Implementation of Cloud-Connected IoT system using ESP32 Doit and Node-red

Neelash Kannan Annadurai HWU Elect Electronic & Comp Eng Heriot Watt University

Edinburgh, United Kingdom

na3018@hw.ac.uk

*Abstract*—This project shows the creation of an IoT system integrated with cloud using ESP8266 microcontroller, MQTT for message exchange protocol and Node-RED for server-side data processing and Control design with MongoDB as the Database solution. The project is divided into four sections. The first step creates the development environment, where the ESP8266 is configured to join a local WiFi network to control basic hardware. The second part introduces MQTT protocol to provide efficient, minimalistic and securedata communication to the cloud with additional optional steps incorporating SSL/TLS security in case of necessity. During the third phase, Node-RED is used to set up an interactive, dynamic dashboard of the current values of the-gathering sensors and controlling the ESP8266 and its connected LED. Last is the fourth section this is about data analytics of accumulated IoT data through MongoDB to demonstrate the controlling capabilities of IoT in creating intelligent and adaptive spaces. The working video of this project is attached with the github repo as mentioned in reference [10].

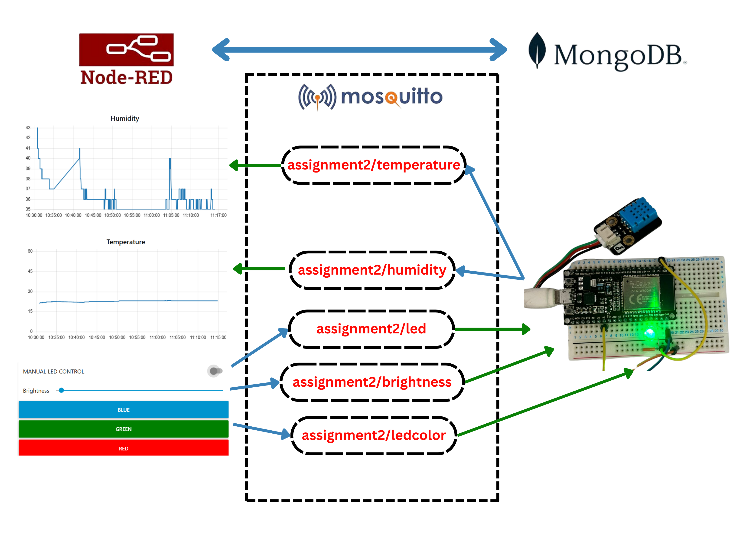
Keywords— Internet of Things (IoT), ESP8266 microcontroller, MQTT protocol, Node-RED, MongoDB, cloud connectivity, data visualization, remote control, SSL/TLS security, data analytics, IoT dashboard, sensor data, low-code development, smart environments.

# Introduction

The Internet of Things (IoT) redefines the relations between human and physical object that can send and receive data via the web and allow access to information anytime, anyplace. This capability enables IoT devices to gather, exchange and process data away from a central hub, providing use cases across smart residences, health, farming, and production. This project also involves IoT system having the Espressif ESP32 DoIT microcontrollers, DHT11 temperature and humidity sensors, multicolor LED, the MQTT for information exchange, and Node-RED for designing the GUI. The ESP32 microcontroller acts as the heart where it can capture sensor data and then broadcast it over the local WiFi network with the help of multicolor LED to facilitate interactions with the user. MQTT, a lightweight messaging protocol optimized for IoT, supports efficient data transmission in low-bandwidth environments [2]. Node-RED, a low-code tool, allows for easy development of a web-based dashboard that visualizes sensor data and provides controls for device configuration [1]. This IoT solution comprises three key steps: An ESP32 board setup, connecting the board to the WiFi network, setting up data communication with the cloud through MQTT; and, generating a dynamic dashboard using Node-RED. They can check the temperature and humidity from DHT11 sensor and change the state of the LED from anywhere with the help of internet connection. In fact, it exemplifies how IoT works from the basic setup of devices, cloud connectivity, as well as usability to make adaptable IoT on an appropriately large scale. This work explains the way MQTT and Node-RED can be employed in the efficient management of data while bringing IoT interfaces within easy reach; this reveals the concept of IoT in the development of enhanced data-driven applications.

# Network Architecture

Network architecture for this system is developed to allow for real-time operations and interaction between the microcontroller, the cloud platform as well as the user interface. It is needed for data acquiring form the sensors and to regulate the HW components in a local network manner. The MQTT (Message Queuing Telemetry Transport) protocol is the system’s foundational protocol that is utilized in IoT systems due to its low bandwidth and low power consumption [4]. This architecture’s key components are the microcontroller, ESP32, the MQ-TT broker, and Node-RED for data display and easy interaction with the system. The communication between the microcontroller and the cloud platform is done using the MQTT topics Since data can be published from the microcontroller to the broker as well as received from the broker to the microcontroller as the case maybe. If to describe the architecture schematically, it will look as follows, with an emphasis on components’ interaction as well as topics involved into the process, as shown in the Fig. 2.1.



*Fig.2.1.Network Architecture [10]*

## Device to Cloud:

The microcontroller is connected to a local wifi to establish a connection between the board and the MQTT broker. I used these Arduino Libraries, “PubSubClient” and “WiFi.h” to achieve this. I Created three different topics “assignment2/temperature” and “assignment2/humidity” to publish the sensor data and “assignment2/led” to subscribe for led control from node-red.

## Cloud Broker – Mosquitto MQTT:

The MQTT cloud broker is responsible for data transfer between the microcontroller and Node-Red dashboard. Here I tried with hive and few other public servers due to firewall issue I’m not able to connect to these. So, I use mosquito MQTT which can installed locally and first I tested it with publishing and subscribing it from two individual command prompts. Then I used the command “ipconfig” in my cmd to find out my IP Address which is used as my MQTT Server address.

