GROUP ASSIGNMENT (PRACTICAL)

How IOT contribute to agriculture development

Author: Tran Thanh Minh

Student ID: 103809048

SWE30011 – IoT Programming

Fall 2023

Lecturer: Tuan Tran

Swinburne University of Technology

Contents

I. Summary

a. Introduction

As the global population burgeons and environmental concerns loom large, the need for more effective, sustainable, and data-centric farming methods becomes increasingly evident. The focus is on how IoT technologies empower farmers to monitor their land and crops remotely, accessing crucial information and insights on their smartphones or other personal devices.

This report examines a proposed smart farming system that goes beyond the conventional, integrating cloud computing for data analysis and utilizing IoT sensors for real-time data collection. The system comprises sensors for environmental data, an edge server for real-time processing, and a database for storing the collected information. Clients interact with the system through web browsers, enabling real-time monitoring, distributed architecture, and scalability. The incorporation of cloud technology adds a layer of sophistication, enabling alarms, triggers, and management through cloud servers. The utilization of ThingsBoard further enhances the system's capabilities, providing a comprehensive solution for modernizing agriculture practices.

b. Topic background

An age of transformation has begun with the rise of the Internet of Things (IoT), which is changing industries all over the world. Agriculture is one industry that has been significantly affected. Data-driven practices and gadget connectivity are transforming conventional farming techniques. In this regard, my survey article explores the consequences of IoT innovation in agriculture, ranging from sustainable practices to precision farming, and its multifarious influence.

Agriculture has always been associated with manual work and tradition. But as the world's population has increased and environmental concerns have grown, it is becoming more and more clear that farming needs to adopt more effective, sustainable, and data-driven methods. IoT integration in agriculture opens a new technological frontier that tackles long-standing issues. This study investigates how farmers may monitor their land and crops from a distance using the Internet of Things, getting vital information and insights on their cellphones or other personal devices. Due to the extraordinary control and reactivity that this degree of connectivity provides, both farmers and residents can take prompt action to reduce possible risks and losses.

The adoption of IoT technologies has brought in a new era of agricultural efficiency and management in farming techniques. The Internet of Things has greatly enhanced sustainable farming operations. IoT-enabled continuous environmental condition monitoring helps with illness prevention and pest management.

The advantages and disadvantages of a suggested smart agriculture system for cornfields that makes use of drones and wireless sensor networks are examined. Even while drones have the potential to increase crop yields, there are certain obstacles, like the necessity for numerous sensors and their high cost.

The survey concludes by examining a smart farming system that uses cloud computing for analysis and Internet of Things sensors for data collection. Farmers receive recommendations from machine intelligence on how to increase agricultural yields while lessening their impact on the environment. highlighting how IoT devices can collect data in real-time and use that data to help farmers make decisions that will increase crop yields, save costs, and protect the environment. The survey report acknowledges the transformative potential of IoT, but it also points out the obstacles that still need to be overcome before it can be widely used in agriculture, including high costs and complexity. IoT is expected to play an increasingly important role in agriculture as it develops, making farming and industry's future more profitable, sustainable, and efficient.

c. Proposed system

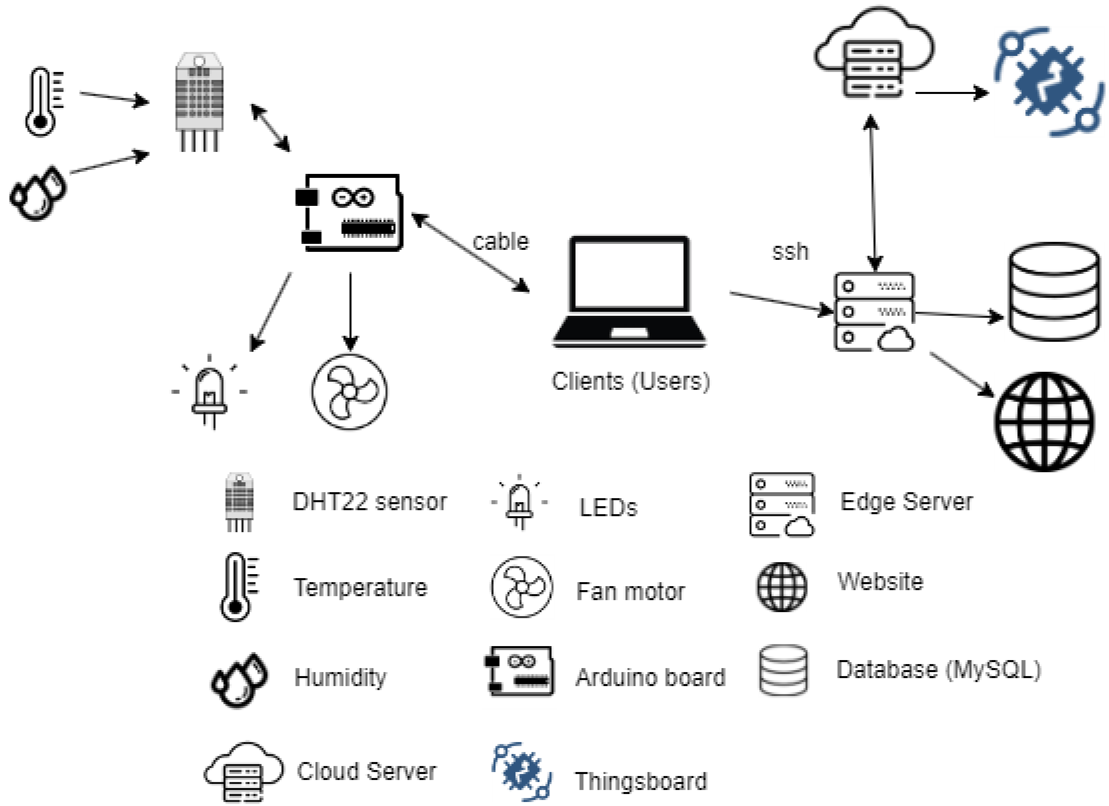


Figure 1: Sketch of proposed system

The following elements make up the suggested system:

- **Sensors**: These devices gather environmental data, including humidity and temperature.
- **Actuators**: These devices will work based on the conditions of the system.
- **Edge server**: The edge server gathers and processes sensor data instantly. Moreover, serial input can be written to the Arduino board using this method.
- **Database**: The information gathered by the edge server and sensors is kept in the database.
- **Cloud server**: This will show the data visualization get from Arduino and sound an alarm.
- **Website**: This will show the data got from the sensors and will interact with Arduino.

Clients can use web browsers to communicate with the system. The networked architecture, scalability, and real-time monitoring of this suggested system are just a few advantages. With the ability to monitor in real-time and detect changes instantly, the system may react to changes swiftly and sound a warning via actuators. The system's dispersed components contribute to its scalability and dependability, and it may be expanded with the addition of more sensors and actuators. This system can assist in the monitoring and management of agricultural equipment, including greenhouses and irrigation systems. To manage humidity, temperature, and the current time stamp, this system also uses MySQL storage.

DHT22 is a good fit for this use case as the sensor for measuring temperature and humidity, however humidity will be primarily utilized for system management. According to PLNTS.com (2021), plants prefer a humidity of between 60% and 80%. However, for this project, I will adjust the humidity based on trigger circumstances to make humidity variations easier to control.

The edge server, responsible for instant data processing from the sensors, utilizes MQTT to securely transmit this processed data to the cloud server. This enables the cloud server to receive real-time updates on environmental conditions, such as humidity and temperature, captured by the sensors in the agricultural setting.

Once the cloud server receives the data, it engages with ThingsBoard. Through ThingsBoard, the cloud server not only displays data visualization but also triggers alarms based on predefined conditions. This integration enhances the system's responsiveness, allowing for timely alerts and notifications in the event of significant changes in the monitored conditions.

In essence, the MQTT communication protocol establishes a robust link between the edge and cloud servers, enabling a continuous flow of data that contributes to the overall effectiveness of the smart farming system. Through this interconnected architecture, the system achieves a harmonious coordination between on-site data processing and centralized management, ultimately optimizing agricultural operations for enhanced efficiency and sustainability.

II. Conceptual Design

a. Block diagrams

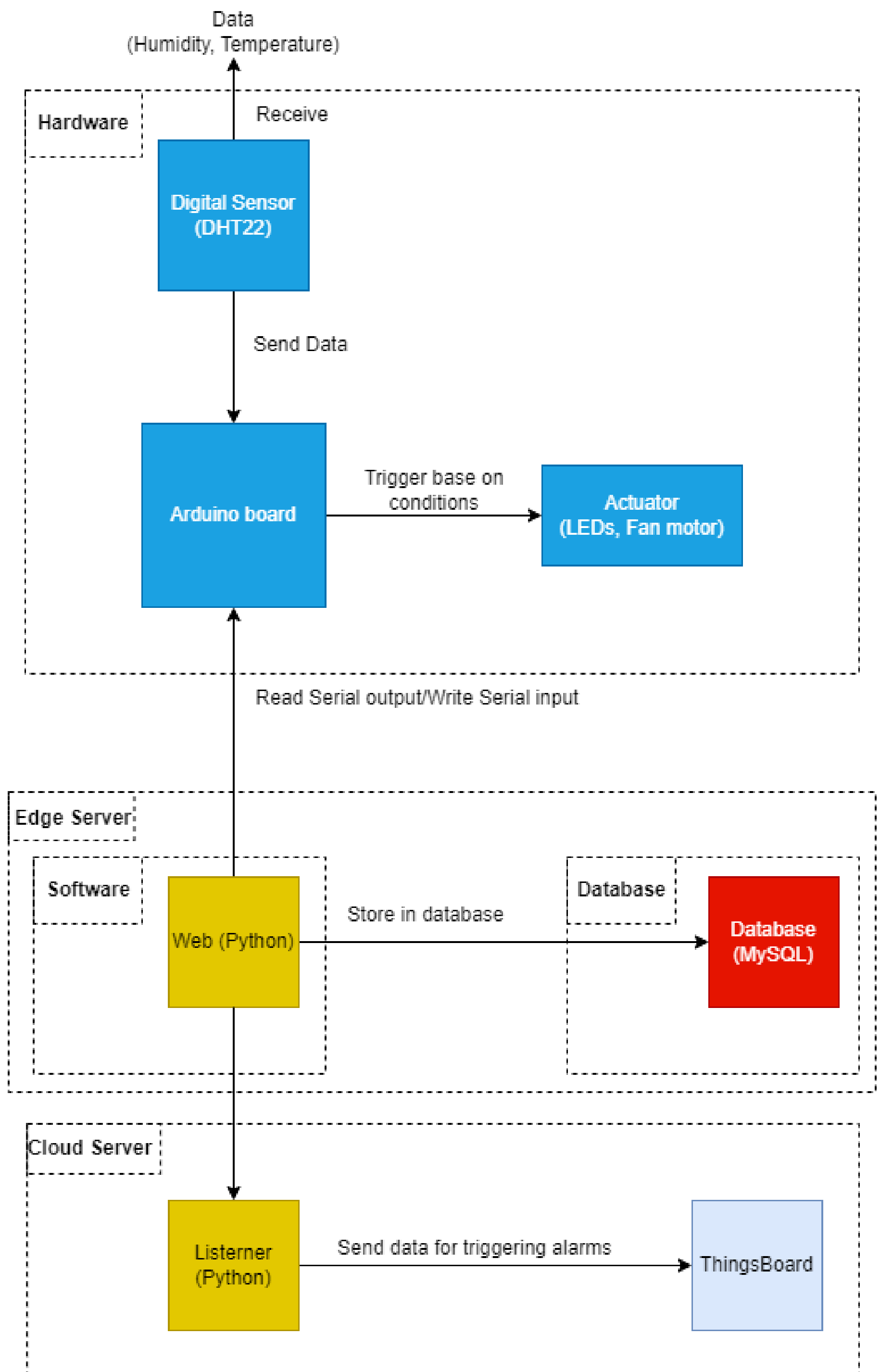


Figure 2: Block diagram for proposed system

In the initial phase of data acquisition, the digital sensor (DHT11) actively gathers environmental data, specifically humidity and temperature, from its surroundings. This collected data is then transmitted to the Arduino board, serving as the central processing unit for the smart farming system. The Arduino board interprets the received data and, when necessary, activates the actuators. This real-time interaction between the sensor, Arduino board, and actuators forms a responsive loop that promptly communicates changes in environmental conditions to the user.

Simultaneously, the web application serves as the user interface, providing a graphical representation of the gathered data. The application reads the serial output from the Arduino board and displays the information in an easily understandable format for users. Depending on user preferences, the web application offers functionality to store this data in a MySQL database, facilitating long-term data storage and analysis.

In the subsequent phase, the system introduces a mode triggered by the user to autonomously send data to a cloud server. Upon enabling this mode, the system seamlessly transfers the gathered data to ThingsBoard—an IoT platform designed for data visualization and management. This transition to the cloud enhances the system's capabilities by providing a centralized and accessible repository for agricultural data.

