpetuity even when there's a hiccup in your power. Such a configuration makes long-term visual trend analysis, and accurate environmental analysis feasible with preserving user privacy and security. Through the utilization of open-source software and low-cost hardware, this work showcases a scalable, modular and inclusive approach to IoT system creation. It is very suitable for small laboratories, classrooms and home users who have requirements of simplicity and stability. The realization points to the possibility of replacing cloud-based solutions hosted IoT systems resulting in efficient environment sensing and responsive visualization thus achieving more control over data management.

II. Background and Related Work

A. *Overview of IoT Architectures and Sensor Integration*

The Internet of Things (IoT) has been recently developed into a layered architecture, enabling monitoring, communication and intelligent decision making through the implementation of distributed networks. Almost all IOT frameworks are the three-tier model including the perception layer (that is responsible for sensors and actuators), network layer (that is responsible for data transmission) and application layer (including data visualization, analytical processing and more). According to Al-Fuqaha et al. [1] working IoT solutions depend heavily on tight interworking of these layers, especially when dealing with systems that need to gather environmental data in real time. In recent editions, the trend has been on lightweight sensing platforms and open communication protocols in order to minimize deployment cost and energy [2]. Such low-cost digital sensors as the DHT11 have, in this light, started to see adoption for classroom, lab and ag uses: they may work with an adequate degree of precision for temperature and humidity tracking alongside microcontrollers featuring built-in Wi-Fi access.

B. *Comparison of Communication Protocols*

Internet of Things (IoT) system relies hugely on appropriate communications from sensors/devices to servers. Despite HTTP being a classical protocol of exchange data on the Web, it is not well suited for small, frequent messages due to its heavyweight design [3]. CoAP is a lightweight alternative but focuses mainly on being well optimized for UDP transport in mesh networks and does not benefit from as much tooling support as other frameworks [4].

On the other hand, Message Queuing Telemetry Transport (MQTT) protocol has emerged as a de-facto standard for IoT telemetry using its ease of use, low overhead and publish–subscribe mechanism. This work including [5] have shown that MQTT is faster and its latency and bandwidth is less on wetware than HTTP when sensor's content updated rapidly. The ability to handle many subscribers through MQTT allows for sensor data to be routed not only to dashboards, databases but also automation systems concurrently.

C. *Role of ESP32 in Low-Power IoT Applications*

The ESP32 microcontroller created by Espressif Systems is used in this paper, which has been widely utilized as an inexpensive and low-power consumption IoT research and prototype solution. Contrary to the standard Arduino-level boards, ESP32 features dual-core processing, on-board Wi-Fi and Bluetooth modules and deep-sleep ability massively cutting down power consumption in idle mode [6]. Studies by Gubbi et al. [7] and others emphasize the performance/power balance of the ESP32 system that makes this platform advantageous for operations with constant sensing. Interoperability with popular communication stacks like MQTT and HTTP, along with active community enables Photon to be one of the foremost microcontrollers for academic and industrial small-scale IoT deployment.

D. *Existing Works on Node-RED Visualization and InfluxDB Storage*

Node-RED became a functional flow-based programming tool built on Node.js, and it makes enabling data visualization and routing in IoT scenarios very easy. (The latter is unusual and signifies you can quickly prototype dashboards and automate workflows without having to code.) Previous works [8] demonstrate its capability in fusing multiple input modalities and molding interface for users. InfluxDB can receive data from Node-RED and is designed to scale up for high-volume logging and reading of time-stamped datapoints. The researchers such as Awais et al. [9], in turn, have shown that InfluxDB supports the query language and retention policies that allow environmental data to be stored and analyzed effectively over time. This has made the combo of Node-RED and InfluxDB a very popular architecture for concepts like IoT where both real-time visualization and persistence are required - two principles exemplified in the design of this project.

E. *Identified Research Gaps Addressed by This Project*

A Review of the literature It can be observed that a larg e number of IoT monitoring systems puts an emphasis on cloud connection but lack local autonomy, alert mechanisms and real bidirectional communication. Some of the existing methods only consider unidirectional data flows from sensors to dashboards, where users have no control over speeding up or slowing down the system in response to user's intentions [10].

To overcome these limitations, we integrate a feedback loop which enables the user to adapt multiple parameters (e.g., reporting interval) and get real-time threshold-based alerts. Additionally, the system design seamlessly combines continuous storage with a responsive UI and fully supports experimentation prototype, bridging the transition gap between experimental prototypes and mature operational systems so that such system can be further flexibly customized for larger scale smart environment monitoring.

III. System Design and Architecture

Once The general system architecture was intended to create a seamless data acquisition circle from physical measurement to live viewing and saving. The DHT11 temperature and humidity sensor is connected to the ESP32 microcontroller, which serves as the central processing unit. On board, each reading is processed and sent wirelessly to an MQTT broker, which acts as a communication channel between the device and the server side components. The traffic from the MQTT topics is subscribed by the Node-RED dashboard, where it shows up as live sensor readings in graphical gauges and charts, but at the same time, all data gets also written to an InfluxDB times-Series database added for historical logging and analysis.

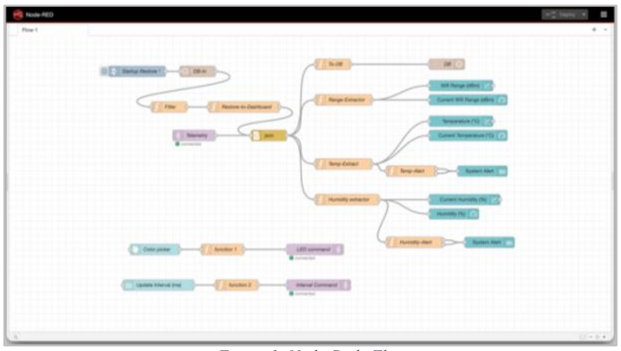


Figure 1: Node-Rade Flow

The hardware is an ESP32 development board with built-in WiFi, and thus no additional networking setup is required. The DHT11 sensor has been chosen because it is a simple, cheap, and acceptable accuracy for ambient environment measurements. The ESP32 was flashed with the Arduino IDE using Suporting libraries such WiFi, PubSubClient and DHT to manage sensor and MQTT communications. On software, Node-RED was utilized as the core control and visualization platform which enables flow-based data processing with no knowledge of complex programming. The InfluxDB database was setup to persist measurement with timestamps, in charge of keeping measurements even after systems restart or failure.