## Node-Red:

Node-RED is a flow-based development tool for visual programming, designed to connect devices and services with ease. It provides a user-friendly interface for wiring together hardware devices, APIs, and online services [1]. To Install this I used “npm install -g node-red”. Once it’s installed use the command “node-red’ to run it and the http address would be seen in the command prompt once it starts. The dashboard for this is not found in default so it need to be installed using the manage pallete.

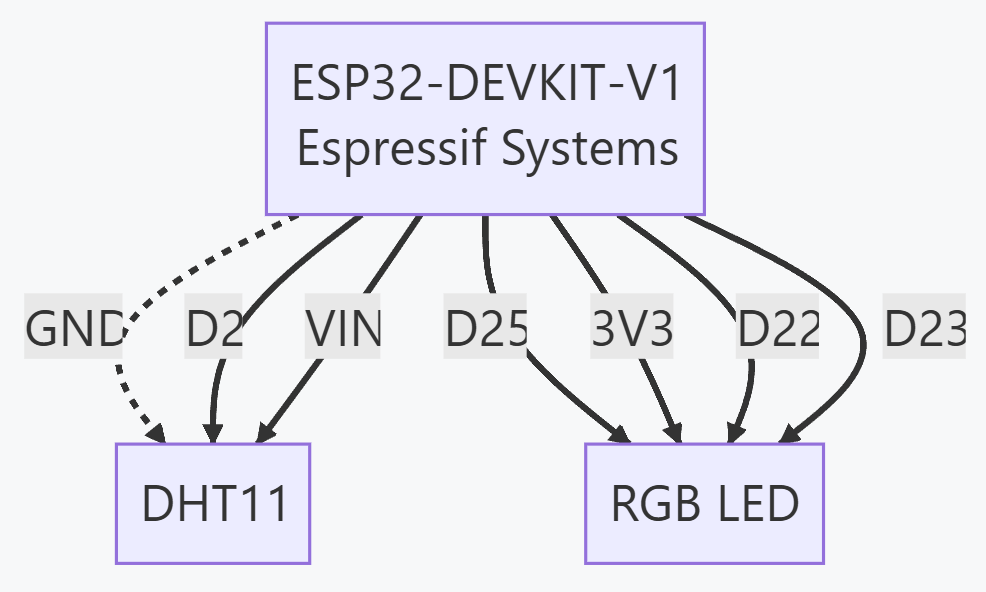
## MongoDB:

MongoDB plays a crucial role in this architecture by acting as the database for storing sensor data in a structured format. MongoDB is suitable when there are

numerous insert queries that have to be processed by the DBMS [5]. Data received from the microcontroller, including temperature, humidity is stored in MongoDB for long-term retention and analysis. The database allows for efficient querying of historical data, and it is integrated with the Node-RED platform through a MongoDB node. This integration enables seamless storage and retrieval of IoT data, helping users track changes over time. The sensor readings are stored in collections and can be accessed or analyzed for insights into environmental conditions, system performance, and user interactions.

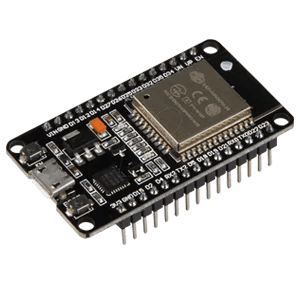
# Hardware Overview

The Hardware architecture of this IoT system is mainly depended on the Espressif ESP32 DoIT microcontroller, The ESP32, a cost-effective yet powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities, serves as the core component of our system, enabling flexible and scalable deployment [1]. This microcontroller serves as the core processing unit, managing both the collection of data from the sensor and have the control of IoT System functionalities. It Supports various communication protocols which makes it well-suited for this project. Fig.3.1 is the circuit diagram of the hardware system.



*Fig.3.1 Circuit Connection of Microcontroller to sensor and RGB LED [10]*

## Espressif ESP32 DoIT Devkit:

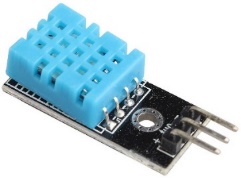


*Fig.3.2 Espressif ESP32 DoIT Dev Kit [10]*

This microcontroller mentioned in Fig.2.1 is used for

connecting the local Wi-Fi network to do data exchange between the device and the cloud platform via the MQTT protocol. The microcontroller operates at 80 MHz and has 38 GPIO pins which are used to interface the sensor and the LED.

## Sensor:



*Fig.3.3 DHT11 Temperature and Humidity sensor [10]*

This system uses DHT11 temperature and humidity sensor shown in Fig3.3. The DHT11 is a cost-effective digital sensor that measures temperature and humidity accurately [1]. This sensor is connected to the microcontroller via GPIO pins and uses single-wire protocol. The major advantage of this sensor is the accuracy and the low power consumption.

## RGB – LED:



*Fig.3.4 RGB LED diode [10]*

A Multicolor LED as shown in Fig.3.4 is connected to the microcontroller via GPIO pins. The RGB LED have three channels (Red, Green, Blue) where the intensity is adjusted by varying the PWM signal. This used to indicate the temperature changes like if there is higher temperature red color is seen and green if the temperature is normal. Even the color can be controlled by the Node-Red dashboard.

## Power Supply:

The power requirement for the microcontroller is 5v and the micro-usb port provided in the board helped the way that it is more than enough for the sensor and the LED to work.