Within ThingsBoard, users can visualize the environmental data in a dynamic and interactive manner. Moreover, the platform is configured to respond to set conditions, triggering alarms or notifications when specific thresholds are crossed. This integration with ThingsBoard not only enhances data visualization but also introduces an additional layer of intelligence, allowing for real-time alerts and alarms based on user-defined criteria.

b. UML diagram

1. Diagram for how Arduino work

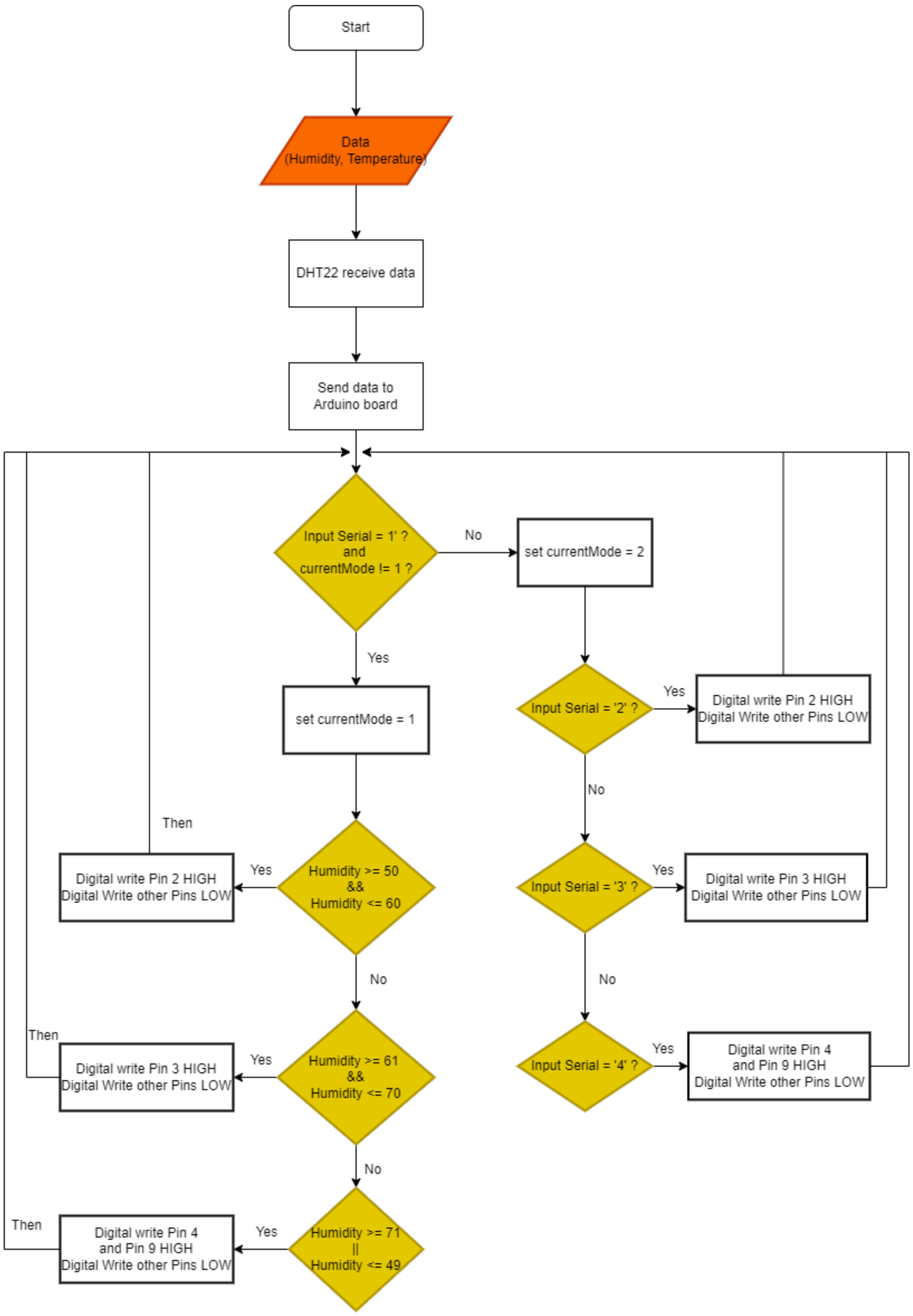


Figure 3: Flow chart for how the Arduino work

In the operational sequence depicted in the chart, the Arduino code functions in a continuous loop, perpetually executing the designated commands. The primary data inputs for this code consist of humidity and temperature readings acquired from the DHT22 sensor.

The code operates in two distinct modes: In the first mode, triggered when the user inputs '1' to the Arduino, the code autonomously initiates and undertakes the task of monitoring and analyzing the humidity data received from the DHT22 sensor. This automated process ensures continuous surveillance of environmental conditions, providing valuable insights into humidity variations.

On the other hand, if the user inputs '2,' '3,' or '4' to the Arduino, the code seamlessly transitions to the second mode. In this mode, the code executes specific actions, notably setting the corresponding pins to a HIGH state while simultaneously setting other pins to a LOW state. This dynamic mode-switching mechanism enables the user to customize and control the behavior of the system based on the input provided, facilitating a flexible and user-driven experience.

2. Diagram for how the Python code work for sending request from the web:

For the below image, the description of the system is visualized. First, when the user accesses the web, the webpage will display and then when the user requests the data, the data will be fetched by Python and display back in the web. The user can choose whenever the Arduino runs automatically or manually for managing the state. If the user chooses to run automatically, the back end of the website will write the Serial input to the Arduino board and the program will auto run (refer to Figure 3). However, if the user chooses to run manually, the user can choose whenever to store the data in the database or not. In both cases, the first thing the backend of the web does is to read the data (humidity and temperature) from the Serial output. If the user chooses to store, the backend of the web will store the data (humidity, temperature) in the database include the time and date of the current request. the backend of the web will send Serial input ‘2’ to the Arduino code and then the Arduino board will trigger the suitable pin to alert (refer to Figure 4).

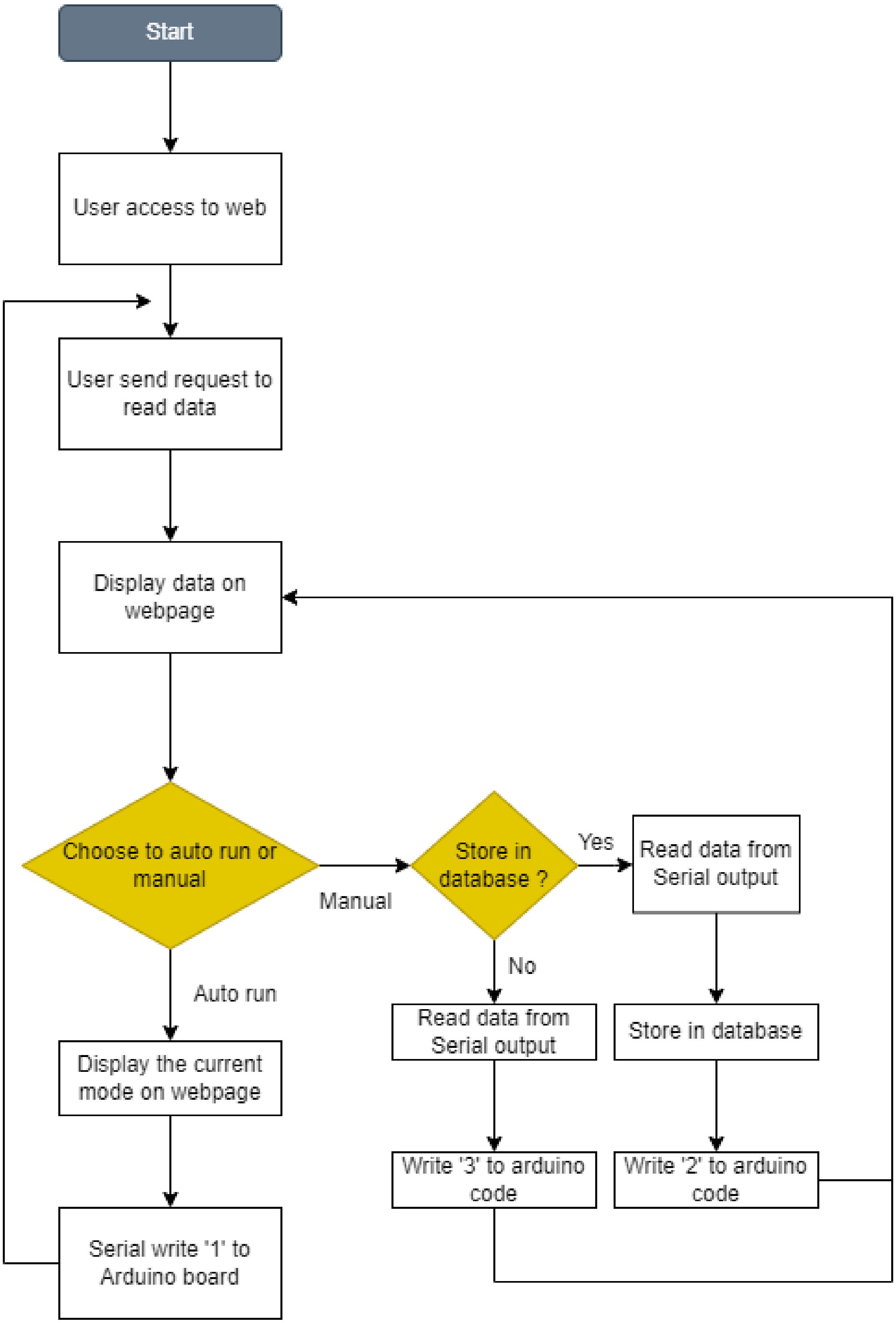


Figure 4: Flow chart for how the python program works and send request.

III. Task Breakdown

Name of Contributors	Specific Work Done
Tran Thanh Minh	- Design system - Code for both Cloud and Edge server to run. - Report

IV. Implementation

The Arduino operates as a pivotal node in this interconnected system, tasked with the collection and initial processing of data from the DHT22 sensor. The DHT22 sensor serves as a reliable source for monitoring the ambient temperature and humidity, providing essential insights for applications such as climate control and data logging. Integrated with the Arduino, the DHT22 delivers sensor values, which are then transmitted as Serial output in a customized format. Arduino can execute actions based on this data, serving as a crucial link between the physical environment and the digital realm.

a. Sensors

DHT22 is a type of sensor that measures the ambient temperature and humidity. DHT22 is utilized in this project to keep an eye on the surrounding circumstances. This is critical for applications where temperature and humidity information are required, including data logging or climate management. The Arduino module is integrated with the DHT22, allowing it to read sensor data and print serial output in a format of choice. Based on the information obtained, it is capable of acting.

b. Actuators

Within the system, actuators play a dynamic role in responding to environmental conditions. LEDs serve as visual indicators, signaling specific conditions based on received data, such as illuminating when a predetermined temperature threshold is reached. Meanwhile, the fan motor serves a practical purpose by regulating humidity levels. Triggered by DHT22 readings, the fan motor engages when humidity surpasses a set point, providing automated humidity control. This functionality is paramount for real-world applications, especially in scenarios requiring efficient cooling systems.

c. Software/Libraries

Python takes the reins as the primary programming language in this project, collaborating with essential libraries like MySQLdb, serial, Flask, and render_template. Python's versatility is harnessed for data processing, communication, and front-end development. Through serial communication, Python interacts with the Arduino, retrieving sensor data and storing it in a MySQL database. Flask, in conjunction with render_template, facilitates the creation of a user-friendly web interface. Users can monitor and control the system by sending requests to programmed endpoints, ensuring a seamless and interactive experience.

d. Edge servers and Cloud Connectivity

Beyond local processing, an edge server powered by Python and Flask further refines the data before communicating with a high-level system. This server also plays a pivotal role in integrating the system with cloud servers and ThingsBoard. Communication protocols, such as MQTT, facilitate seamless interaction between the edge server and Arduino, enabling the transfer of processed data to the cloud.

e. Cloud Computing

Cloud computing adds a layer of sophistication to the system, allowing for centralized storage, processing, and accessibility of data. The cloud server, connected via MQTT, receives data from the edge server and triggers alarms based on set conditions using ThingsBoard. This integration enhances the system's capabilities, providing users with a comprehensive platform for data visualization, real-time monitoring, and intelligent alerts, fostering a holistic approach to managing environmental conditions.

f. Communication protocols

The MQTT communication protocol connects the edge and cloud servers seamlessly. This protocol ensures efficient and reliable data transfer between these components, forming a cohesive system architecture.

g. Website

The user interface is realized through a web application. Flask and render_template enable the creation of a user-friendly website, offering a platform for users to interact with and monitor the system. This website facilitates the seamless exchange of requests, enhancing the user experience.

h. Database

MySQL assumes the role of the database system, efficiently storing and managing sensor data. This relational database structure allows for streamlined information retrieval, incorporating crucial time and date information. The database becomes a repository for all records, enabling comprehensive analysis of temperature and humidity patterns over time.

i. Tinkercad Model

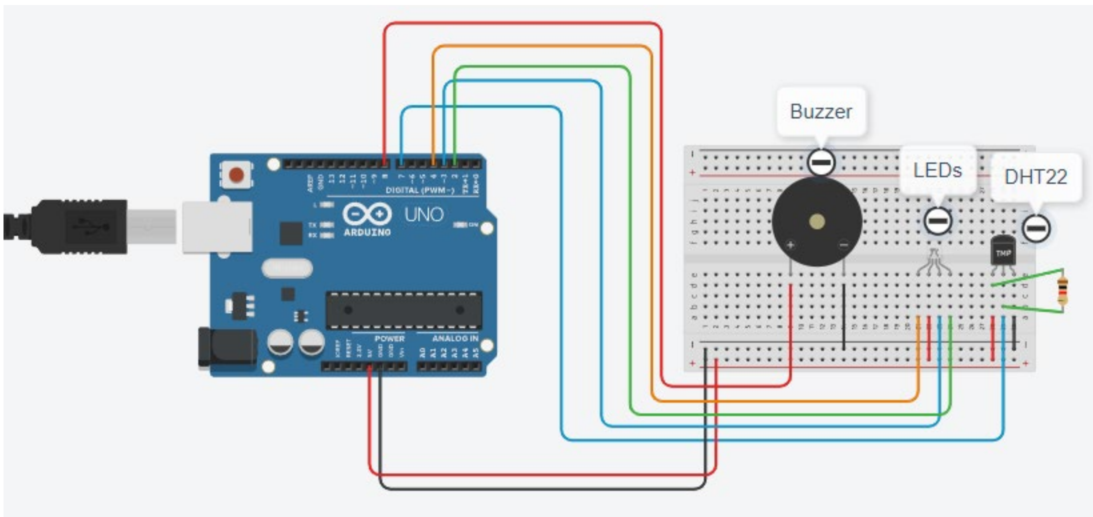


Figure 6: Tinkercad Model of proposed system

Since Tinkercad does not offer the right motor for my suggested design, I have substituted the buzzer for the display. The functionality remains unchanged, though. The criteria specified in the code will cause the system to alert. The circuit's logic is shown in the schematic image below.