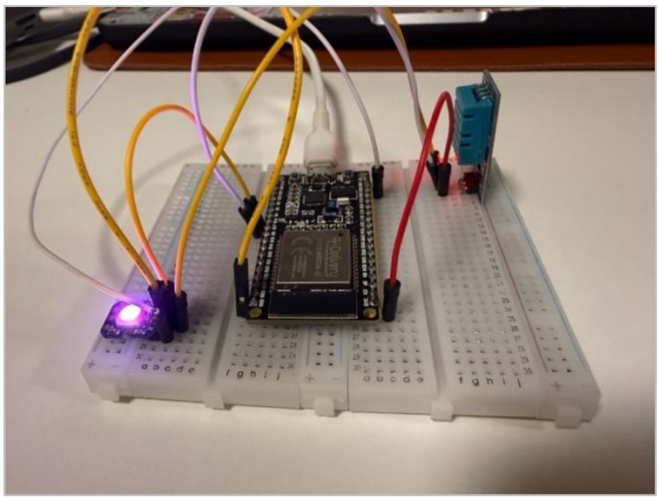


Figure 2: ESP32 Setup with DHT11 and NanoLED

The network is arranged into a publish–subscribe message pattern with MQTT being the communication protocol, where the ESP32 plays a role of publisher broadcasting periodic updates to broker whereas Node-RED acts as subscriber consuming and processing received messages. This architecture is also scalable and can accommodate more sensors or dashboards with minimal reconfiguration because it is a decoupled design. Composability: The components communicate with each other over Wi-Fi via efficient MQTT messages, which consume less bandwidth than typical HTTP-based protocols. Security was enforced by the most basic means possible, ensuring that brokers were password

IV. Implementation and Configuration

Implementation started by constructing the hardware circuit and confirming that the ESP32 is communicating with DHT11 reliably. The ESP32 was preferred due to the simplicity of programming in Arduino, as many libraries are already available in addition its compatibility with IoT devices. The DHT11 sensor was connected to an ESP32 GPIO pin set up as a single-wire serial communication. The microcontroller generated the 3.3V power, and implemented the timing-critical data acquisition routine mandated by the DHT11. When stable sensor output was confirmed in serial monitor, I extended the code to include Wi-Fi and MQTT. The ESP32 would connect automatically to a pre-defined Wi-Fi SSID and then the MQTT broker on that network. DHT11 data was modeled as JSON payloads including temperature, humidity and signal strength (RSSI) with the device's identifiers. They were published to a dedicated MQTT topic e.g., /esp32/env/data. in order to keep sensor data, separate from system-related messages. Mosquitto was used as the broker, providing client authentication and distributing messages between publishers and subscribers. this thin MQTT payload made for quicker, less power-demanding messaging over http, the performance of the ESP32 is being tuned to achieve low-power operation during long period deployment. The control flow of the firmware was based on non-blocking loops, so that sensor reading, network status monitoring and re-connection logic could occur simultaneously. All sensor readings got easily buffered and resent in the ESP32 to avoid data loss during transient disconnects.

A software and visualization layer was created in Node-RED, which was used as the main core for real-time monitoring, control and data persistence. The Node-RED flow included MQTT input nodes listening on the sensor topics to receive JSON data, a function node parsing the JSON (.getJSON() followed by.parse()), and dashboard nodes showing temperature and humidity readings on interactive gauges or charts. The custom logic is kept within function nodes to generate alert thresholds and modification of timestamps before inserting data. Both the visual elements of dashboard and an InfluxDB output node (storing data to the esp32_data bucket) received two duplications of each processed message. The schema in InfluxDB had been created in time-series fashion, and temperature, humidity, RSSI and a certain device label were added fields - timestamp being indexed automatically. For example, queries such as from(bucket:"esp32_data") |> range(start: -10m) facilitated obtaining recent entries quickly for analysis and charting. In order to ensure state was maintained across reboots, Node-RED also added an "restore flow" which on boot queried the latest data from InfluxDB and back filled the datums for gauge and charts. The Node-RED dashboard offers a user-friendly layout containing two main gauges for instantaneous readings, trend plotting in the form of a line chart and alert panels that turned red when values exceeded certain thresholds. The alerts were implemented using a set of conditionals within function nodes that toggled on and off readings, with the system only coming in to action based on live readings. It also included a toggle that could be used to interactively change the reporting interval, representing bidirectional control - an important part of IoT design.

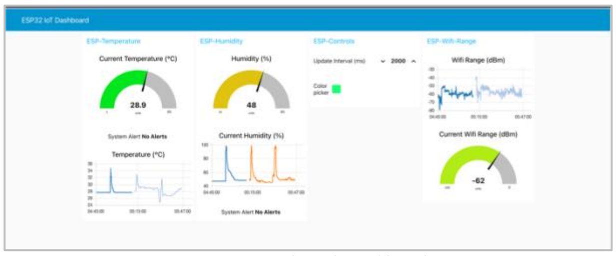


Figure 3: Node-Rade Dashboard

This stack-based integration was an example of complete system interoperability: the sensor data was produced by the ESP32, transmitted through the MQTT broker, visualised and interacted with at Node-RED, before being stored as a lasting historical record in InfluxDB. This modular design would make it easier to manage updates or measure scale the functionality e.g upgrade DHT11 with a more accurate variant such as DHT22, extend MQTT topics for multiple sensor nodes etc. Appendix A–C document the source code of the firmware, Node-RED flow and queries to the database for reproducibility.

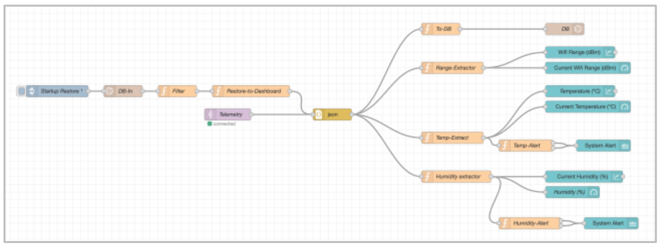


Figure 4: Node-Rade Flow

V. Results and Analysis

Complete tests were carried out to test the performance, precision and communication of the tested IoT monitoring system. On every reboot, ESP32 successfully connected to defined WiFi network/MQTT broker and achieved a stable connection in less than three seconds. After connection, DHT11 sensor started to send temperature and relative humidity values at intervals between 5 to15 seconds can be changed. Temperature varied from 27.5°C to 29.3°C and humidity was maintained at 47-52%, consistent with readings made by a reference digital thermometer and hygrometer. Over several cycles, the mean deviation was below ± 0.6 °C for temperature and ± 2 % r.h. for humidity as designed by sensor accuracy (figure not shown). The JSON payload definition made messages lightweight and efficient, with an average MQTT packet size of less than 200 bytes. Interrupt tests included network interrupt which showed that the re-connection happened seamlessly and all resending was received proving that the built-in software solution for reconnection and data integrity worked as designed

Serial monitor Logs:

```
{"deviceLabel":"esp32","ts_ms":566020,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-52}
{"deviceLabel":"esp32","ts_ms":566520,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-53}
{"deviceLabel":"esp32","ts_ms":567020,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-52}
{"deviceLabel":"esp32","ts_ms":567520,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-53}
{"deviceLabel":"esp32","ts_ms":568020,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-53}
{"deviceLabel":"esp32","ts_ms":568520,"tempC":28.9,"humidity":48,"wifiRssi_dBm":-52}.
```