# Software Overview

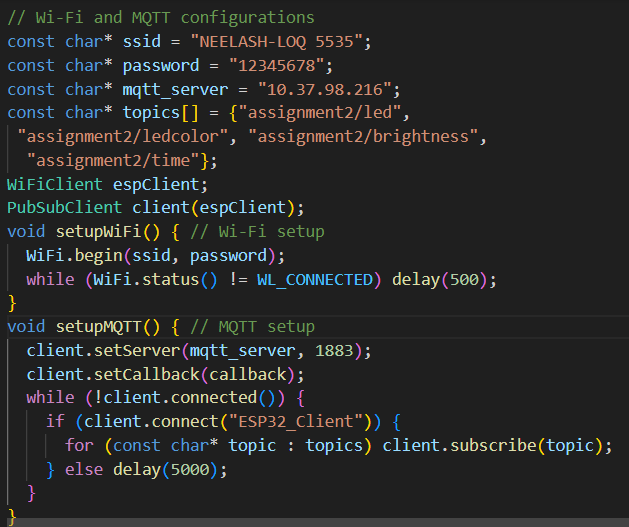
The Software implementation for this project is done with the help of “Platform IO” , an IDE extension for VS code. This is used to for programming the microcontroller.

## PlatformIO Setup:

As said in [6] PlatformIO is can be installed by adding an extension in the VS code. Once it’s done restart the vs code so that we can find the platformio home icon in the right bottom of the vs code which routes to the home page of platformio. Creating a new projects pops up a window showing options to select the board, framework and name the project. Here comes the best part of platformio where it installs all the required libraries and drivers for the selected microcontroller. While creating the project there would be a delay of ten to fifteen minutes. I used three libraries for this project “WiFi.h” to connect microcontroller to my laptop hotspot, “PubSubClient” to connect my microcontroller and MQTT broker. “DHT.h” to read temperate and humidity input from DHT11 sensor.

## Connecting Microcontroller to MQTT:

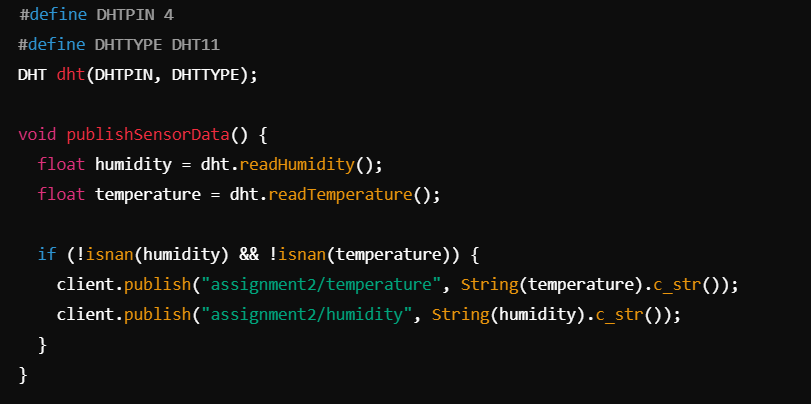
The initial part of the code as mentioned in Fig 4.1 is dedicated to establishing a WiFi connection. The ESP8266 connects to a specified WiFi network using the “WiFi.begin(ssid, password)” function, which continuously attempts to connect until a connection is established. Once connected, the MQTT client is initialized and configured to connect to the cloud-based MQTT broker. I used a mosquito MQTT broker running it locally in my laptop. The code in Fig.4.1 explains the connection between the local wifi and MQTT broker to microcontroller.



*Fig.4.1Wifi and MQTT Server connection to ESP32 [10]*

## Sensor Data Collection:

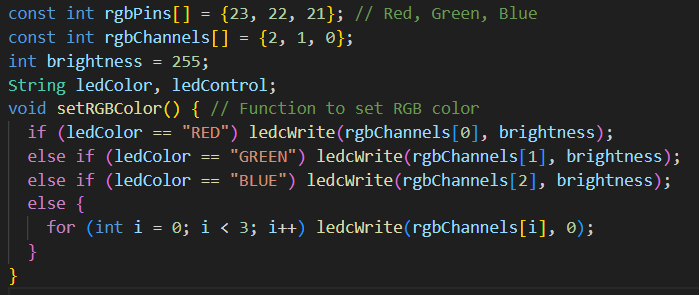
The DHT sensor is connected in such a way that the data are collected from it in certain intervals. These temperature and humidity data are published with the help of these topics, “assignment2/temperature” and “assignment2/humidity”. The data is sent to MQTT broker with the help of “publish()” function. The code Fig.4.2 explains the data publishing from the microcontroller to MQTT broker.



*Fig.4.2 Data collection from dht11 [10]*

## RGB LED Control:

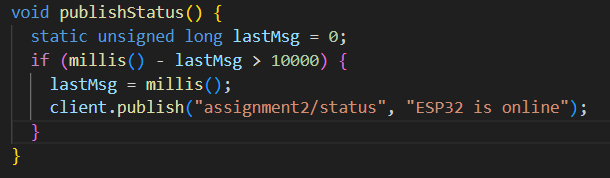
To control RGB LED, microcontroller is subscribed to a topic “assignment2/control/led”, where the node-red publishes command to change LED color. The code in Fig.4.3 explain how the microcontroller is subscribing to a topic and receiving the command to control LED.



*Fig.4.3. RGB-LED Control [10].*

## Status Messaging:

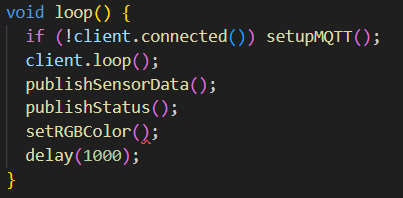
To monitor the device status, as status message of “ESP32 is Online” is published to the topic “assignment2/status” for every 10 seconds as mentioned in Fig.4.4. This message indicates that the microcontroller is connected and actively monitoring the environment.