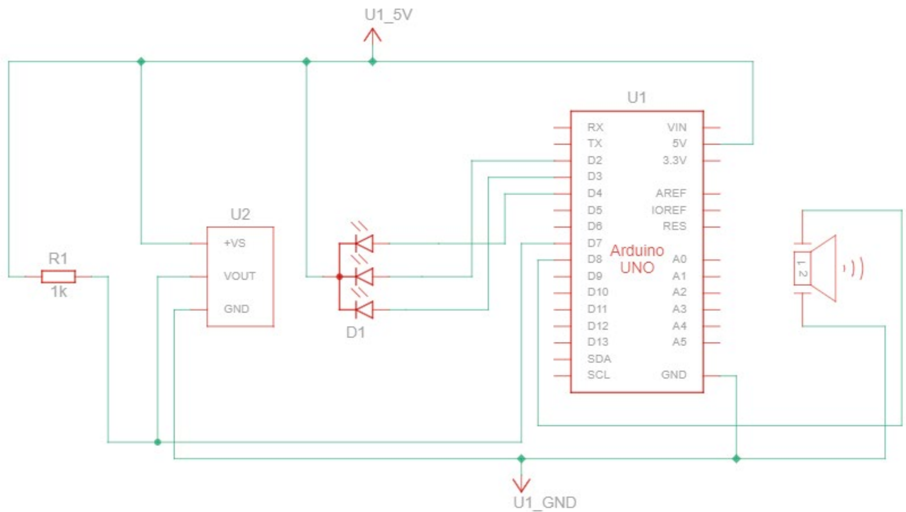


Figure 7: Schematic view of Tinkercad

j. Evidence of building system

Here are 2 figures that show how I has routed the wired to make the connection between my personal device and the Arduino.

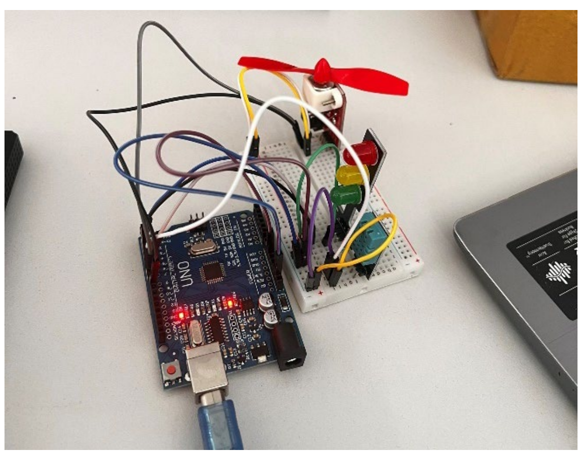


Figure 8: Side view of proposed system

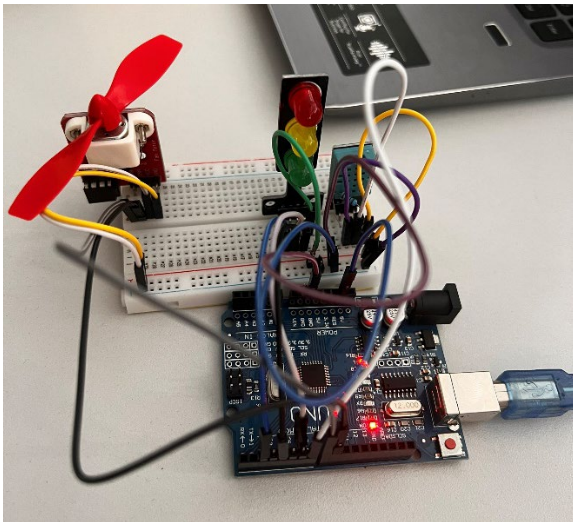


Figure 9: Front view of proposed system

k. Rule Chains for triggering alarm

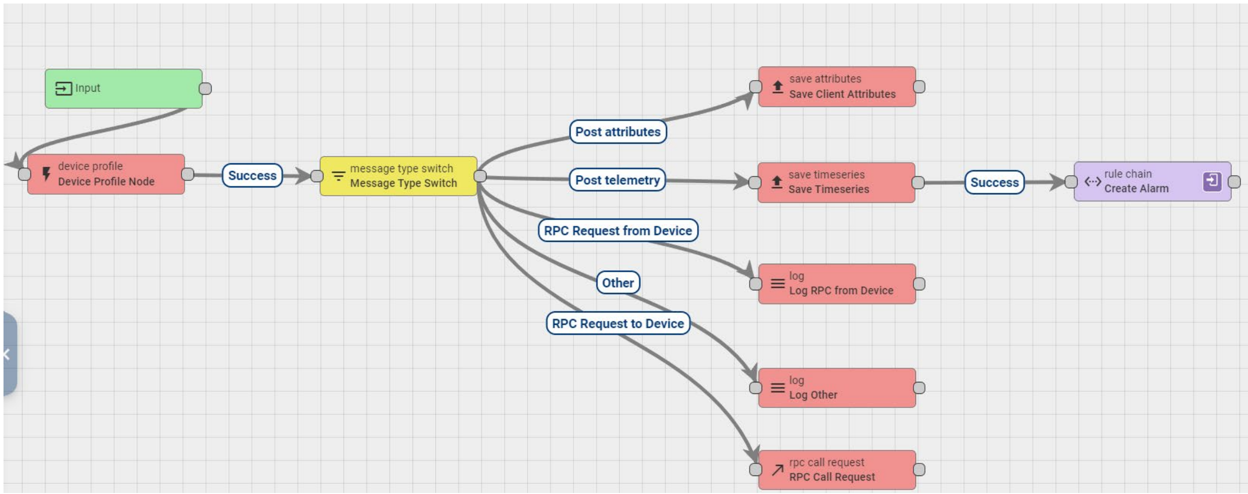


Figure 10: Root Rule Chain

The input data will be received from the device, this data will be in the form of telemetry data. The create alarm component is used to create alarms based on telemetry data, so that when the humidity exceeds a certain threshold then it will trigger alarms.

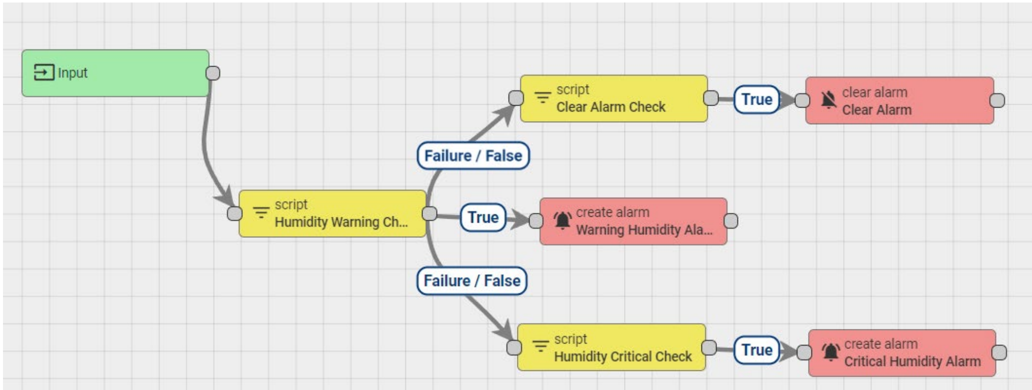


Figure 11: Rule Chain for Creating Alarm

This image shows the process of input is read from device. Then it will check whenever the temperature is in range from 61 – 70 or above 71 or below 50, then it will trigger the suitable alarm.

I. ThingsBoard Dashboard

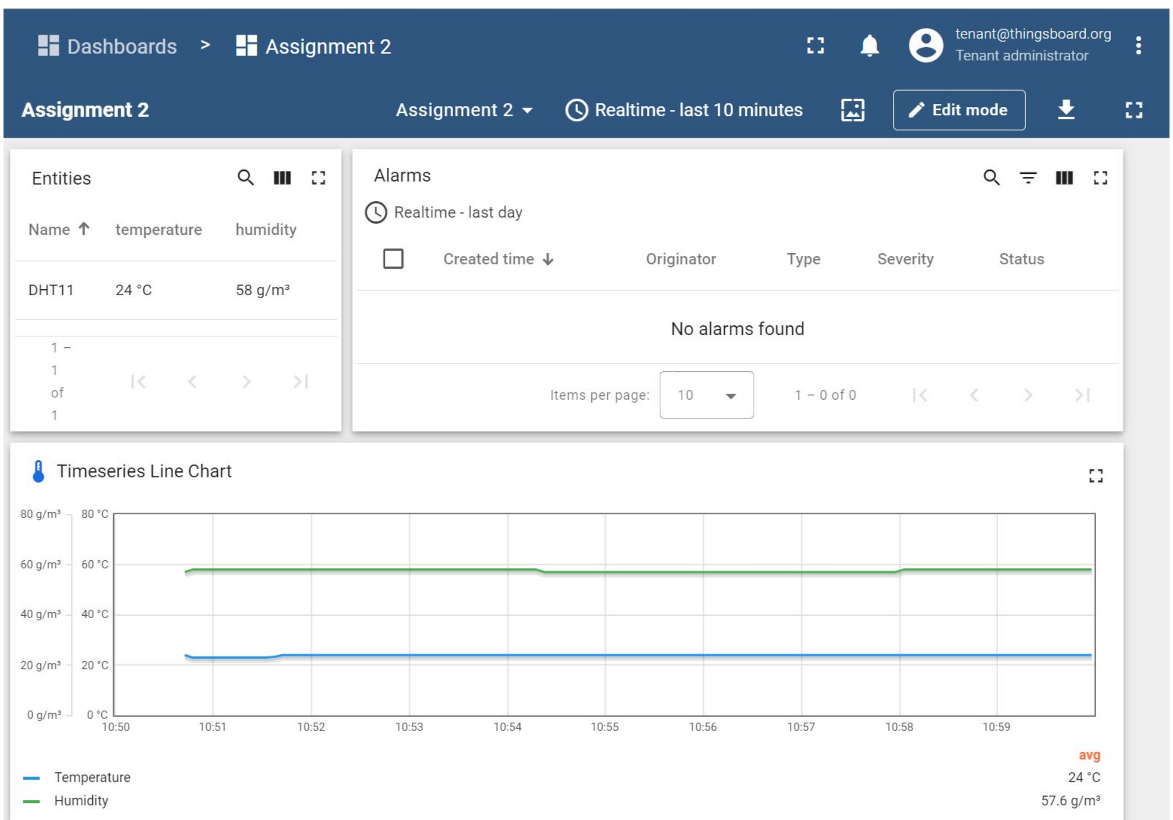


Figure 12: Assignment 2 Dashboard

V. User Manual

1. Installation:

Install the VMware from this link : Download VMware for Windows

Then, you need to install the Raspberry Pi OS for Windows: Download Rasp Pi OS

2. Setting up VMware:

Import the file Rasp Pi OS ISO to VMware twice, the first one is for the Edge Server, the second one is for Cloud Server.

Then you need to upgrade for both 2 servers by using these commands:

- sudo apt update
- sudo apt upgrade

Then, you need to configure so that it can be connected through SSH by the following commands:

- sudo systemctl enable ssh.service
- sudo systemctl start ssh.service

3. Configure Ports on both servers:

Go to the settings for both servers.

For Edge server, you need to enable Serial Ports, Port 1. Then enable Port Number COM1 (so this port will display as /dev/ttyUSB0 in Linux)

For Cloud server, you need to disable Serial Port so that it only allows the connection from Edge server through MQTT.

4. Install packages for servers

Edge server:

First, you need to install the packages for Python and MySQL with these command lines:

- sudo apt-get install mariadb-server

- sudo apt-get install python-pymysql python3-pymysql

With the above packages, the user can use Python to interact with database.