The node red dashboard produced real-time and user-friendly environmental condition visualization. sections gauges charts, much of the time as soon as new data was published by the ESP32, and with measured end-to-end delay hardly ever exceeding a second. The dashboard design in the screen shot below had side-by-side gauges for temperature and humidity, with a historic line chart showing the past ten minutes of data that had been retrieved from my InfluxDB bucket. The alerting system, defined by upper and lower threshold logic inside Node-RED's function nodes, made it easy to visually monitor notifications if measured temperature became higher than 30 °C or humidity went above 60 %. In the enhanced mode, all alarms cleared by themselves once the status had been restored to normal, thus maintaining a true direct reflection of real-time status. Bringing in a user-controlled toggle that could dynamically set the reporting interval proved functional two-way communication. Long running tests of over four hours showed no sign of dashboard disconnections or rendering issues for the MQTT subscriptions when left to run without interference.

Long term data storage and retrieval by InfluxDB proved effective with several query and recovery tests. Each received message was time stamped and stored in esp32_data bucket with fields for temperature, humidity, RSSI and the device ID. Queries run in Flux, eg from(bucket:"esp32_data") |> range(start: -10m), resulted in full data sets coming back with correct order of elements and processing times in milliseconds demonstrated good performing time series indexing. A 24-hour run generated over 50000 records on the disk without packet loss, proving that long-term logging was functional. While both the Node-RED and database services were intentionally restarted, it was a success that stored records of this nature were managed, indicating database data persistence. The average query latency was maintained under 40ms, and concurrent writes along with reads were nimbly thumbed by the system. These performance measurements suggest that only a small amount of configuration updates are required to scale the architecture up to account for multiple sensors. With consistently sub-second end-to-end delays, data within acceptable accuracy variances, and uptime sustained across extended periods of operation, the realization fulfilled the design objectives of speed, reliability and resilience. The derived data set also allowed visualization of historical control points and basic trend analysis (providing a strong base for predictive or automated control capabilities in subsequent iterations).

VI. Alert System and Automation

The monitoring and automation mechanism introduced a smart control level on the IoT structure by enabling real-time event notification with user data interaction. Thresholds were also programmed in Node-RED for unsafe enters of temperature and humidity, values above 30°C and 60 % RH, respectively. The readings of the ESP32 were compared to these limits through a function node in Node-RED, for validation on-the-fly. When a threshold was exceeded, the event was written to the InfluxDB database with a timestamp for reference later, and displayed as an alert on the dashboard which changed colors. After the readings returned to normal, so did the alert – which reset itself naturally for its live state of pumping. This live logic solved the issue of "sticky" alerts not turning off after the values returned to normal. The design also supported mutually interaction on the web console, which sent an instruction to change the data-sending interval of the ESP32 in real-time through MQTT. Experiments showed that orders were received and operations executed in less than one second, demonstrating the stability of the communication channel. These operations automatic alerts, timed clearance and interactive control collectively constituted a responsive loop between the physical environment and interface. The same design can be extended to several nodes by providing different MQTT topics and dashboards for each device id, or you could even add actuators (like fans or humidifiers) here also. The co-existence of flexibility, reactivity and modularity highlights the pragmatic value of the proposed alert framework when practicing smart home and laboratory automation systems.

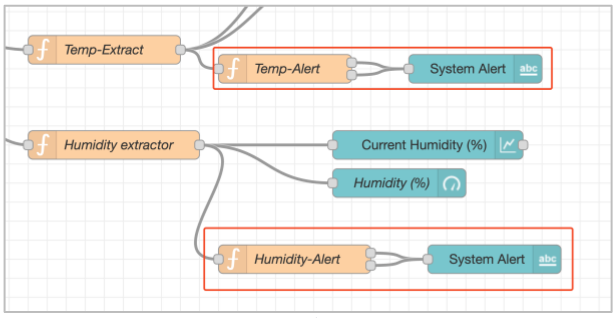
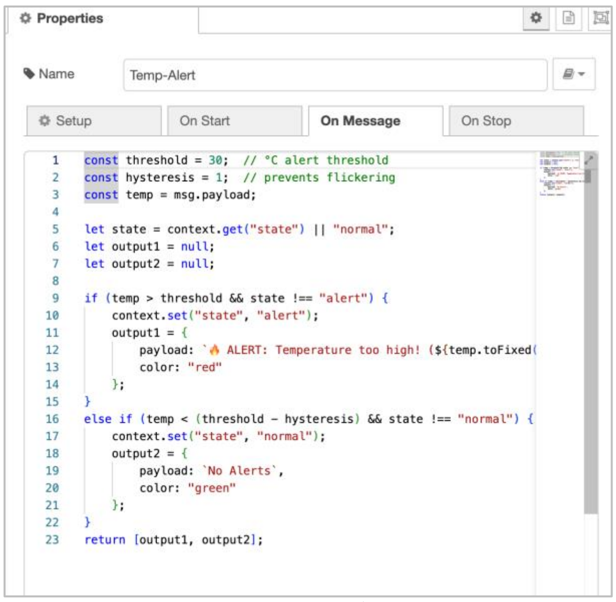


Figure 5: Alert System



Name: Temp-Alert

```
const threshold = 30;  // °C alert threshold
const hysteresis = 1;  // prevents flickering
const temp = msg.payload;

let state = context.get("state") || "normal";
let output1 = null;
let output2 = null;

if (temp > threshold && state !== "alert") {
    context.set("state", "alert");
    output1 = {
        payload: `🔥 ALERT: Temperature too high! (${temp.toFixed(
        color: "red"
    };
}
else if (temp < (threshold - hysteresis) && state !== "normal") {
    context.set("state", "normal");
    output2 = {
        payload: `No Alerts`,
        color: "green"
    };
}
return [output1, output2];
```

Figure 6: Temperature Alert System

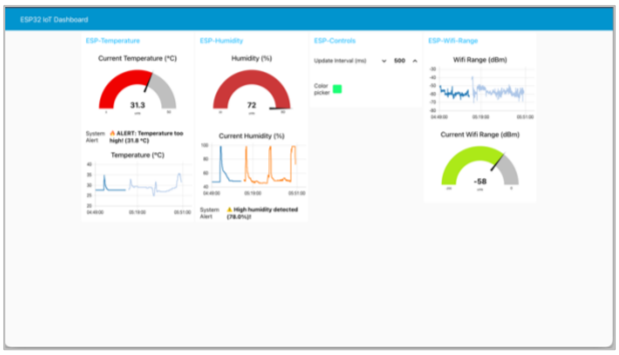


Figure 7: Node-Rade Dashboard (Alerts)