*Fig.4.4 Microcontroller status monitoring [10].*

## Error Handling and Reconnection:

If the connection between board and wifi or board and MQTT gets disconnected, the microcontroller tries to reconnect again and again as mentioned in Fig.4.5. This is essential for maintaining the communication reliability.

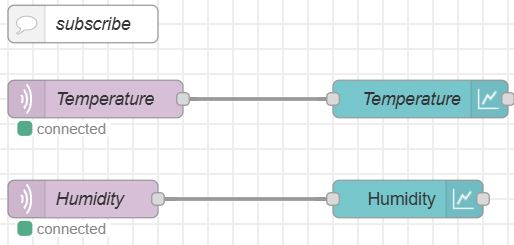


*Fig.4.5 Error Handling [10].*

# Node-Red Setup

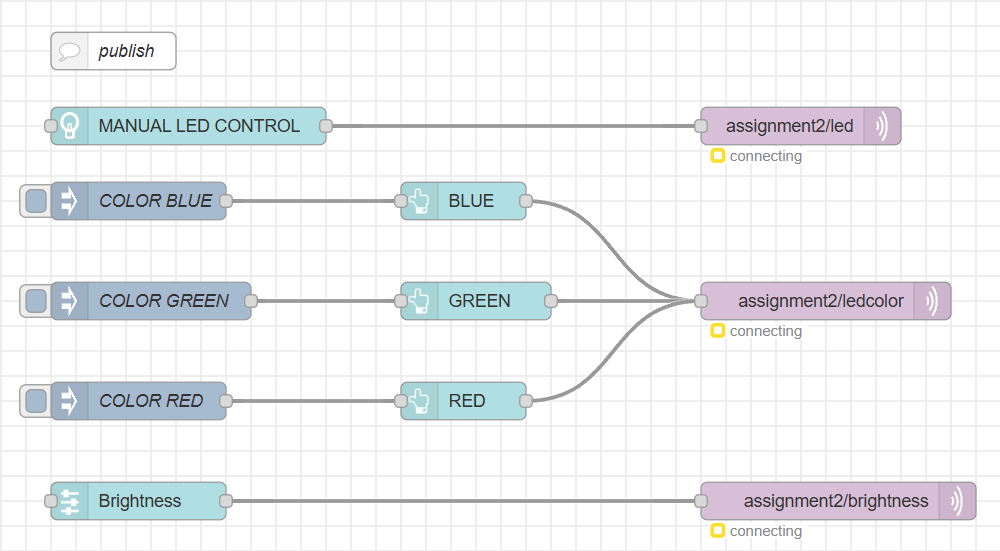
Node-Red provides ease for connections by just drawing wires or links and adding parameters to them. The flow diagrams are self-explanatory [3]. In this Assignment there are major five topics where two are subscribed by the node-red and three are published by the node-red.

Lets consider the Fig.5.1 it shows the flow of the topic “assignment2/temperature” and “assignment2/humidity”. I connected the “MQTT in” node, which connects to MQTT broker and subscribe to messages from specific topic to chart node which plots the input data in a graph format, Similarly to the gauge node for better visualization.



*Fig.5.1 MQTT Subscription with topics published by Microcontroller [10].*

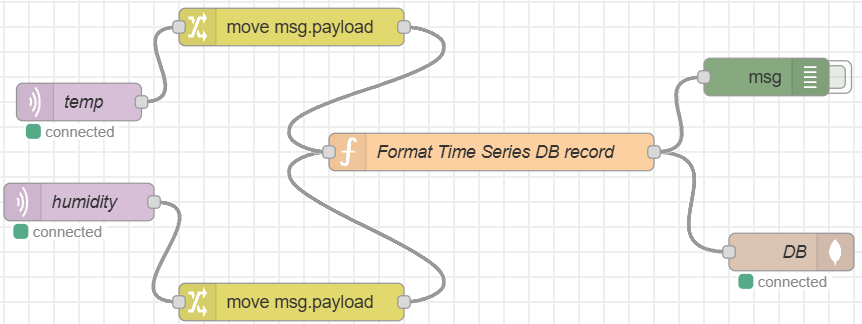
The following scenario in Fig.5.2 shows flow of the topics “assignment2/led”, “assignment2/ledcolor” and ‘assignment2/brightness”. These are topics which are being published by the Node-Red or MQTT Broker to the microcontroller. In this I connected “Button” node, which adds button to user interface to “MQTT out” node, which connects to MQTT broker and sends message. So that when I click the button the data is published. This is used to turn on the manual control for the led. Next I connected the “Inject” node which is used to inject a message into the flow either manually or at a regular interval, to the “button node” and to the “MQTT out”. Here the messages for changing colour of the RGB LED is sent from the Broker. The last one is “scroll node”, which is used to add scroller to the dashboard UI, to “MQTT out” node to send messages to control intensity of the RGB LED.