Then, you need to install flask package for using library Flask by using the command line:

- sudo apt-get install python-flask

We need to install the client mosquito on edge server by using command line:

- sudo apt-get install mosquito-clients

We also need to install MQTT client library paho-mqtt for sending data through cloud server by using command:

- sudo apt-get install python3-paho-mqtt

Cloud server:

First, we need to install the MQTT broker and client for cloud server:

- sudo apt-get install mosquito mosquito-clients

We also need to allow the anonymous connections by opening config file:

- sudo nano /etc/mosquito/mosquito.conf

and then add the following lines to the end of it:

- allow_anonymous true
- listener 1884 192.168.153.196

We also need to install mosquito and enable it by using the command line:

- sudo systemctl start mosquito.service
- sudo systemctl enable mosquito.service

5. Install ThingsBoard on Cloud Server:

First, you need to install java 11 by using this command line:

- sudo apt install openjdk-11-jdk

You need to install the suitable PostgreSQL with the following command lines:

- wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add –
- echo "deb https://apt.postgresql.org/pub/repos/apt/ $(lsb_release -cs)-pgdg main" | sudo tee /etc/apt/sources.list.d/pgdg.list
- sudo apt update
- sudo apt -y install postgresql-12
- sudo service postgresql start

Once the PostgreSQL installation is done then you need to use the following command for creating new user and configuring your password:

- sudo su – postgres
- psql
- \password
- \q

Then you need to connect to database to create thingsboard database with the following commands:

- psql -U postgres -d postgres -h 127.0.0.1 -W
- CREATE DATABASE thingsboards;
- \q

Then you need to configure ThingsBoard file:

- sudo nano /etc/thingsboard/conf/thingsboard.conf

Then you need to add the following lines to that file and replace with the PASSWORD that you previously set:

- export DATABASE_TS_TYPE=sql
- export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/thingsboard
- export SPRING_DATASOURCE_USERNAME=postgres

- export SPRING_DATASOURCE_PASSWORD=PASSWORD
- export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS

Then you need to run the installation script for ThingsBoard:

- sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo

Lastly, you can start the ThingsBoard with the command:

- sudo service thingsboard start

After starting, you can go to the web UI with the link: http://localhost:8080/

Or you can go to the web UI from your device with the IP address of the cloud server:

http://192.168.153.131:8080/ (in this case 192.168.153.131)

Then, you need to log in with these credentials:

- Username: tenant@thingsboard.org
- Password: tenant

6. Create Dashboard for the system:

Firstly, you need to add a new customer. By going to the “Customers” category on the side bar, you can add a new customer with the information from the Figure 12.

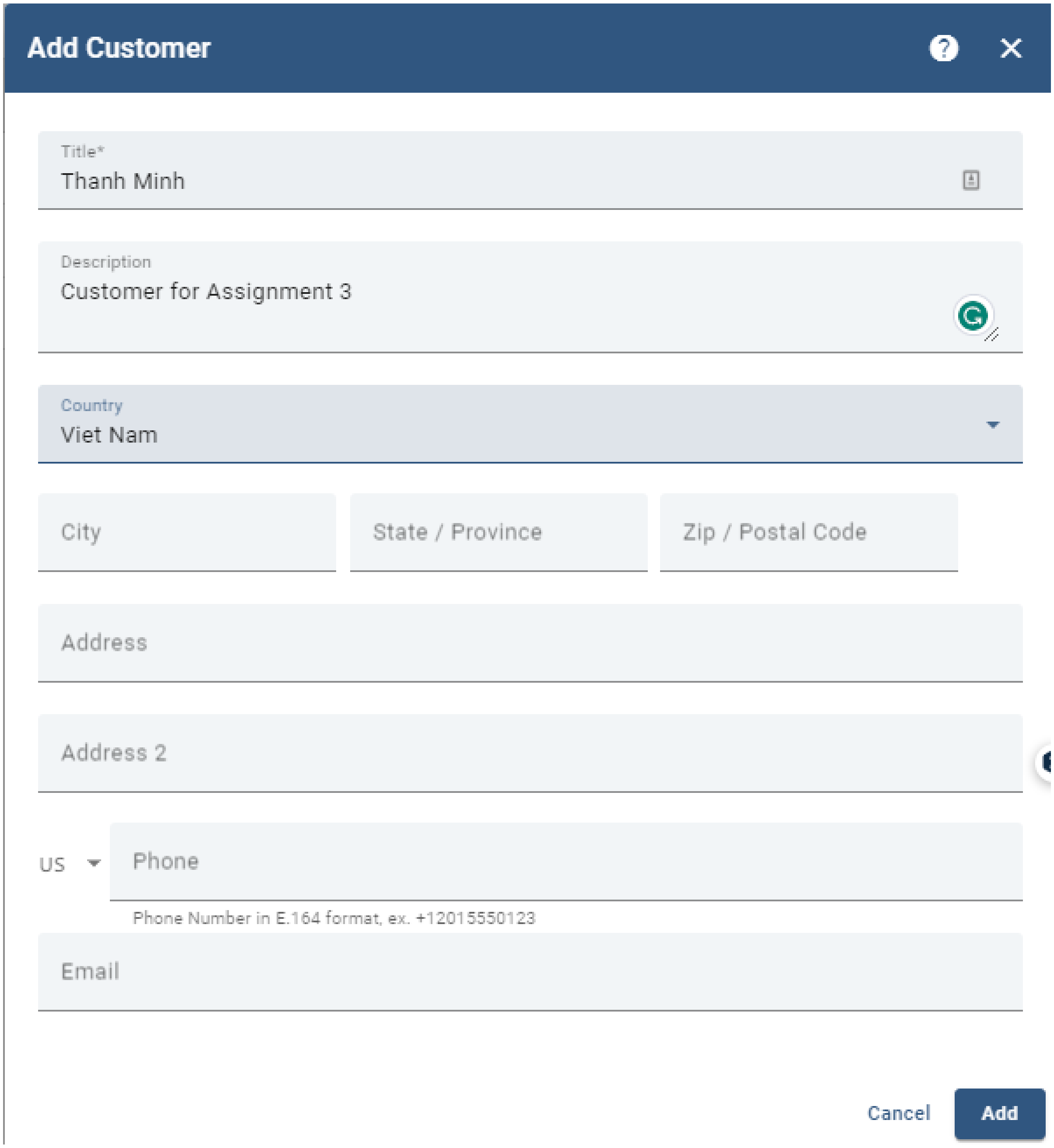


Figure 12: Add new customer Thanh Minh

Then, you need to open the customer you have just created and go to “Manage users”. Then, you need to click the “+” to add new user with the following detail:

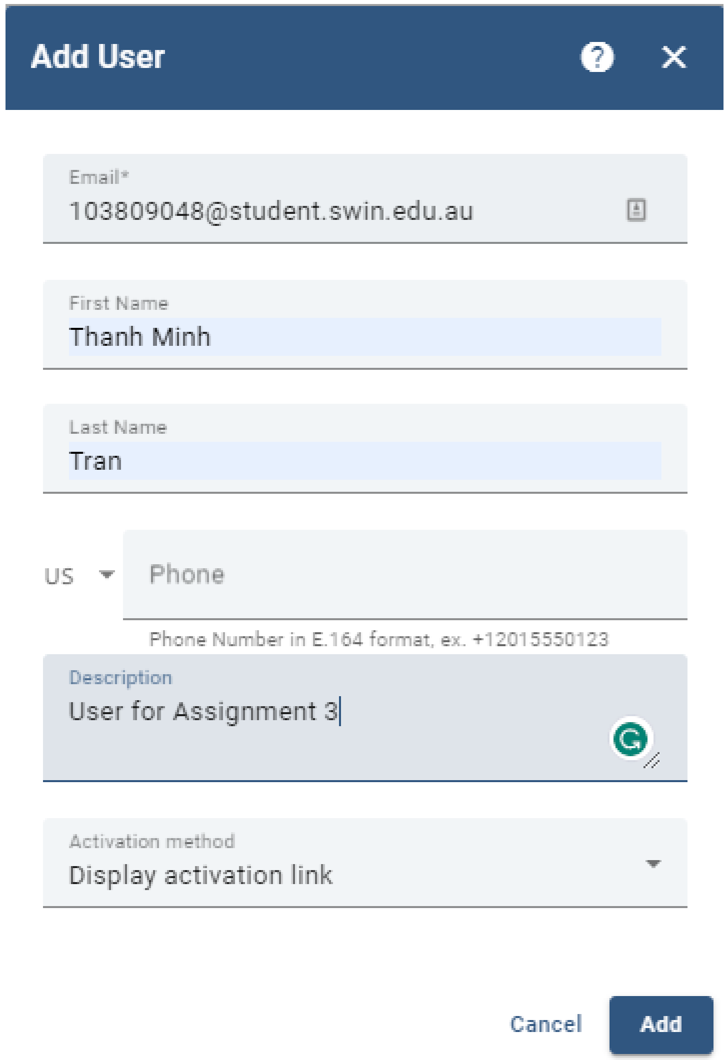


Figure 13: Assign user to customer.

After that, you will get an activation link to set your password and you can login to see the assigned dashboard.

Secondly, you need to create a device. After logging in as tenant administrator, on the left side, you will see "Devices" in the Entities category. You can add the new input device with the "+" button with the following details:

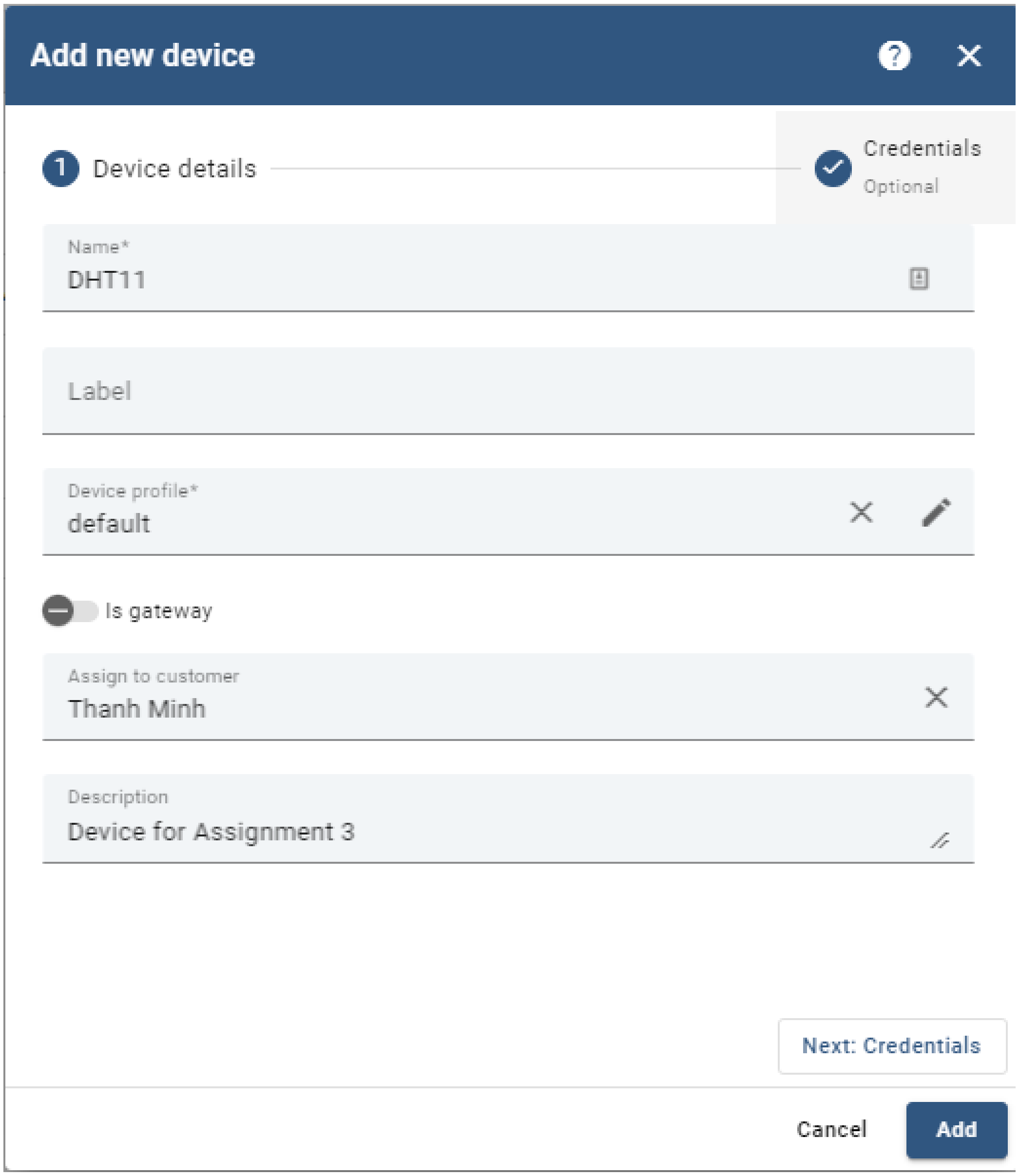


Figure 14: DHT11 device for assignment 3

Then you will get instructions of how to interact with ThingsBoard through MQTT protocol from Linux clients.