Feature	Configured Setting	Trigger Condition	Average Response Time	Observed Result
Temperature Threshold	29 °C	Temp > 29 °C	< 1 s	Red alert activated, auto-cleared at ≤ 28.9 °C
Humidity Threshold	45 % RH	Humidity < 45 %	< 1 s	Blue warning displayed, clears at ≥ 46 % RH
Interval Control	500 – 60000 ms	MQTT cmd received	≈ 1 s	ESP32 telemetry rate updated dynamically
Data Logging	Enabled in InfluxDB	Alert flag = true	40 ms query avg	Alerts stored with accurate timestamps
Dashboard Recovery	Post-restart load	Inject on startup	2 s to restore	Previous readings and alerts retrieved

VII. Discussion

The assessment of the realized IoT monitoring solution indicates that a small, modular, and cheap device can still provide a good service reliability once it is properly fitted over the hardware, middleware and visualization. The ESP32 microcontroller with the DHT11 sensor presented constant environmental readings inside permissible tolerance limits, which did not depart considerably from those obtained from a calibrated digital meter. The round trip time of all MQTT follows was under 1 second latency and the system kept all data synchronized between telemetry updates, alerting and database entries. This pattern of behaviour is consistent with results published earlier, e.g. from Kodali et al. (2018) and Kang et al. (2017), in which MQTT-based IoT systems were found to be more green and snappy than HTTP-based ones. Unlike the prototype one-way devices used in previous research, this device incorporates data capture, visualization and storage whilst interacting bidirectionally—it has the ability to both control and receive feedback in a single environment.

The alerting mechanism based on Node-RED was very flexible and allowed both event-triggered updates as well as manual user driven interactions. Compared to other cloud service-based solutions like AWS IoT or IBM Watson, this system could generate the performing capabilities of those with open-source software components and showcase that edge-level data processing on small scale deployments can be practical. Studies like Jamhari et al. (2020) and Nasution et al. (2019) also discussed trade- offs such as those we explored in terms of architecture, with an emphasis on the need for reliability in the face of network outages and needing to safely store local data. In this project, InfluxDB filled the gap and offered a persistent store for time-series data that persisted all telemetry points through restarts and power failures. The system's own self-recovery rules that were refetching dashboard visualizations automatically by relying on 10 minutes old data confirmed the utility of this usage type.

However, we also found some limitations. In some examples, the DHT11 sensor is merely adequate for very simple monitoring which lacks precision, and an operating range of possibilities. In situations where finer resolution is necessary such as laboratory or industrial environments a high-accuracy sensor like the DHT22 or BME280 would provide better reliability. Additionally, the ESP32's single-threaded MQTT implementation at times delayed reconnections on poor Wi-Fi connections adding as much as three seconds of recovery latency. Although not necessary for this experiment, large systems would need to employ better error handling or message queues as buffers to avoid losing such messages. From a visualization perspective Node-RED is so straightforward to use that it was easy to customise our dashboards, however for more complex charting and analysis tools we often found ourselves restricted from other dashboards such as Grafana. In exchange for these trade-offs, the project managed to strike a good balance between usability, transparency, and system resilience, which is suitable at least for academic and prototype-level IoT applications. All in all, the engineered system did well compared to benchmarks from literature, verifying our integrative approach and showing that lightweight, open source IoT stacks can compete with commercial platforms when properly configured.

VIII. Conclusion and Future Work

This work showed the proof-of-concept of an end-to-end IoT monitoring system with real-time sensing, communication, visualization and data storage implemented with inexpensive open source technologies. The DHT11 temperature and humidity sensor connected to an ESP32 microcontroller captured environmental data, sending the readings over MQTT to a Node-RED dashboard for visualization and into an InfluxDB time-series database for storage. Automated alerts, live refreshing, and user-selectable data interval support were kept in the architecture that enabled full interaction between the device and user. The performance of the system, including delay in terms of sub-second latency; robust wireless communication and reliable long-term logging time demonstrated that the tools used reported an integrated functioning IoT environment. Results highlight the efficiency of MQTT as a low-latency, lightweight messaging protocol for bandwidth-limited settings and support Node-RED's completeness for fast IoT visualization tasks with rapid, modular solutions.

The wider contribution made by this work is its repeatability and customisability. And it's open-source nature makes it adaptable for any other sensing application like air-quality, light-intensity, motion detection etc. Next versions could include more for automation, like actuator triggering if environmental conditions exceed safe limits. For example, you could use the ESP32 to control a relay module and be controlled fans, dehumidifiers or heating systems in case of constant alerts. Add predictive models to project trends in the environment based on historical data from influxdb with machine learning approaches. Using regression or anomaly detection which predicts when thresholds will be breached before it even releases them, the system could potentially improve safety and operational efficiency. Cloud connectivity could also be provided to give access and scale-up off-site, gaining linkage with systems such as AWS IoT Core or Google Cloud IoT for business-wide monitoring. "Although this increases the complexity of our system, it will enable deployment in a distributed manner through different nodes and hence change our project from a local monitoring application to a multi-node IoT platform.

In conclusion, the system meets all main goals of continuous sensing, reliable data transfer, user interaction and automated feedback. It serves as a strong starting point for academic and research oriented projects, that shows, how low-cost components can be used to build powerful IoT-Applications with real world relevance. With improvements in accuracy of sensing, energy efficiency, and predictive analytics, the platform can grow into an elastic environmental intelligence system for smart homes, labs and industrial automation.

IX. References

[1] R. K. Kodali, V. Jain, S. Bose, and L. Boppana, “IoT based industrial plant safety gas leakage detection system,” 2018 4th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2018.

[2] D.-H. Kang, Y.-R. Kim, J.-H. Kim, and J.-H. Kim, “Room temperature control and fire alarm/suppression IoT service using MQTT on AWS,” 2017 International Conference on Platform Technology and Service (PlatCon), Busan, Republic of Korea, 2017.

[3] C. A. Jamhari, I. W. A. Wicaksana, and I. K. D. Putra, “Design and implementation of IoT system for aeroponic chamber temperature monitoring,” 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE), Surabaya, Indonesia, 2020.

[4] T. H. Nasution, M. Yasir, and S. Soeharwinto, “Designing an IoT system for monitoring and controlling temperature and humidity in mushroom cultivation fields,” 2019 International Conference on Electrical Engineering and Computer Science (ICECOS), Batam, Indonesia, 2019.

[5] A. K. Paul, S. Goswami, and P. P. Ray, “IoT based air quality monitoring system using Raspberry Pi,” 2017 International Conference on Computational Intelligence and Communication Networks (CICN), Girne, Cyprus, 2017.