*Fig.5.2.MQTT Broker publishing topics to*

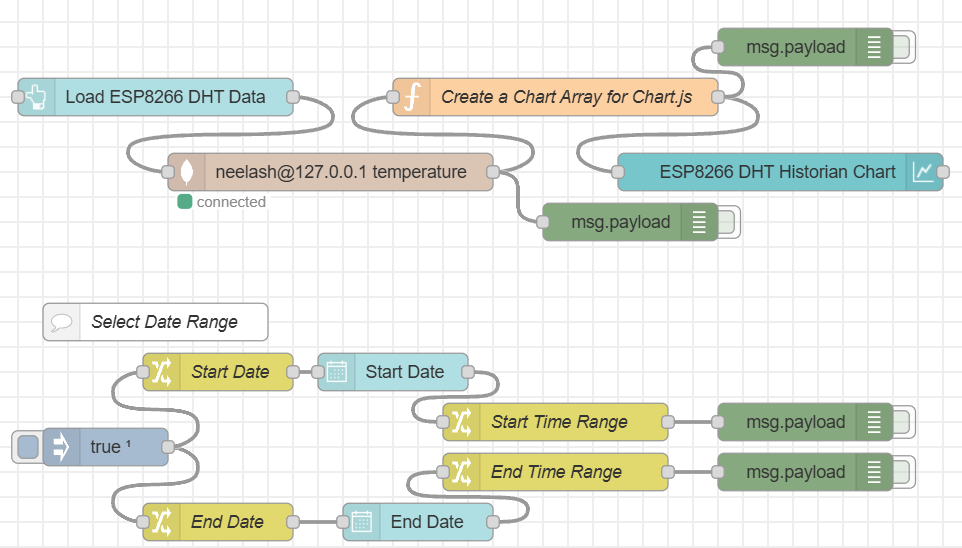
*Microcontroller [10]*

In Fig.5.3 Node-RED flow collects temperature and humidity data from respective sensor nodes, processes the data by extracting it into the message payload using the "move msg.payload" nodes, and then formats the data into a time-series record within a function node. The function node is responsible for structuring the data, possibly adding a timestamp or reorganizing it for compatibility with a time-series database. Finally, the formatted data is sent to a database node, where it is stored for future analysis, allowing for efficient querying and monitoring of environmental conditions over time.



*Fig.5.3 MQTT Broker using node-red to send data to Database [10]*.

In Fig.5.4 Node-RED flow is meant to fetch from a cloud MongoDB data base and display a temperature data that arrived through an ESP8266 probably employing a DHT sensor. The flow starts by reading the temperature data from the cloud MongoDB database where it resides and converts the data into format that can be plotted on the chart using a function node. The chart is then plotted on a historical chart using Chart.js button. The data can be filtered easily through an interface wherein users can choose a particular date time limit. With the selected start and end dates, and start and end times enabled, the cloud MongoDB data base is queried to obtain the specified range of temperature between the chosen start and end dates for flexible control of historical temperature trends visualization.



*Fig.5.4 Fetching data from MongoDB for history graph [10].*

# Result

The IoT system, which has been implemented for this project, can indeed integrate ESP32 microcontroller, MQTT protocol, Node-RED for the graphical representation of the IoT system and MongoDB for the storage and management of big data in cloud-connected environment. The system achieved the following key outcomes:

## Data Communication and Device Control:

The serial communication with the ESP32 microcontroller to the MQTT broker (Mosquitto) was achieved. The ESP32 was able to scan the local WiFi network and connect to it with the due credentials, publish the temperature and humidity as well as subscribe to commands for the connected RGB LED. The DHT11 sensor data was acquired and transmitted effectively in the every 10 sec and both temperature and humidity values were shown on the Node-RED incoming dashboard. The system allowed the control of the RGB LED wirelessly; it took color commands and brightness from the Node-RED application.

## Node-RED Dashboard Performance:

As said in [8] The Node-RED dashboard offered an easy and interactive means whereby the user could view pilot environmental data and manage the system from a distance. Some of the concepts that were implemented on an Infotainment system dashboard were graphical temperature and humidity display and the gauge temperature and humidity display. Besides, the client needed to manage the RGB LED colour and intensity, which served as an indication of the IoT system interaction. The “inject” and “button” nodes in Node-RED made sure that all commands from the user interface where relayed properly to the microcontroller to change LED parameters. The time taken to respond to these interactions was also poor, indicating effective real time control.

## Data Storage and Analytics:

MongoDB is a document-oriented database system classified as a NoSQL database system [7]. This integration with MongoDB made it convenient to store and update depending on the various sensors used. Temperature and humidity data logged by the ESP32 were stored into specific MongoDB collections and could be reconstructed for analysis purposes. By employing Node-RED flow, the required historical data was fetched from the Database and data was visualized on Node-RED Dashboard. The temperature data acquisition could be done historically and could be filtered by time intervals and visualized enabling the user to look at the trends in the temperature at specified intervals of time. This data analytics functionality demonstrates how IoT systems can create adaptive and smart environments using the past sensor data.

## System Reliability and Error Handling:

It also showed the right performance for continuous data publishing, LED control as well as data retrieval. Controlling measures were put in place to guarantee that the system stays functional in situation where disconnections occur. The microcontroller was also programmed to check for the connection of the WiFi and the MQTT broker in case of disconnection and then proceed to reconnect in order to carry out its functions continuously. The reliability of the system was also tested and verified over several cycles with demonstrated capability of the microcontroller of publishing sensor data and receiving control commands. The reliability of the system was also tested and verified over several cycles with demonstrated capability of the microcontroller of publishing sensor data and receiving control commands as of the reliability tests done in [9].

## Scalability and Future Work:

It also showed the right performance for continuous data publishing, LED control as well as data retrieval. Controlling measures were put in place to guarantee that the system stays functional in situation where disconnections occur. The microcontroller was also programmed to check for the connection of the WiFi and the MQTT broker in case of disconnection and then proceed to reconnect in order to carry out its functions continuously.