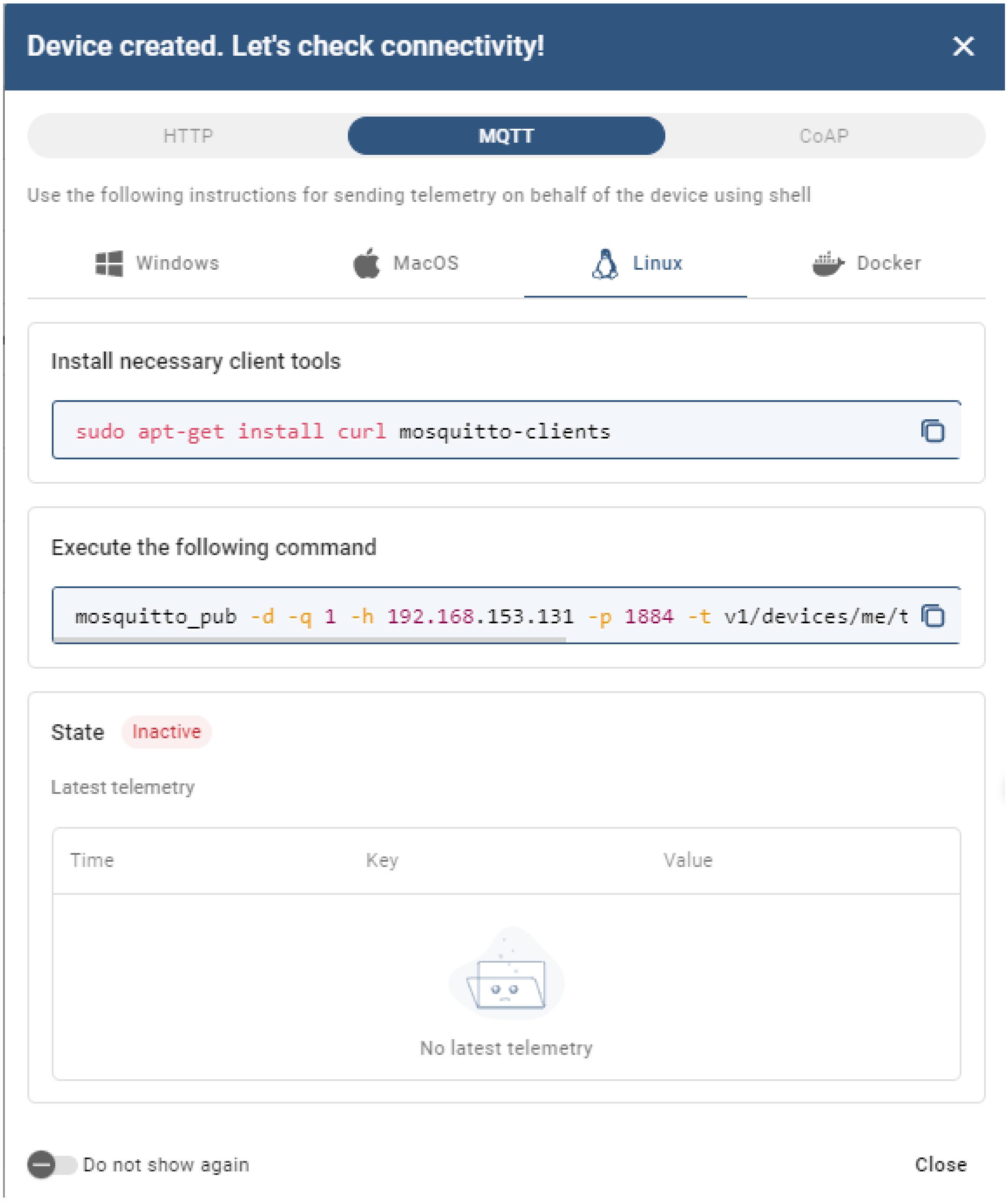


Figure 15: Instruction of interacting with ThingsBoard

Remember the command from Figure 15 so that later on we will convert it into Python code for interacting from the web to the ThingsBoard.

Here is the explanation for the command line "*mosquitto_pub -d -q 1 -h 192.168.153.131 -p 1884 -t v1/devices/me/telemetry -u OwvtaAzFoOYeUU1WSILH -m "{temperature:25, humidity: 50}*" ":

- **mosquitto_pub**: This is the command itself, indicating that you are using the Mosquitto MQTT publish tool.
- **-d**: This flag stands for "debug" mode. When this flag is used, the command runs in the foreground, and it prints debugging information to the console.
- **-q 1**: This sets the Quality of Service (QoS) level for the message to 1. QoS levels in MQTT represent the delivery guarantees for a message. QoS 1 ensures that the message is delivered at least once.
- **-h 192.168.153.131**: This specifies the MQTT broker's host address, in this case, the IP address is 192.168.153.131.
- **-p 1884**: This sets the port number for the MQTT broker to 1884. MQTT brokers typically listen on port 1883, but in this case, it's using a different port, 1884.
- **-t v1/devices/me/telemetry**: This sets the MQTT topic to which the message will be published. Topics in MQTT are used to categorize messages. In this case, the topic is set to v1/devices/me/telemetry.
- **-u OwvtaAzFoOYeUU1WSILH**: This provides the MQTT username for authentication. MQTT brokers can be configured to require a username and password for clients to connect.
- **-m "{temperature:25, humidity: 50}"**: This is the message payload that will be sent to the specified topic. In this example, it's a JSON-formatted message with a temperature value of 25 and humidity value of 50.

Then, you need to create a dashboard for the user with this information:

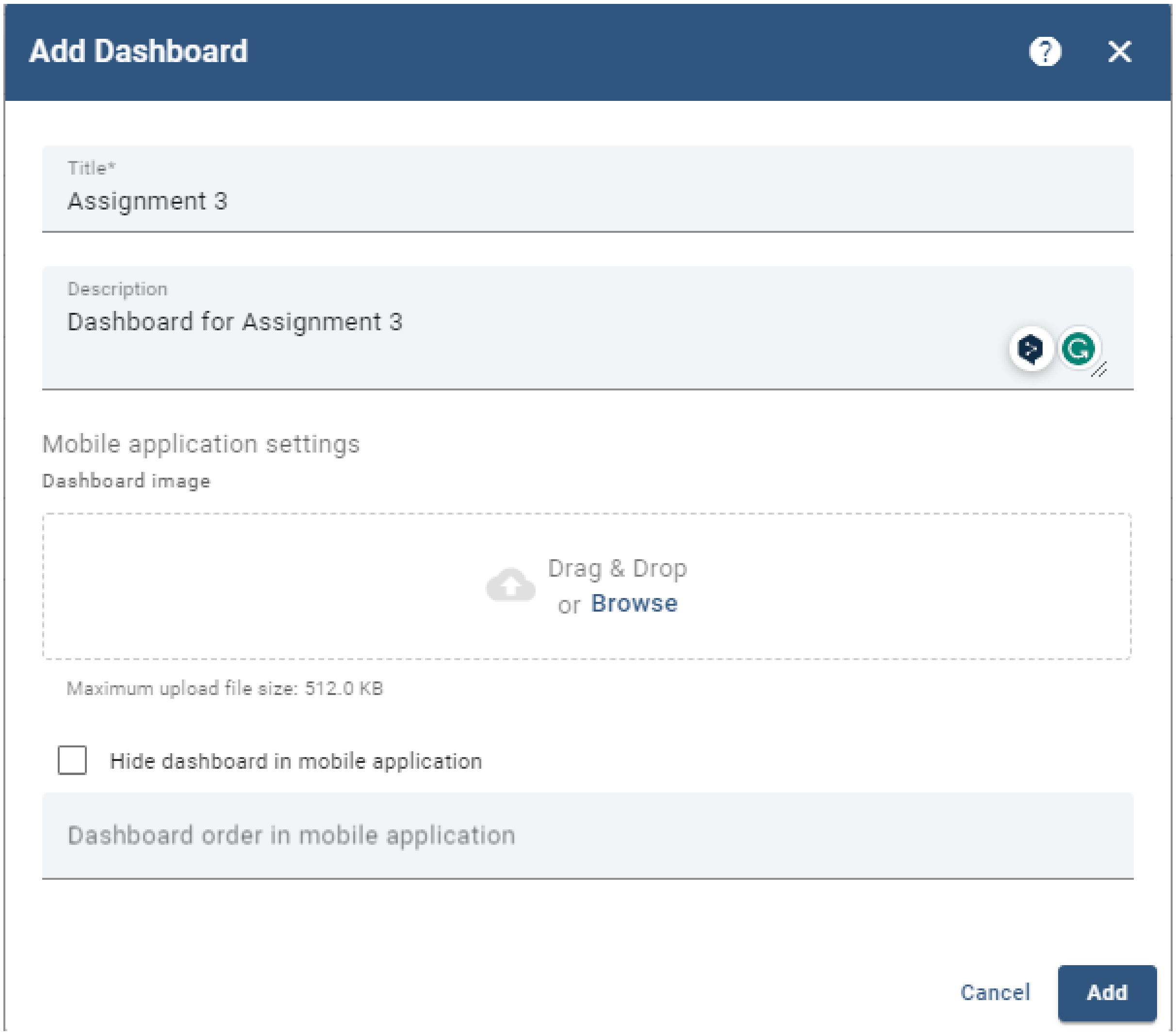


Figure 16: Creating dashboard for Assignment 3

Then, you need to enter the edit mode to add widget to the dashboard.

Click to Add Widget -> Table -> Entities table

Then create the Entities table widget for the dashboard with the following information:

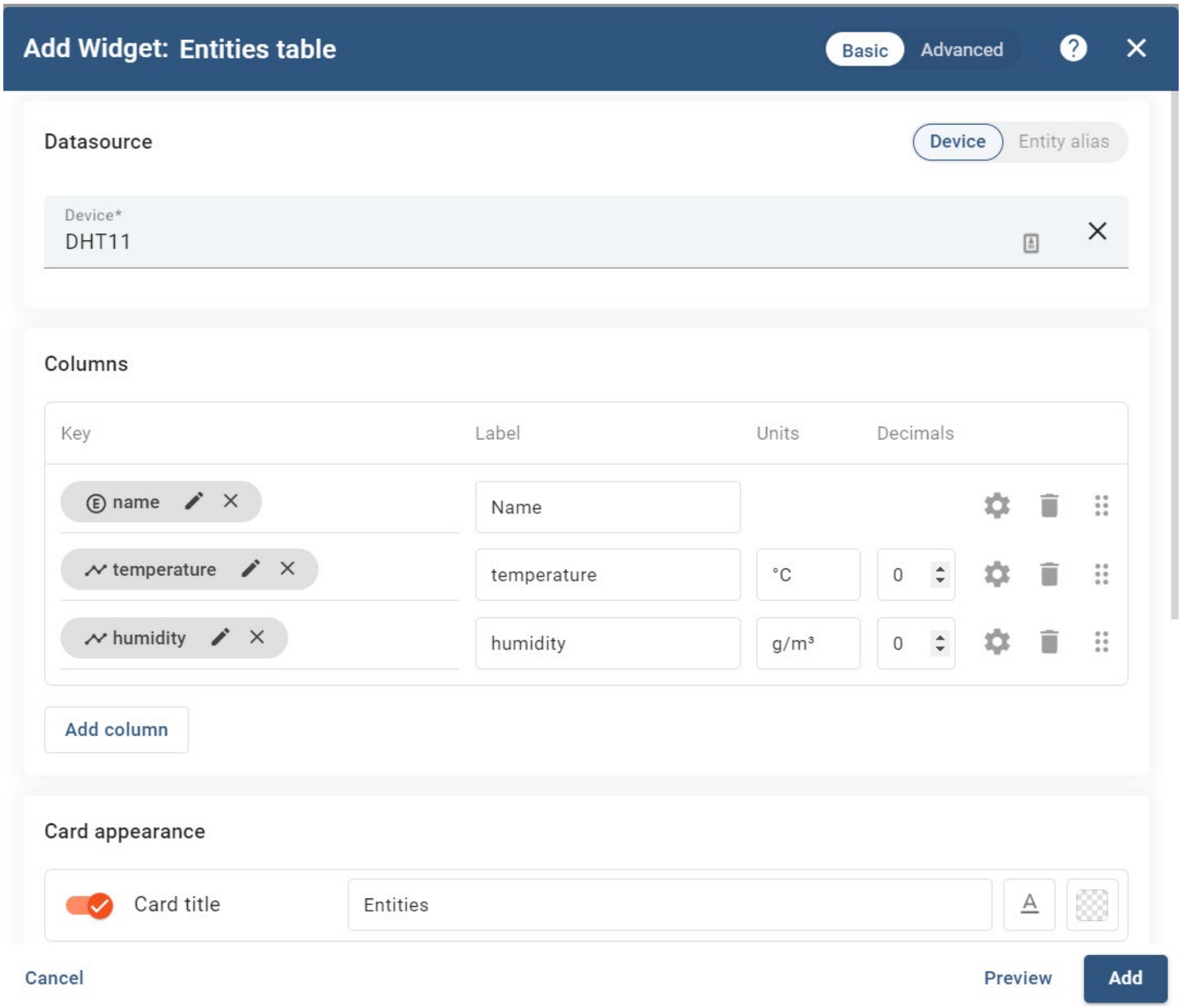


Figure 17: Creating Entities table widget

Then, you need to create the timeserires Line Chart widget in dashboard for visualizing the data with the following steps: Search for Timeseries Line Chart -> Add the following information :