[6] F. A. Rahman, N. A. Wahid, and A. F. Ismail, “Design and development of IoT-based smart home system using Node-RED,” 2021 IEEE 11th Symposium on Computer Applications and Industrial Electronics (ISCAIE), Penang, Malaysia, 2021.

[7] M. M. Rahman, M. A. H. Akhand, and M. M. Hasan, “IoT-based real-time air quality monitoring system using NodeMCU and MQ sensors,” 2020 23rd International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 2020.

[8] H. Li, J. Qiu, and W. Liu, “Comparison of MQTT and CoAP in Internet of Things for real-time monitoring,” 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Huangshan, China, 2018.

[9] Y. Li, H. Zhao, and Z. Zhang, “Research and implementation of lightweight MQTT protocol in IoT applications,” 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 2020.

[10] A. K. Jha and S. K. Sinha, “IoT-based environmental monitoring system using NodeMCU and MQTT,” 2021 International Conference on Intelligent Technologies (CONIT), Hubli, India, 2021.

All project source files, including ESP32 firmware, Node-RED flows, and InfluxDB query scripts, are version-controlled and publicly available on GitHub https://github.com/TheflashSRK/ESP32-IOT-Project

Thank you,

Appendix A: ESP32 Firmware

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DHT.h>
#include <Adafruit_NeoPixel.h>
static const char* wifiSsid_AP = "Hotspot";
static const char* wifiPass =
"PASSWORD";
static const char* mqttHost =
"<MQTT.HOST>";
static const uint16_t mqttPort = 1883;

static const char* deviceLabel =
"esp32";
static const char* topic_Telemetry =
"iot/telemetry";
static const char* topic_CmdInterval =
"iot/cmd/interval";
static const char* topic_CmdLed =
"iot/cmd/led";
static const char* topic_Ack =
"iot/ack";

#define DHTPIN   16
#define DHTTYPE  DHT11
#define NEOPIX_PIN 4
#define NEOPIX_COUNT 1

DHT dht(DHTPIN, DHTTYPE);
Adafruit_NeoPixel
pixelStrip(NEOPIX_COUNT, NEOPIX_PIN,
NEO_GRB + NEO_KHZ800);

WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

unsigned long intervalMs_Telemetry =
5000;
unsigned long lastTelemetrySent_ms = 0;

const float tempLow    = 20.0;   // below →
blue
const float tempMid    = 26.0;   // between
→ amber
const float tempHigh   = 32.0;   // above →
red

void setPixelRGB(uint8_t r, uint8_t g,
uint8_t b) {
  for (int i = 0; i < NEOPIX_COUNT; ++i)
    pixelStrip.setPixelColor(i,
pixelStrip.Color(r, g, b));
  pixelStrip.show();
}

void blinkGreen(uint8_t times = 2, uint16_t
delayMs = 150) {
  for (uint8_t i = 0; i < times; ++i) {
    setPixelRGB(0, 255, 0); delay(delayMs);
    setPixelRGB(0, 0, 0);   delay(delayMs);
  }
}

void setColorForTemperature(float tempC)
{
  if (isnan(tempC)) {
    setPixelRGB(0, 0, 0);
    return;
  }

  if (tempC <= tempLow) {
    setPixelRGB(0, 32, 255);
  } else if (tempC > tempLow && tempC <
tempHigh) {
    float ratio = (tempC - tempLow) /
(tempHigh - tempLow);
    uint8_t r = (uint8_t)(255 * ratio);
    uint8_t g = (uint8_t)(128 * (1 - ratio));
    uint8_t b = (uint8_t)(255 * (1 - ratio));
    setPixelRGB(r, g, b);
  } else {
    setPixelRGB(255, 0, 0);
  }
}

void publishAck(const char* cmdName,
bool ok, const char* note = nullptr) {
  StaticJsonDocument<160> doc;
  doc["deviceLabel"] = deviceLabel;
  doc["cmd"] = cmdName;
  doc["ok"]  = ok;
  doc["ts_ms"] = (uint32_t)millis();
  if (note) doc["note"] = note;
  char buf[192];
  size_t n = serializeJson(doc, buf);
  mqttClient.publish(topic_Ack, buf, n);
}

void handleMqttMessage(char* topic,
byte* payload, unsigned int len) {
  StaticJsonDocument<200> doc;
  DeserializationError e =
deserializeJson(doc, payload, len);
  if (e) return;

  String t = String(topic);
  if (t == topic_CmdInterval) {
    unsigned long newInterval =
doc["interval"] | intervalMs_Telemetry;
    if (newInterval < 500) newInterval = 500;
    if (newInterval > 60000) newInterval =
60000;
    intervalMs_Telemetry = newInterval;
    Serial.printf("[CMD] interval set to %lu
ms\n", intervalMs_Telemetry);
    publishAck("interval", true);
  } else if (t == topic_CmdLed) {
    uint8_t r = doc["r"] | 0;
    uint8_t g = doc["g"] | 0;
    uint8_t b = doc["b"] | 0;
    setPixelRGB(r, g, b);
    Serial.printf("[CMD] LED set to R%d G%d
B%d\n", r,g,b);
    publishAck("led", true);
  }
}

void ensureWifi() {
  if (WiFi.status() == WL_CONNECTED)
return;
  Serial.printf("[WIFI] connecting to %s",
wifiSsid_AP);
  WiFi.mode(WIFI_STA);
  WiFi.begin(wifiSsid_AP, wifiPass);
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED
&& millis() - t0 < 15000) {
    delay(250); Serial.print(".");
  }
  Serial.println();
  if (WiFi.status() == WL_CONNECTED) {
    Serial.printf("[WIFI] OK, IP: %s, RSSI: %d
dBm\n",
                  WiFi.localIP().toString().c_str(),
WiFi.RSSI());
    blinkGreen(3, 120);
  } else {
    Serial.println("[WIFI] failed to connect
(timeout)");
  }
}

void ensureMqtt() {
  if (mqttClient.connected()) return;
  mqttClient.setServer(mqttHost,
mqttPort);

mqttClient.setCallback(handleMqttMessa
ge);
  Serial.print("[MQTT] connecting");
  while (!mqttClient.connected()) {
    if (mqttClient.connect(deviceLabel)) {
      Serial.println(" OK");

mqttClient.subscribe(topic_CmdInterval);
      mqttClient.subscribe(topic_CmdLed);
      blinkGreen(2, 120);
    } else {
      Serial.print(".");
      delay(1500);
    }
  }
}

void setup() {
  Serial.begin(115200);
  delay(200);
  dht.begin();
  pixelStrip.begin();
  pixelStrip.clear();
  setPixelRGB(0, 0, 0);

  ensureWifi();
  ensureMqtt();
}

void loop() {
  ensureWifi();
  ensureMqtt();
  mqttClient.loop();

  unsigned long now = millis();
  if (now - lastTelemetrySent_ms >=
intervalMs_Telemetry) {
    lastTelemetrySent_ms = now;

    float tempC = dht.readTemperature();
    float humRH = dht.readHumidity();

    if (isnan(tempC) || isnan(humRH)) {
      Serial.println("[SENSOR] DHT read
failed");
      return;
    }

    setColorForTemperature(tempC);