# Conclusion

In this project, we have realized and implemented an IoT cloud system that can be connected , controlled and monitored through the ESP32 DoIT microcontroller, MQTT protocol, Node-RED, and MongoDB as the data store and analyser. The system also shows how individual hardware components can be connected to the cloud and modified in real time with real time illustrated in the form of a graph and can be controlled remotely through an intuitive graphical user interface. By using MQTT, we remained generous and unobtrusive in terms of utilizing the bandwidth because it allowed us to switch data on temperature and humidity from the ESP32 microcontroller to the cloud platform. This engagement was made possible through the Node-RED application where there was a convenient way of developing an interactive dashboard to help three users to monitor the data sensed and control the system from a remote place. Moreover, storing the data in the MongoDB database also helped in further analysing it to determine the conditions outside. The unawareness of the hardware and software components, the freedom of the MQTT protocol and the low-code characteristics of Node-RED allow for the building small to large IoT applications with ease. This project also demonstrates the significance of using cloud connectivity and data analytics to bring raw sensor information into useful outcomes while underlining the use of IoT systems in making intelligent environment adaptive system. These technologies can provide a means to opening up new areas of using such systems at homes, in the areas of health care, horticulture, factory floors, etc. This system shows a sound initial design of IoT smart devices and its integration with cloud environment and hence we believe it provides a platform for future development and expansion of the concept.

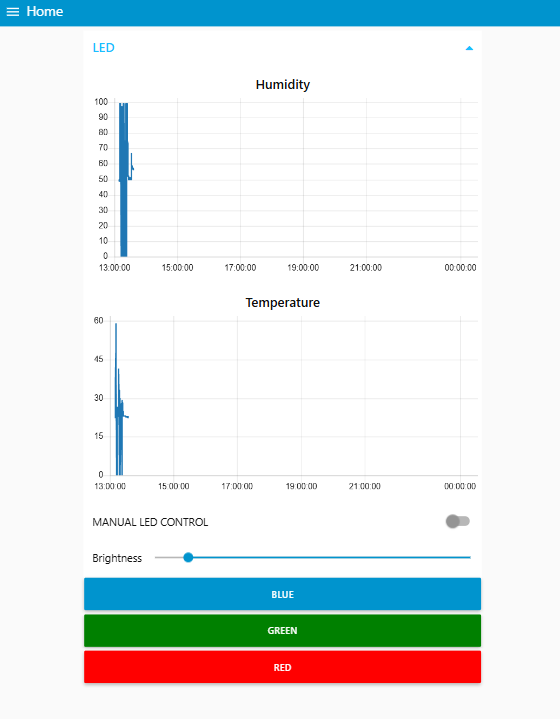
##### References

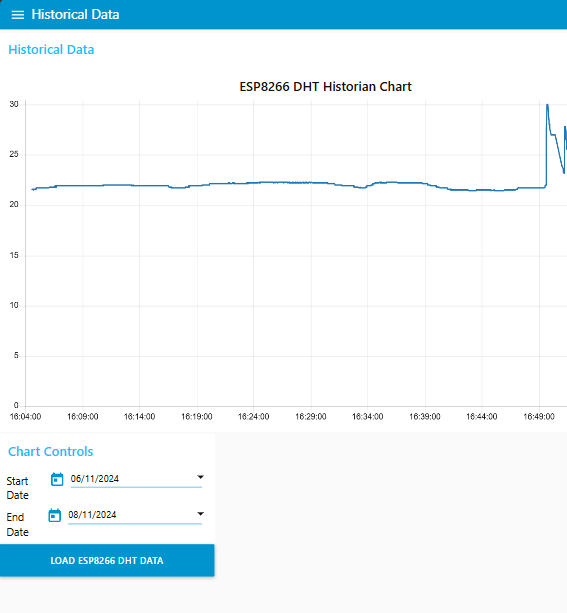
1. A. Kumar Arigela, C. Banapuram and N. Venu, "Remote based Home Automation with MQTT: ESP32 Nodes and Node-RED on Raspberry Pi," 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Kirtipur, Nepal, 2024, pp. 313-318, doi: 10.1109/I-SMAC61858.2024.10714655.
2. N. H. Motlagh, M. Bagaa and T. Taleb, "UAV-Based IoT Platform: A Crowd Surveillance Use Case," in IEEE Communications Magazine, vol. 55, no. 2, pp. 128-134, February 2017, doi: 10.1109/MCOM.2017.1600587CM.
3. A. Rajalakshmi and H. Shahnasser, "Internet of Things using Node-Red and alexa," 2017 17th International Symposium on Communications and Information Technologies (ISCIT), Cairns, QLD, Australia, 2017, pp. 1-4, doi: 10.1109/ISCIT.2017.8261194.
4. P. S. B. Macheso, T. D. Manda, A. G. Meela, J. S. Mlatho, G. T. Taulo and B. M'mame, "Environmental Parameter Monitoring System Based on NodeMCU ESP8266, MQTT and Node-RED," 2022 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2022, pp. 1-4, doi: 10.1109/ICCCI54379.2022.9740787.
5. T. Mladenova and I. Valova, "Performance Study of MySQL and MongoDB for IoT Data Processing and Storage," 2022 International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2022, pp. 60-63, doi: 10.1109/ICAI55857.2022.9960134.
6. M. S. Abdul, S. M. Sam, N. Mohamed, N. H. Hassan, A. Azizan and Y. M. Yusof, "Peer to Peer Communication for the Internet of Things Using ESP32 Microcontroller for Indoor Environments," 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Republic of, 2022, pp. 1-6, doi: 10.1109/ICTC55196.2022.9952832.
7. C. Rattanapoka, S. Chanthakit, A. Chimchai and A. Sookkeaw, "An MQTT-based IoT Cloud Platform with Flow Design by Node-RED," 2019 Research, Invention, and Innovation Congress (RI2C), Bangkok, Thailand, 2019, pp. 1-6, doi: 10.1109/RI2C48728.2019.8999942.
8. A. Rachid and A. Djedjig, "IoT and MQTT based web monitoring of a solar living laboratory," 2022 2nd International Conference on Digital Futures and Transformative Technologies (ICoDT2), Rawalpindi, Pakistan, 2022, pp. 1-6, doi: 10.1109/ICoDT255437.2022.9787471.
9. B. Goyal, K. K. Dixit, A. Dogra, M. Nagar, S. V. Akram and J. Kaur, "Empowering Assets and Vehicles with Cutting-Edge ESP32 Real-Time Tracking System," 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 504-509, doi: 10.23919/INDIACom61295.2024.10498283.
10. N. Kannan, "Neelash's IoT Assignment 2," GitHub repository, 2024. [Online]. Available: https://github.com/neelashkannan/Neelash-s-IoT-Assignment-2. [Accessed: Nov. 9, 2024].