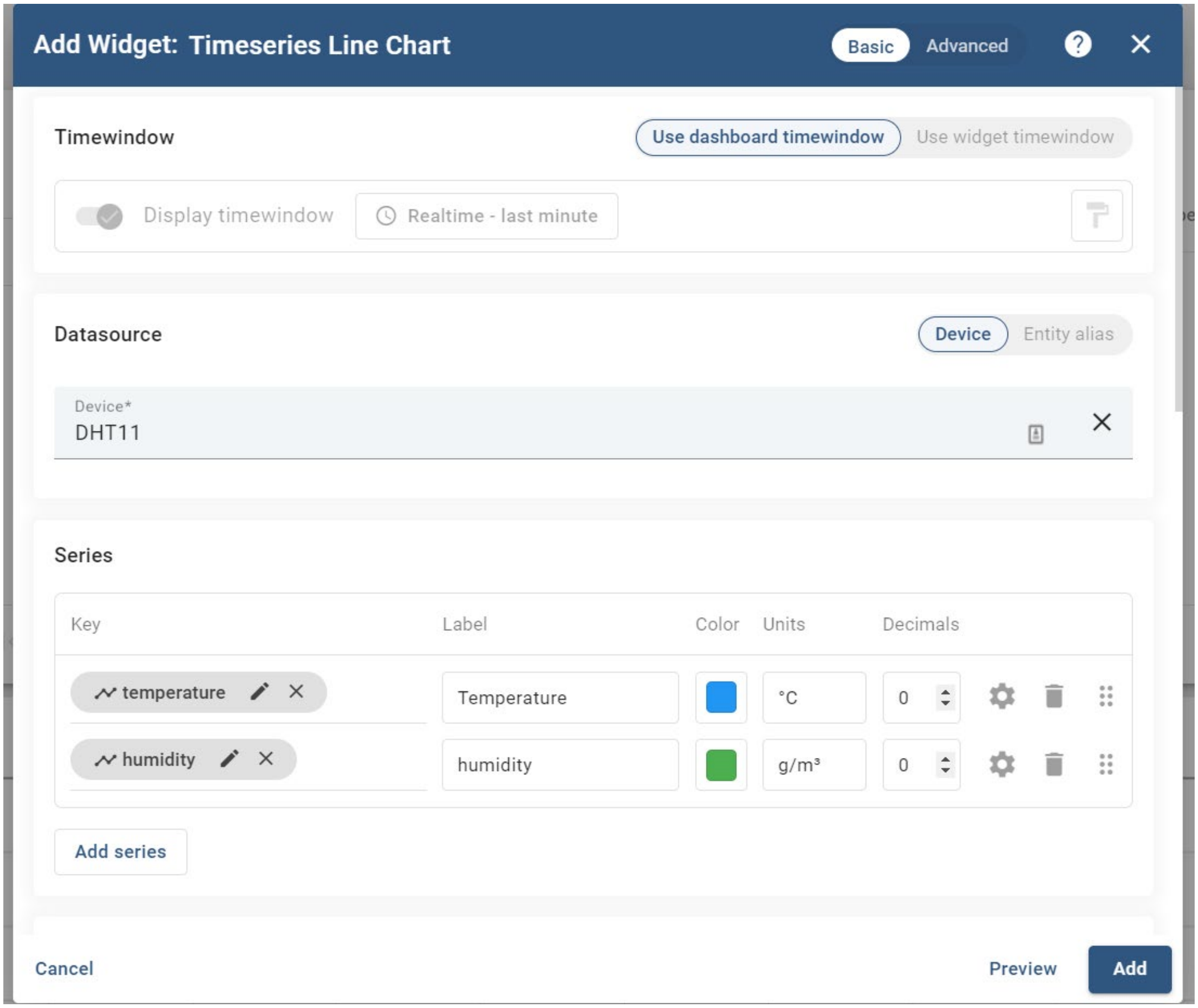


Figure 17: Creating Timeseries Line Chart

Then, you need to creating the alarms table by selecting the alarm widgets -> alarms table.

Then choose the following categories for the alarm table:

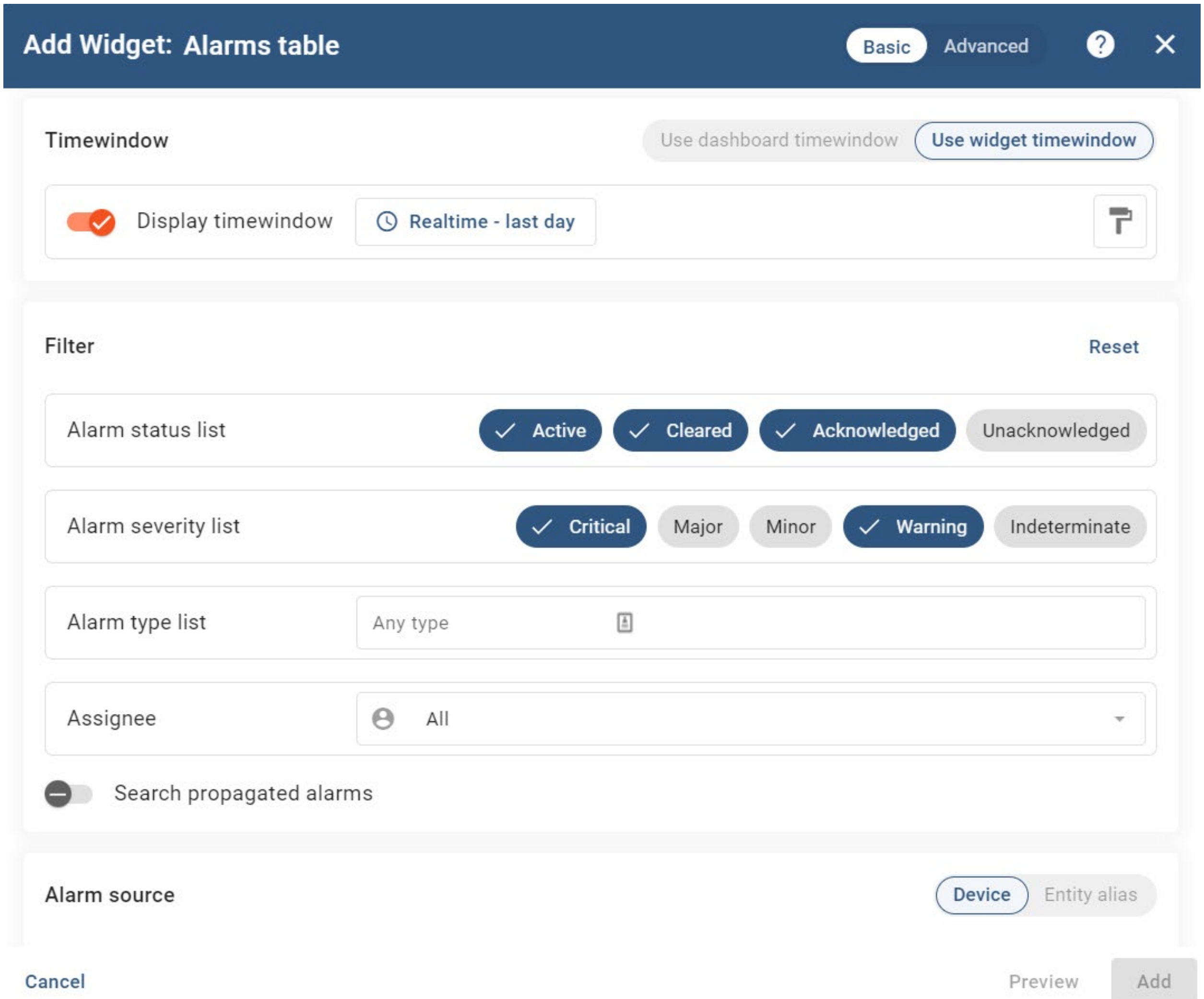


Figure 18: Creating Alarm Table widget.

7. Configure the Alarm

Now, from the ThingsBoard main menu, select the Rule Chains, then create a new Rule Chain:

Figure 19: Create Rule Chain named Create Alarm

Then, you need to search for the "script" node and configure like the image below:

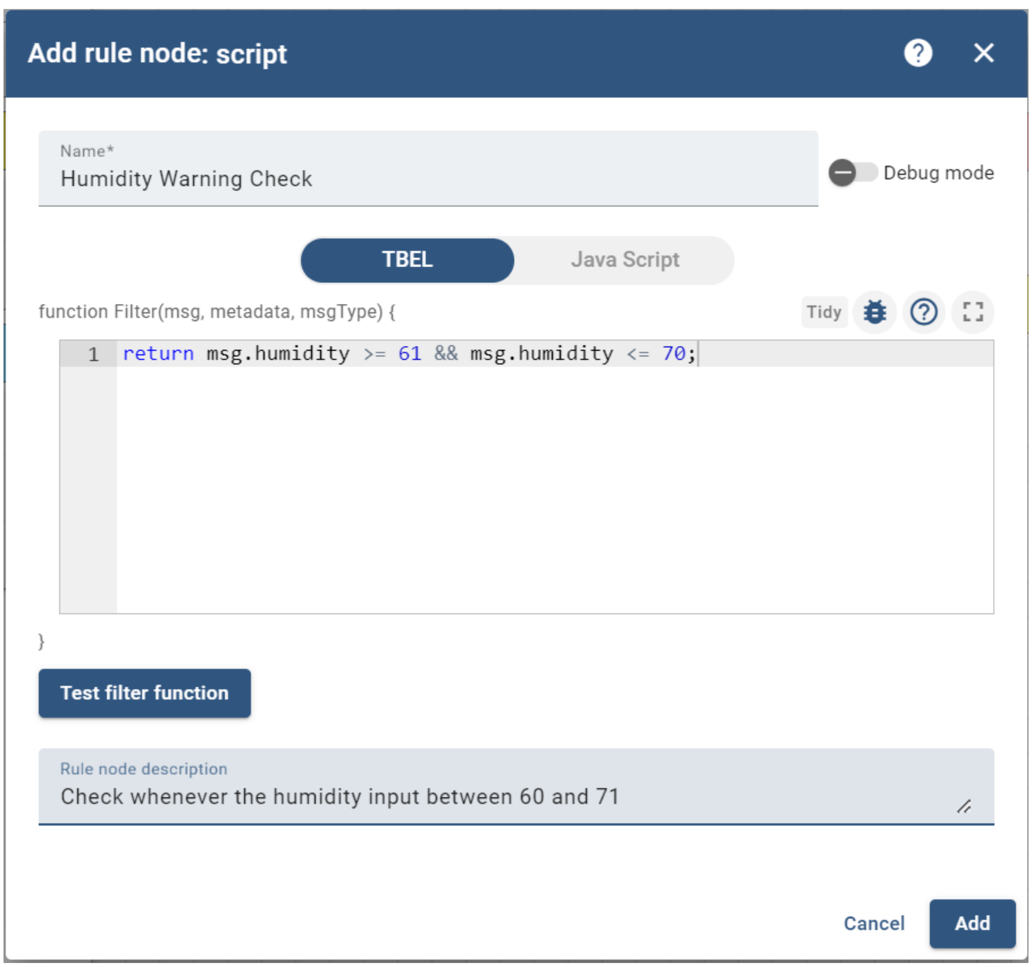


Figure 20: Create rule node: script for checking the Humidity Warning range.