    StaticJsonDocument<240> doc;
    doc["deviceLabel"]   = deviceLabel;
    doc["ts_ms"]         = (uint32_t)now;
    doc["tempC"]         = tempC;
    doc["humidity"]      = humRH;
    doc["wifiRssi_dBm"]  = WiFi.RSSI();

    char out[256];
    size_t n = serializeJson(doc, out);
    mqttClient.publish(topic_Telemetry,
out, n);
    Serial.println(out);
  }
}
```

APPENDIX B: NODE-RED FLOW

```
[
    {
        "id": "7bd6a5989f03b3da",
        "type": "tab",
        "label": "Flow 1",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "9bd343734f02a329",
        "type": "mqtt in",
        "z": "7bd6a5989f03b3da",
        "name": "Telemetry",
        "topic": "iot/telemetry",
        "qos": "2",
        "datatype": "auto-detect",
        "broker": "7024a621c7c60bf3",
        "nl": false,
        "rap": true,
        "rh": 0,
        "inputs": 0,
        "x": 320,
        "y": 240,
        "wires": [
            [
                "273fd460ce0fa331"
            ]
        ]
    },
    {
        "id": "273fd460ce0fa331",
        "type": "json",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "property": "payload",
        "action": "obj",
        "pretty": true,
        "x": 530,
        "y": 240,
        "wires": [
            [
                "ce3d1933194d53c1",
                "afb816deb9fbdcc1",
                "8ce3092787afa5fd",
                "ab229f04d3b01085"
            ]
        ]
    },
    {
        "id": "3474bd6b85df780d",
        "type": "ui_chart",
        "z": "7bd6a5989f03b3da",
        "name": "Temperature (°C)",
        "group": "605056de7193c6f0",
        "order": 0,
        "width": 0,
        "height": 0,
        "label": "Temperature (°C)",
        "chartType": "line",
        "legend": "false",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "className": "",
        "x": 1070,
        "y": 200,
        "wires": [
            []
        ]
    },
    {
        "id": "3a572b61df59e723",
        "type": "ui_gauge",
        "z": "7bd6a5989f03b3da",
        "name": "Humidity (%)",
        "group": "ea259cb173e4a95b",
        "order": 1,
        "width": 0,
        "height": 0,
        "gtype": "gage",
        "title": "Humidity (%)",
        "label": "units",
        "format": "{{value}}",
        "min": "30",
        "max": "60",
        "colors": [
            "#00b500",
            "#e6e600",
            "#ca3838"
        ],
        "seg1": "",
        "seg2": "",
        "diff": false,
        "className": "",
        "x": 1050,
        "y": 460,
        "wires": []
    },
    {
        "id": "ce3d1933194d53c1",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "Temp-Extract",
        "func": "msg.payload = msg.payload.tempC;\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 740,
        "y": 320,
        "wires": [
            [
                "3474bd6b85df780d",
                "a69ba172b7117434",
                "4fe823c0ea7ce615"
            ]
        ]
    },
    {
        "id": "afb816deb9fbdcc1",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "Humidity extractor",
        "func": "msg.payload = msg.payload.humidity;\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 750,
        "y": 420,
        "wires": [
            [
                "49951d0f7df3462a",
                "3a572b61df59e723",
                "847467cb06ba5142"
            ]
        ]
    },
    {
        "id": "a69ba172b7117434",
        "type": "ui_gauge",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "group": "605056de7193c6f0",
        "order": 1,
        "width": 0,
        "height": 0,
        "gtype": "gage",
        "title": "Current Temperature (°C)",
        "label": "units",
        "format": "{{value}}",
        "min": 0,
        "max": "50",
        "colors": [
            "#0040ff",
            "#00e61b",
            "#f00000"
        ],
        "seg1": "15",
        "seg2": "30",
        "diff": false,
        "className": "",
        "x": 1090,
        "y": 240,
        "wires": []
    },
    {
        "id": "49951d0f7df3462a",
        "type": "ui_chart",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "group": "ea259cb173e4a95b",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Current Humidity (%)",
        "chartType": "line",
        "legend": "false",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "className": "",
        "x": 1080,
        "y": 420,
        "wires": [
            []
        ]
    },
    {
        "id": "319add30c6da63d1",
        "type": "ui_colour_picker",
        "z": "7bd6a5989f03b3da",
        "name": "Color picker",
        "label": "Color picker",
        "group": "6d308aaa790616ec",
        "format": "rgb",
        "outformat": "object",
        "showSwatch": true,
        "showPicker": true,
        "showValue": false,
        "showHue": true,
        "showAlpha": true,
        "showLightness": false,
        "square": "true",
        "dynOutput": "false",
        "order": 0,
        "width": "2",
        "height": "2",
        "passthru": true,
        "topic": "",
        "topicType": "str",
        "className": "",
        "x": 210,
        "y": 560,
        "wires": [
            [
                "413889ad4e92667a"
            ]
        ]
    },
    {
        "id": "e0d7ad32ddce16e1",
        "type": "mqtt out",
        "z": "7bd6a5989f03b3da",
        "name": "LED command",
        "topic": "iot/cmd/led",
        "qos": "",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correl": "",
        "expiry": "",
        "broker": "7024a621c7c60bf3",
        "x": 660,
        "y": 560,
        "wires": []
    },
    {
        "id": "413889ad4e92667a",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "function 1",
        "func": "msg.payload = {\n    r: msg.payload.r,\n    g: msg.payload.g,\n    b: msg.payload.b\n};\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 400,
        "y": 560,
        "wires": [
            [
                "e0d7ad32ddce16e1"
            ]
        ]
    },
    {
        "id": "ba76eae2c3f40229",
        "type": "ui_numeric",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "label": "Update Interval (ms)",
        "tooltip": "",
        "group": "6d308aaa790616ec",
        "order": 1,
        "width": 0,
        "height": 0,
        "wrap": false,
        "passthru": true,
        "topic": "",
        "topicType": "str",
        "format": "{{value}}",
        "min": "500",
        "max": "60000",
        "step": "500",
        "className": "",
        "x": 220,
        "y": 660,
        "wires": [
            [
                "3ccccae15865ccf4"
            ]
        ]
    },
    {
        "id": "3ccccae15865ccf4",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "function 2",
        "func": "msg.payload = { interval: msg.payload };\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 440,
        "y": 660,
        "wires": [
            [
                "5cb8fa565202fef6"
            ]
        ]
    },
    {
        "id": "5cb8fa565202fef6",
        "type": "mqtt out",
        "z": "7bd6a5989f03b3da",
        "name": "Interval Command",
        "topic": "iot/cmd/interval",
        "qos": "",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correl": "",
        "expiry": "",
        "broker": "7024a621c7c60bf3",
        "x": 670,
        "y": 660,
        "wires": []
    },
    {
        "id": "9541053411a4e96f",
        "type": "influxdb out",
        "z": "7bd6a5989f03b3da",
        "influxdb": "2a1bfdc60dea8c69",
        "name": "DB",
        "measurement": "",
        "precision": "",
        "retentionPolicy": "",
        "database": "database",
        "precisionV18FluxV20": "ms",
        "retentionPolicyV18Flux": "",
        "org": "iot-lab",
        "bucket": "esp32_data",
        "x": 1030,
        "y": 40,
        "wires": []
    },
    {
        "id": "8ce3092787afa5fd",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "To-DB",
        "func": "msg.measurement = \"esp32_telemetry\";\nmsg.payload = {\n  temperature: msg.payload.tempC,\n  humidity: msg.payload.humidity,\n  wifiRssi: msg.payload.wifiRssi_dBm\n};\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 710,
        "y": 40,
        "wires": [
            [
                "9541053411a4e96f"
            ]
        ]
    },
    {
        "id": "ab229f04d3b01085",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "Range-Extractor",
        "func": "msg.payload = msg.payload.wifiRssi_dBm;\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 740,
        "y": 140,
        "wires": [
            [
                "31c10ad63cb4bca4",
                "bd9c35ccdb3146b9"
            ]
        ]
    },
    {
        "id": "31c10ad63cb4bca4",
        "type": "ui_gauge",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "group": "bc4dae18beef13e6",
        "order": 0,
        "width": 0,
        "height": 0,
        "gtype": "gage",
        "title": "Current Wifi Range (dBm)",
        "label": "units",
        "format": "{{value}}",
        "min": "-200",
        "max": "0",
        "colors": [
            "#ff0000",
            "#eeff00",
            "#52cb3a"
        ],
        "seg1": "",
        "seg2": "",
        "diff": false,
        "className": "",
        "x": 1090,
        "y": 140,
        "wires": []
    },
    {
        "id": "bd9c35ccdb3146b9",
        "type": "ui_chart",
        "z": "7bd6a5989f03b3da",
        "name": "",
        "group": "bc4dae18beef13e6",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Wifi Range (dBm)",
        "chartType": "line",
        "legend": "false",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "className": "",
        "x": 1070,
        "y": 100,
        "wires": [
            []
        ]
    },
    {
        "id": "4fe823c0ea7ce615",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "Temp-Alert",
        "func": "const threshold = 40;  // °C alert threshold\nconst hysteresis = 1;     // prevents flickering\nconst temp = msg.payload;\n\nlet state = context.get(\"state\") || \"normal\";\nlet output1 = null; \nlet output2 = null; \n\nif (temp > threshold && state !== \"alert\") {\n    context.set(\"state\", \"alert\");\n    output1 = {\n        payload: `🔥 ALERT: Temperature too high! (${temp.toFixed(1)} °C)`,\n        color: \"red\"\n    };\n}\nelse if (temp < (threshold - hysteresis) && state !== \"normal\") {\n    context.set(\"state\", \"normal\");\n    output2 = {\n        payload: `No Alerts`,\n        color: \"green\"\n    };\n}\nreturn [output1, output2];",
        "outputs": 2,
        "timeout": "0",
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 930,
        "y": 340,
        "wires": [
            [
                "6d41d56d9a6a4f86"
            ],
            [
                "6d41d56d9a6a4f86"
            ]
        ]
    },
    {
        "id": "847467cb06ba5142",
        "type": "function",
        "z": "7bd6a5989f03b3da",
        "name": "Humidity-Alert",
        "func": "const high = 60;   // upper threshold\nconst low = 35;  // lower threshold\nconst hysteresis = 2;\nconst hum = msg.payload;\n\nlet state = context.get(\"state\") || \"normal\";\nlet alert = null;\nlet cleared = null;\n\nif (hum > high && state !== \"high\") {\n    context.set(\"state\", \"high\");\n    alert = {\n        payload: `⚠️ High humidity detected (${hum.toFixed(1)}%)!`,\n        color: \"orange\"\n    };\n}\nelse if (hum < high - hysteresis && state === \"high\") {\n    context.set(\"state\", \"normal\");\n    cleared = {\n        payload: `No Alerts`,\n        color: \"green\"\n    };\n}\nelse if (hum < low && state !== \"low\") {\n    context.set(\"state\", \"low\");\n    alert = {\n        payload: `💧 Low humidity detected (${hum.toFixed(1)}%)!`,\n        color: \"blue\"\n    };\n}\nelse if (hum > low + hysteresis && state === \"low\") {\n
```