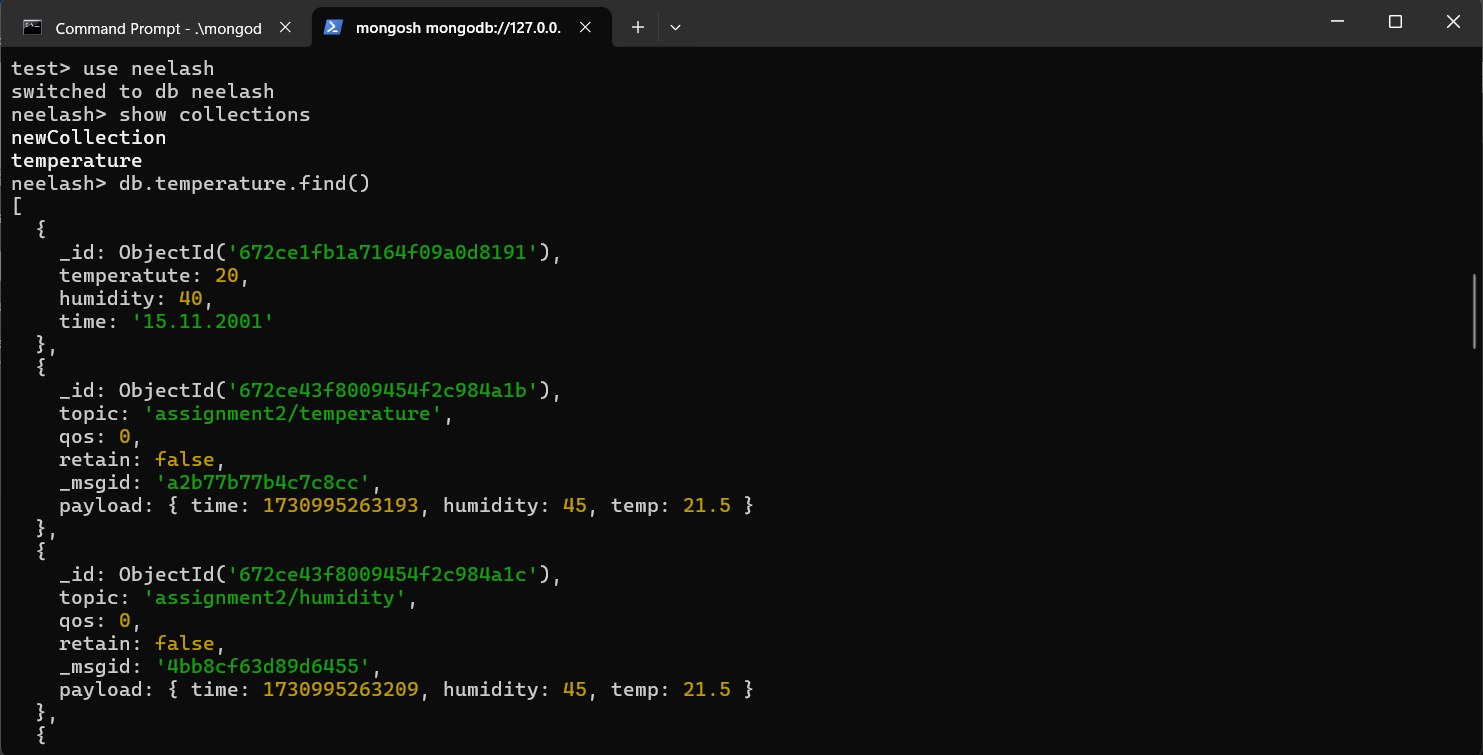
##### Working Video and Github Repo

1. Github Repo - https://github.com/neelashkannan/Neelash-s-IoT-Assignment-2
2. Working Video – (Video is also attached in Readme.md in github repo) https://github.com/neelashkannan/Neelash-s-IoT-Assignment-2/blob/main/Video/working\_video%20conv.mp4