Then you need to search for node "create alarm" and add the following information to create it:

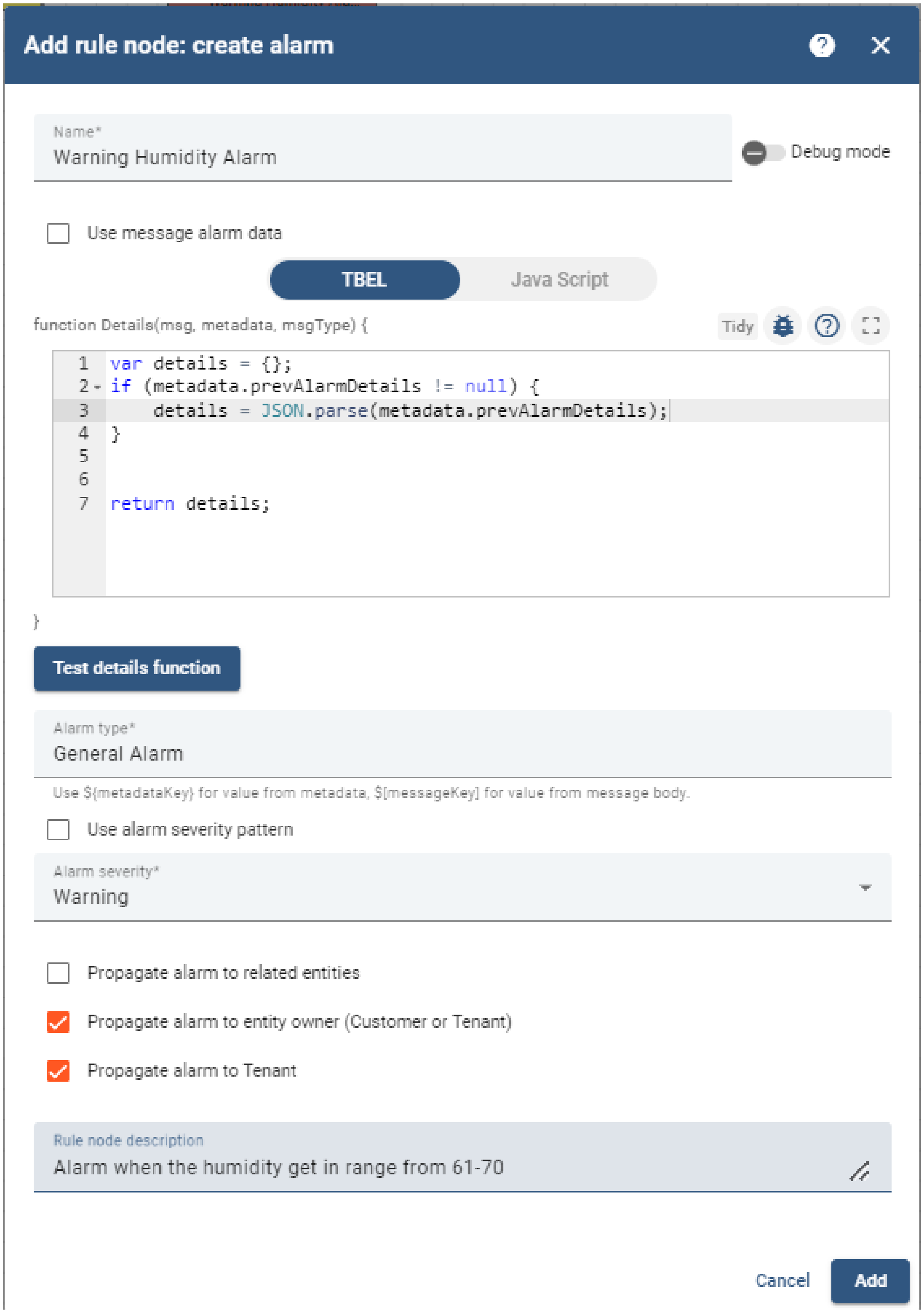


Figure 21: Create alarm for warning when humidity get in range warning

Then, we need to create for another script that used for checking whenever the humidity is too high or too low:

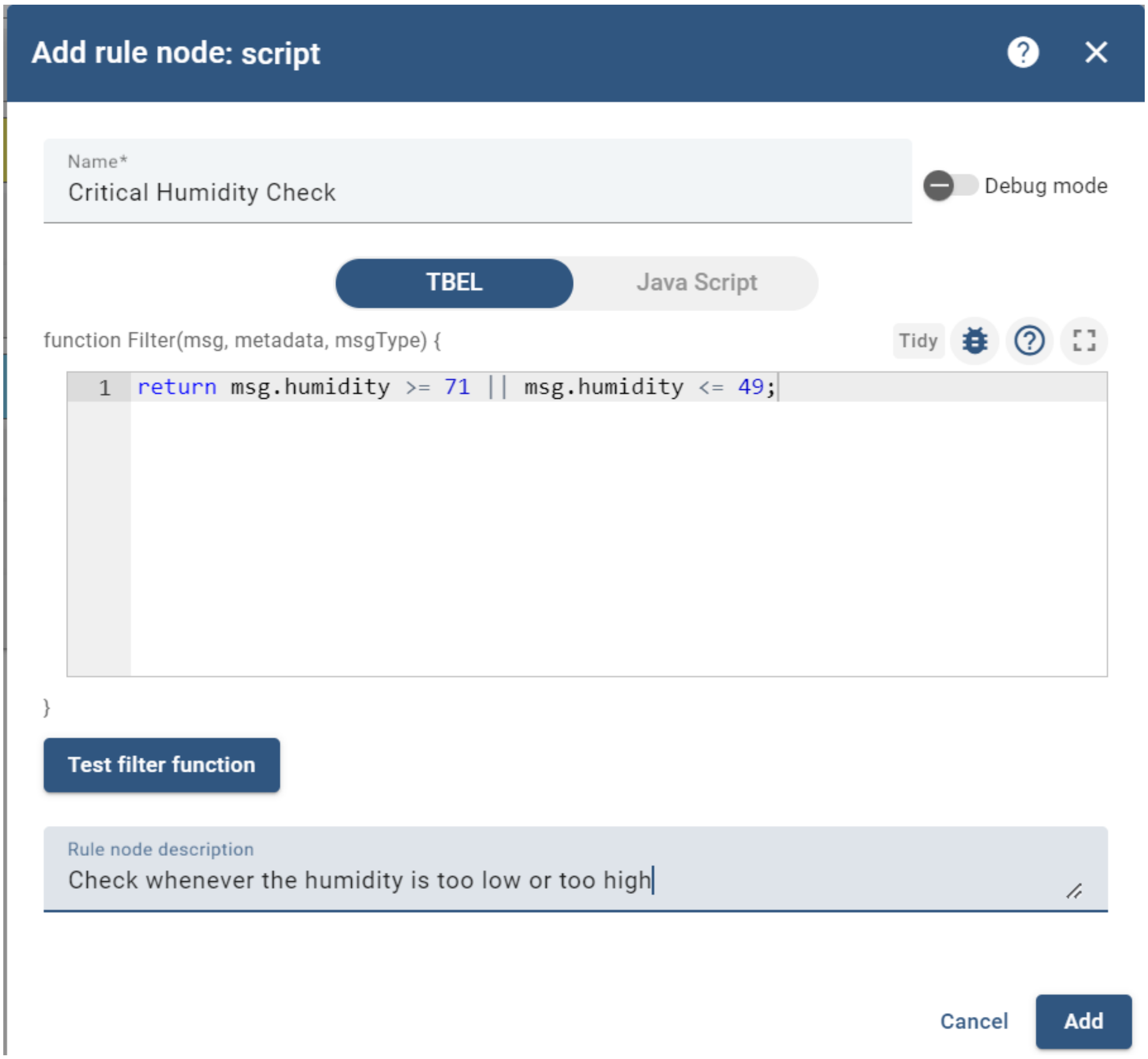


Figure 22: Critical Humidity Check node

Then, we also need to add the alarm for alerting the critical:

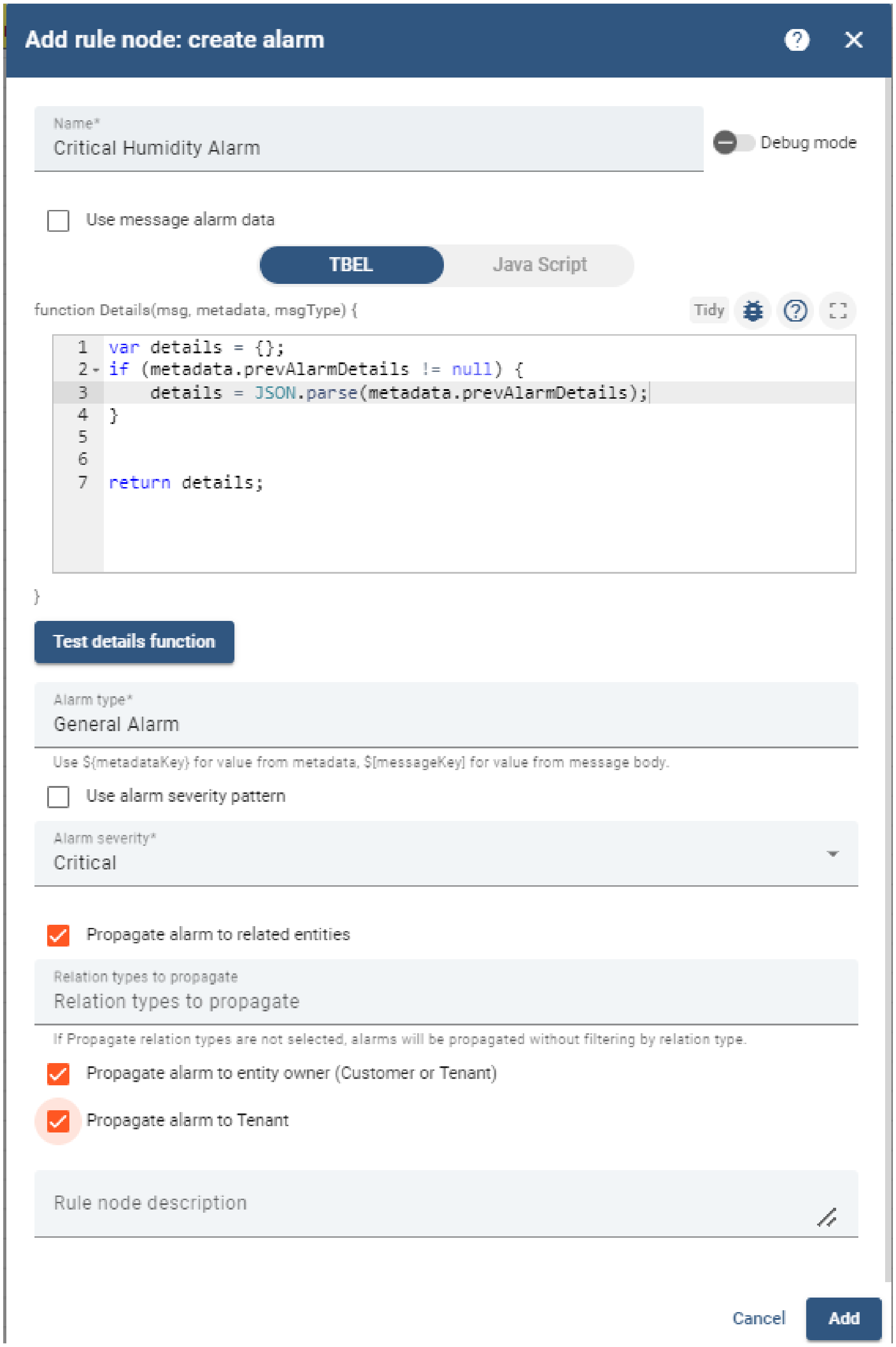


Figure 23: Adding the Critical Humidity Alarm

Then, you need to add script for checking the alarm is normal or not so it will automatically clear all the alarm:

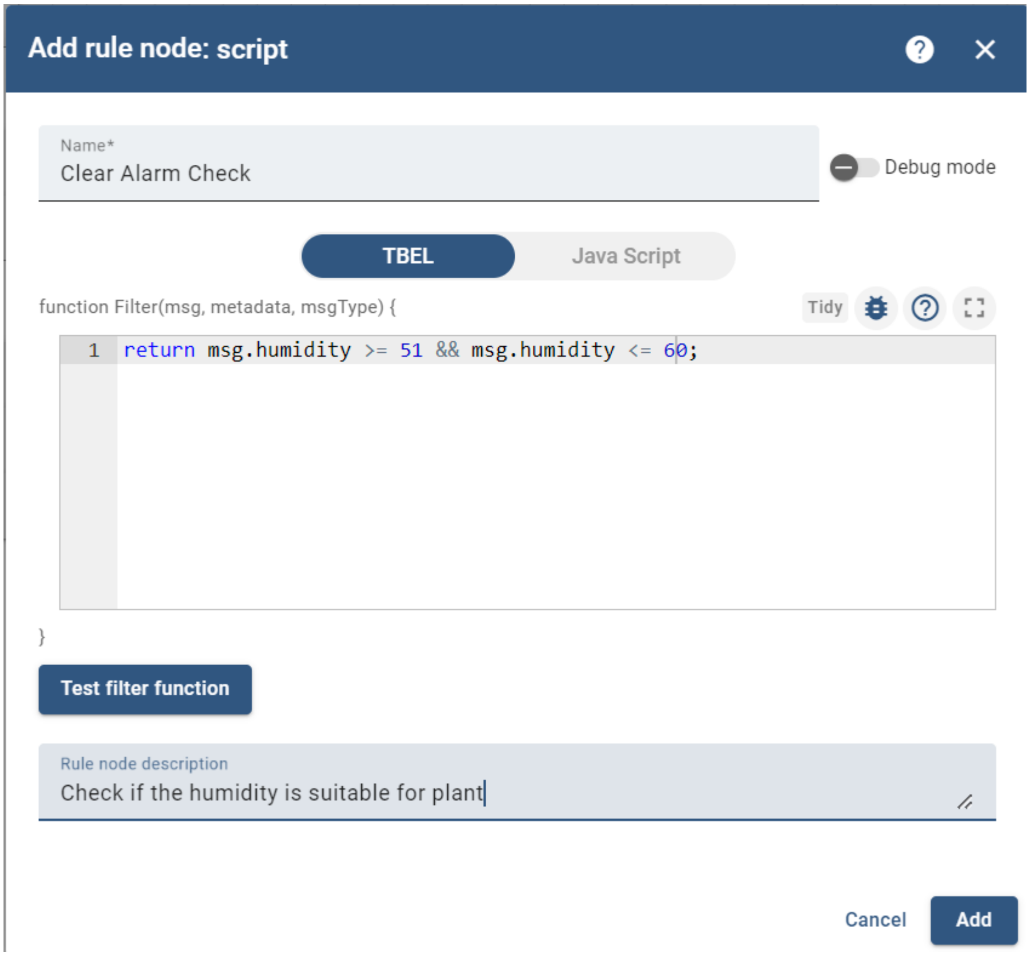


Figure 24: Creating node for checking the humidity is suitable for plants.