```
context.set(\"state\",
\"normal\");\n      cleared =
{\n          payload: `No
Alerts`,\n              color:
\"green\"\n    };\n}\n\nreturn
[alert, cleared];",
       "outputs": 2,
       "timeout": 0,
       "noerr": 0,
       "initialize": "",
       "finalize": "",
       "libs": [],
       "x": 940,
       "y": 540,
       "wires": [
           [

"09c0d4a5eefc5dac"
           ],
           [

"09c0d4a5eefc5dac"
           ]
       ]
    },
    {
       "id":
"6d41d56d9a6a4f86",
       "type": "ui_text",
       "z":
"7bd6a5989f03b3da",
       "group":
"605056de7193c6f0",
       "order": 2,
       "width": 0,
       "height": 0,
       "name": "",
       "label":          "System
Alert",
       "format":
"{{msg.payload}}",
       "layout": "row-center",
       "className": "",
       "style": false,
       "font": "",
       "fontSize": 16,
       "color": "#000000",
       "x": 1130,
       "y": 340,
       "wires": []
    },
    {
       "id":
"09c0d4a5eefc5dac",
       "type": "ui_text",
       "z":
"7bd6a5989f03b3da",
       "group":
"ea259cb173e4a95b",
       "order": 2,
       "width": 0,
       "height": 0,
       "name": "",
       "label":          "System
Alert",
       "format":
"{{msg.payload}}",
       "layout": "row-center",
       "className": "",
       "style": false,
       "font": "",
       "fontSize": 16,
       "color": "#000000",
       "x": 1150,
       "y": 540,
       "wires": []
    },
    {
       "id":
"5b68351e31537d98",
       "type": "inject",
       "z":
"7bd6a5989f03b3da",
       "name":         "Startup
Restore",
       "props": [
           {
               "p": "payload"
           },
           {
               "p": "topic",
               "vt": "str"
           }
       ],
       "repeat": "",
       "crontab": "",
       "once": true,
       "onceDelay": "5",
       "topic": "",
       "payload": "",
       "payloadType": "date",
       "x": 160,
       "y": 60,
       "wires": [
           [