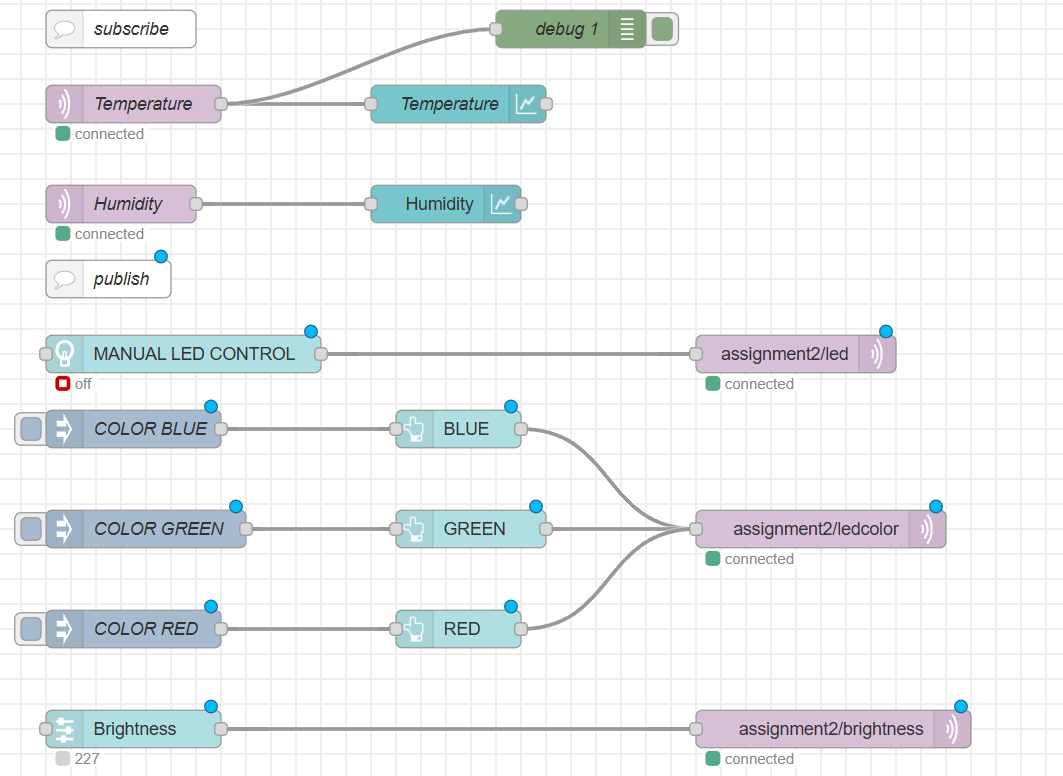
# Appendix

  
  
Temperature, humidity Monitoring and LED Control in Node-red

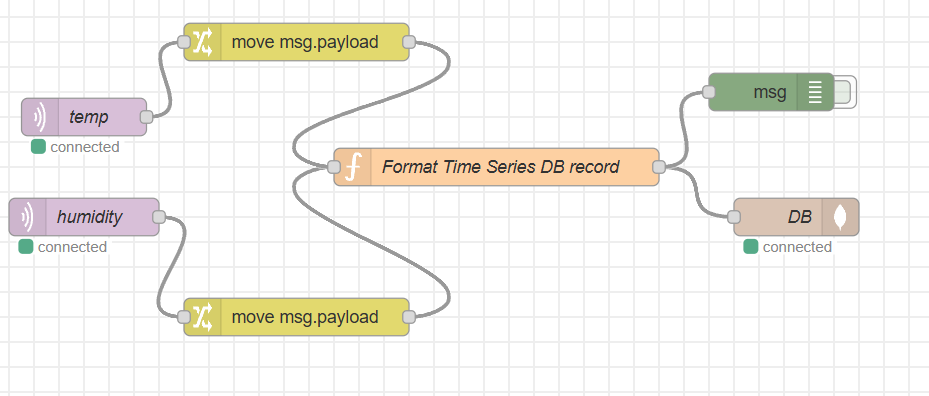
  
History graph – Data loaded from MongoDB



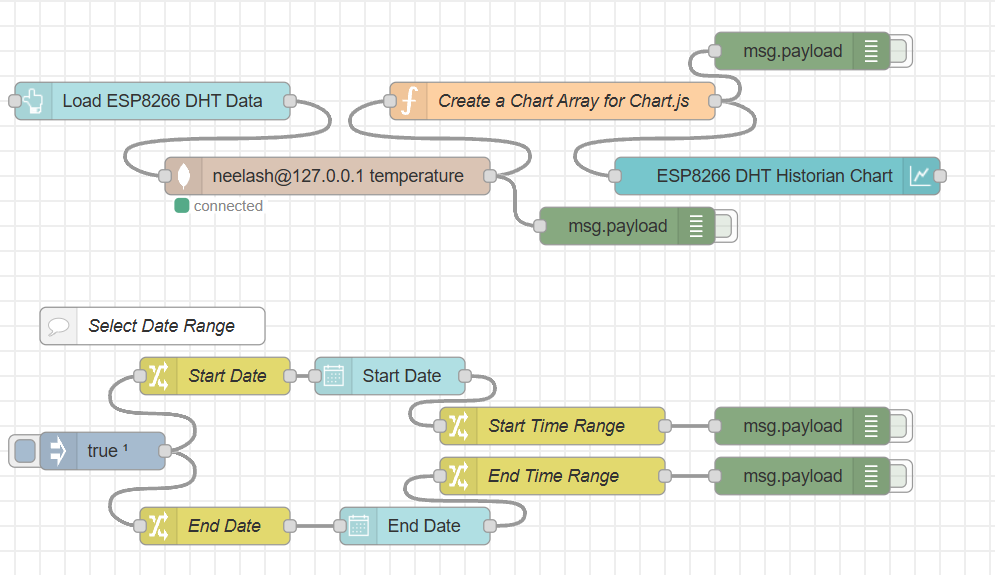
Data Collection in MongoDB



Node-red Flow to subscribe and publish topics



Node-red flow to store sensor data in MongoDB



Node-red flow to Fetch data from MongoDB and represent it based in Graph based on start and end date