"ecc73103213ee1d4"
           ]
       ]
    },
    {
       "id":
"ecc73103213ee1d4",
       "type": "influxdb in",
       "z":
"7bd6a5989f03b3da",
       "influxdb":
"2a1bfdc60dea8c69",
       "name": "DB-In",
       "query": "from(bucket:
\"esp32_data\")\n          |>
range(start:    -24h)\n    |>
filter(fn:     (r)     =>   \n
r._measurement            ==
\"esp32_telemetry\" and \n
(r._field == \"temperature\"
or r._field == \"humidity\"
or        r._field         ==
\"wifiRssi\"))\n  |> last()",
       "rawOutput": false,
       "precision": "",
       "retentionPolicy": "",
       "org": "iot-lab",
       "x": 330,
       "y": 60,
       "wires": [
           [

"6ffa6f702fdb0cad"
           ]
       ]
    },
    {
       "id":
"6ffa6f702fdb0cad",
       "type": "function",
       "z":
"7bd6a5989f03b3da",
       "name": "Filter",
       "func":  "let  data  =
msg.payload;\nif
(!Array.isArray(data)       ||
data.length  ===  0)  return
null;\n\nlet  result  =  {\n
temperature:            null,\n
humidity: null,\n   wifiRssi:
null,\n            timestamp:
null\n};\n\nfor (let point of
data)    {\n          switch
(point._field) {\n        case
\"temperature\":\n
result.temperature          =
parseFloat(point._value.toF
ixed(2));\n            break;\n
case          \"humidity\":\n
result.humidity             =
parseFloat(point._value.toF
ixed(2));\n            break;\n
case            \"wifiRssi\":\n
result.wifiRssi             =
parseFloat(point._value.toF
ixed(2));\n            break;\n
}\n\n         if   (point._time)
result.timestamp            =
point._time;\n}\n\nmsg.pay
load = result;\nreturn msg;",
       "outputs": 1,
       "timeout": 0,
       "noerr": 0,
       "initialize": "",
       "finalize": "",
       "libs": [],
       "x": 250,
       "y": 160,
       "wires": [
           [

"53fe83c1faa2e2c8"
           ]
       ]
    },
    {
       "id":
"53fe83c1faa2e2c8",
       "type": "function",
       "z":
"7bd6a5989f03b3da",
       "name":    "Restore-to-
Dashboard",
       "func":   "let   d   =
msg.payload;\n\nmsg.topic
=
\"iot/telemetry\";\nmsg.payl
oad = {\n     deviceLabel:
\"esp32\",\n           ts_ms:
Date.now(),\n          tempC:
d.temperature,\n   humidity:
d.humidity,\n
wifiRssi_dBm:
d.wifiRssi\n};\n\nreturn
msg;",
       "outputs": 1,
       "timeout": 0,
       "noerr": 0,
       "initialize": "",
       "finalize": "",
       "libs": [],
       "x": 460,
       "y": 160,
       "wires": [
           [

"273fd460ce0fa331"
           ]
       ]
    },
    {
       "id":
"7024a621c7c60bf3",
       "type": "mqtt-broker",
       "name": "",
       "broker": "127.0.0.1",
       "port": 1883,
       "clientid": "",
       "autoConnect": true,
       "usetls": false,
       "protocolVersion": 4,
       "keepalive": 60,
       "cleansession": true,
       "autoUnsubscribe":
true,
       "birthTopic": "",
       "birthQos": "0",
       "birthRetain": "false",
       "birthPayload": "",
       "birthMsg": {},
       "closeTopic": "",
       "closeQos": "0",
        "closeRetain": "false",
        "closePayload": "",
        "closeMsg": {},
        "willTopic": "",
        "willQos": "0",
        "willRetain": "false",
        "willPayload": "",
        "willMsg": {},
        "userProps": "",
        "sessionExpiry": ""
    },
    {
        "id":
"605056de7193c6f0",
        "type": "ui_group",
        "name":            "ESP-
Temperature",
        "tab":
"66d4dd7983f026bb",
        "order": 1,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id":
"ea259cb173e4a95b",
        "type": "ui_group",
        "name":            "ESP-
Humidity",
        "tab":
"66d4dd7983f026bb",
        "order": 2,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id":
"6d308aaa790616ec",
        "type": "ui_group",
        "name":            "ESP-
Controls",
        "tab":
"66d4dd7983f026bb",
        "order": 3,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id":
"2a1bfdc60dea8c69",
        "type": "influxdb",
        "hostname":
"127.0.0.1",
        "port": 8086,
        "protocol": "http",
        "database": "database",
        "name": "DB",
        "usetls": false,
        "tls": "",
        "influxdbVersion":
"2.0",
        "url":
"http://localhost:8086",
        "timeout": 10,
        "rejectUnauthorized":
false
    },
    {
        "id":
"bc4dae18beef13e6",
        "type": "ui_group",
        "name":     "ESP-Wifi-
Range",
        "tab":
"66d4dd7983f026bb",
        "order": 4,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id":
"66d4dd7983f026bb",
        "type": "ui_tab",
        "name":   "ESP32   IoT
Dashboard",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    },
    {
        "id":
"c6702123cc485abf",
        "type":           "global-
config",
        "env": [],
        "modules": {
            "node-red-
dashboard": "3.6.6",
            "node-red-contrib-
influxdb": "0.7.0"
        }
    }
]
```

APPENDIX C: INFLUXDB QUERY RESULTS

```
Result: _result
Table: keys: [_start, _stop, _field, _measurement]
                 _start:time                     _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:14.555000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:19.632000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:24.576000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:29.601000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:34.713000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:39.731000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:44.650000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:49.562000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:54.565000000Z                            47
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              humidity          esp32_telemetry  2025-11-10T04:53:59.566000000Z                            47
Table: keys: [_start, _stop, _field, _measurement]
                 _start:time                     _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:14.555000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:19.632000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:24.576000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:29.601000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:34.713000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:39.731000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:44.650000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:49.562000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:54.565000000Z                          27.6
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z           temperature          esp32_telemetry  2025-11-10T04:53:59.566000000Z                          27.6
Table: keys: [_start, _stop, _field, _measurement]
                 _start:time                     _stop:time           _field:string     _measurement:string                      _time:time                  _value:float
------------------------------  ------------------------------  ----------------------  ----------------------  ------------------------------  ----------------------------
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:14.555000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:19.632000000Z                           -61
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:24.576000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:29.601000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:34.713000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:39.731000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:44.650000000Z                           -59
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:49.562000000Z                           -60
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:54.565000000Z                           -61
2025-11-10T04:53:12.668153000Z  2025-11-10T05:53:12.668153000Z              wifiRssi          esp32_telemetry  2025-11-10T04:53:59.566000000Z                           -60
```