Then, you need to create the node for clearing all the alarm:

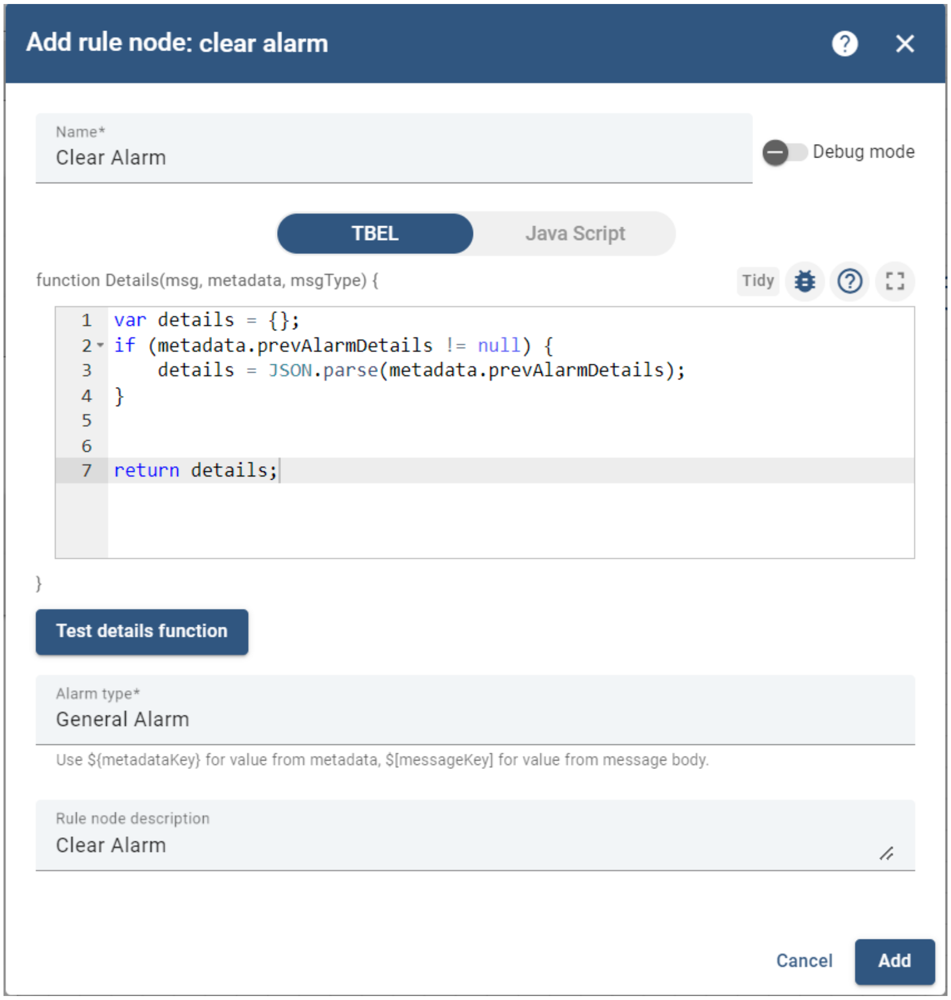


Figure 25: Adding the clear alarm node.

Lastly, you need to drag and drop the entities with the suitable conditions to be triggered when needed like the image:

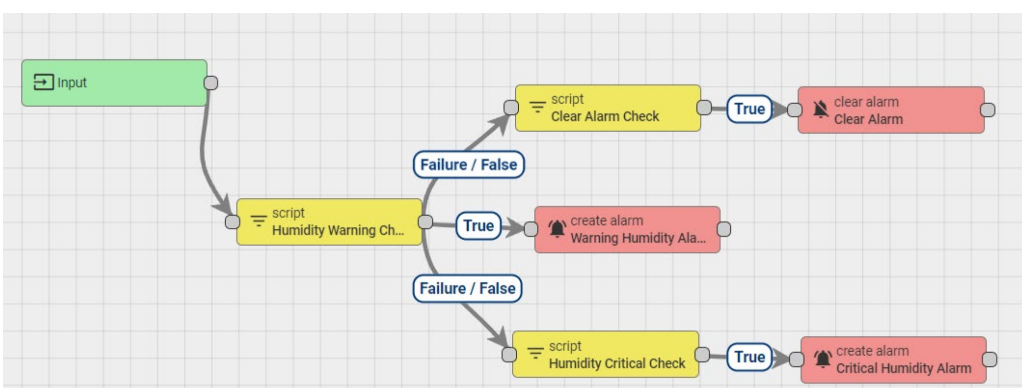


Figure 26: Create Alarm Rule Chain

Then, you need to enter the Root Rule Chain (which is created by default) and adding the “rule chain” node with the following information:

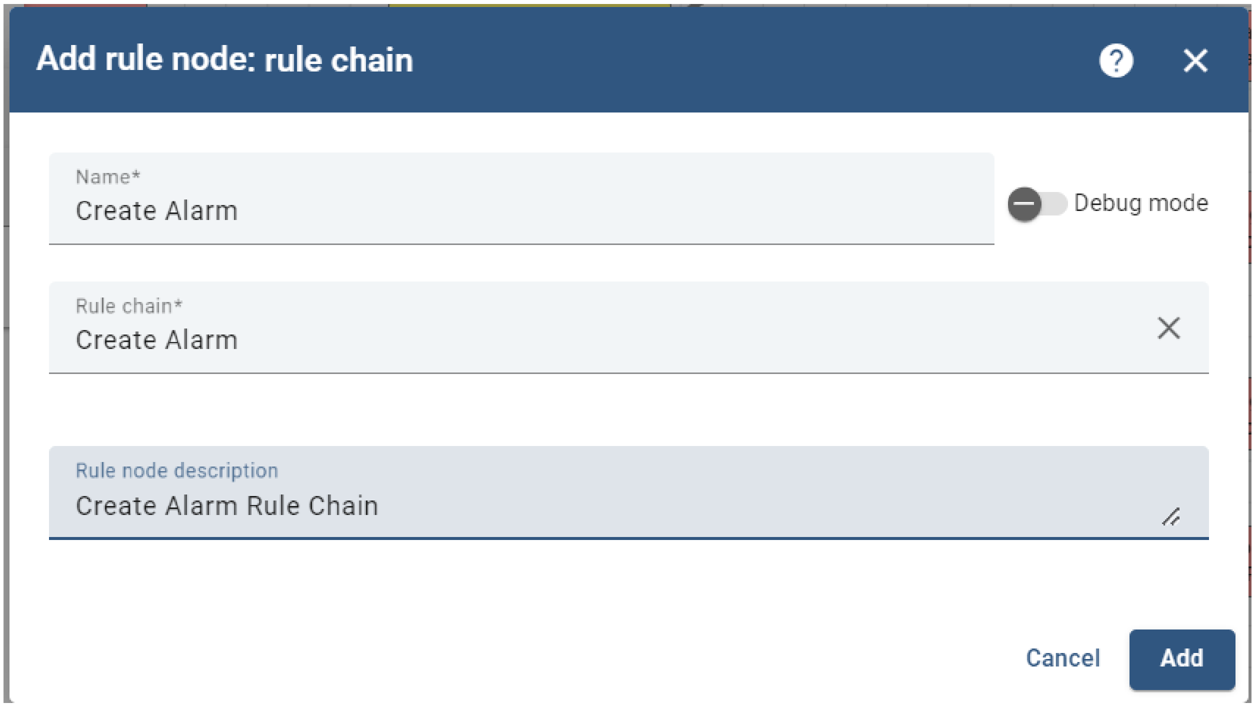


Figure 27: Adding the Creating Alarm Rule Chain Node to Root Rule Chain

Then, you will need to drag and drop and configure the conditions same with the below image:

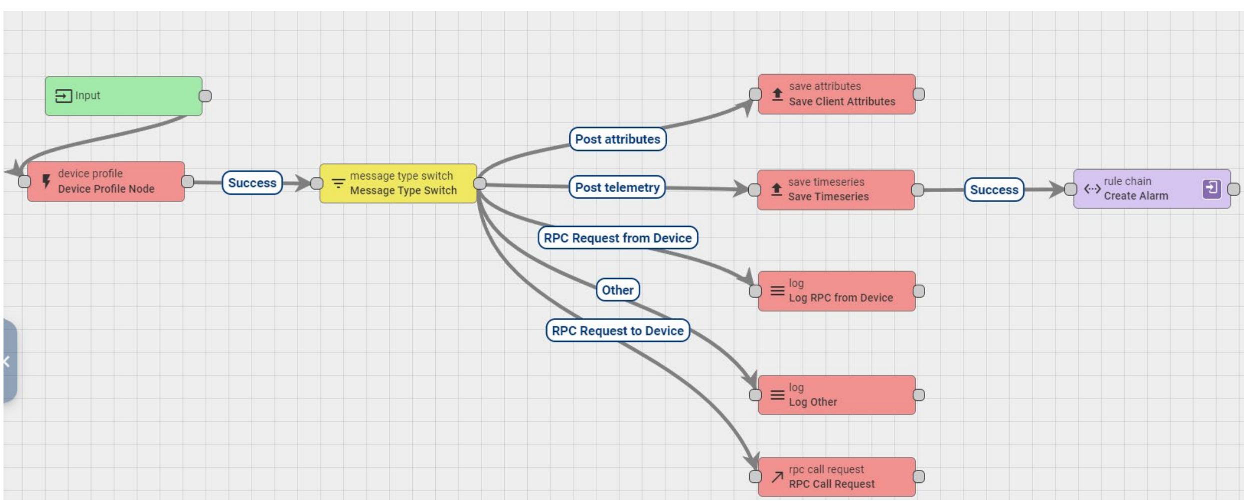


Figure 28: Root Rule Chain

8. Configure MySQL database for storage

With the following commands, you can configure MySQL on Edge server:

- sudo mysql
- CREATE USER 'pi'@'localhost';
- CREATE DATABASE sensor_data;
- GRANT ALL PRIVILEGES ON sensor_data.* TO 'pi'@'localhost';
- USE sensor_data;
- CREATE TABLE sensor_data (dataId INT AUTO_INCREMENT NOT NULL,
  temperature VARCHAR(10) NOT NULL,
  humidity VARCHAR(10) NOT NULL,
  time TIMESTAMP NOT NULL DEFAULT CURRENT_STAMP ON UPDATE CURRENT_TIMESTAMP
  PRIMARY KEY (dataId));
- exit

9. Uploading code to server and Arduino

First, you need to transfer the Arduino code from your personal device to the Arduino Uno board.

Then, you need to transfer the files to both server with the command line from your terminal:

- scp C:\Users\YourUsername\Documents\example.txt user@your_server_ip:/home/user/documents/

In my case I run the following command lines to import code to the Edge server:

- scp ./led_backend.py pi@192.168.153.132:~/webserver/
- scp ./index.html pi@192.168.153.132:~/webserver/templates/

Then, we need to enter the password of the edge server machine to import successfully import the code to the edge server.

So, when we need to run the web on edge server, we need to run the command:

- sudo python ~/webserver/led_backend.py

10. Interacting with the web UI

Base URL: http://192.168.153.132/

Command:

Note: If there is any process fail, the Arduino will receive the Serial input 4 and will trigger red light for alerting the error.

- Get current data. (URL/getData)
    - This will get the current data from the sensor and display it on the web.

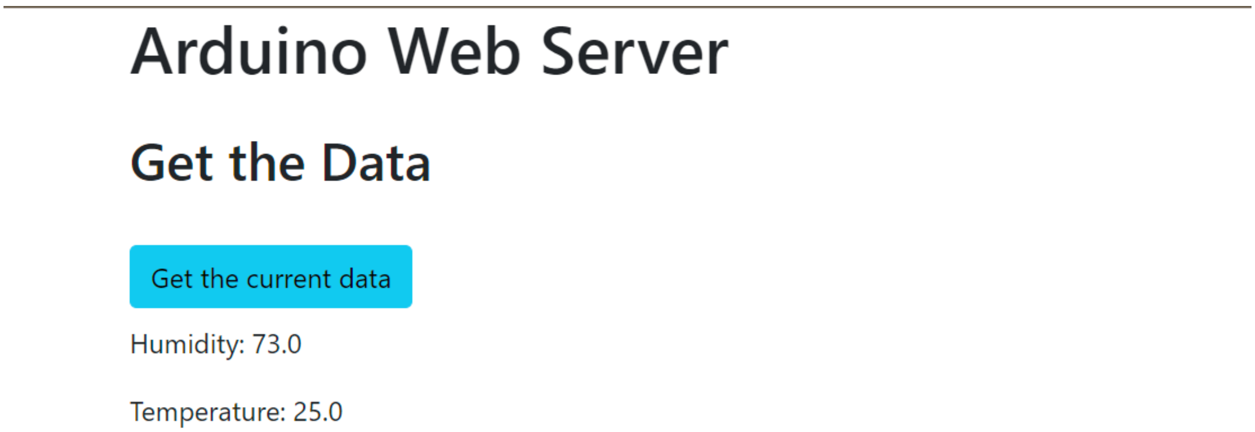


Figure 29: Get current data command.

- Run automatically. (URL/automatically)
    - This will write the Serial input 1 into the Arduino code and the system will run automatically based on the logic configured .
    - This also will automatically send the data to Cloud server foreach 2 seconds

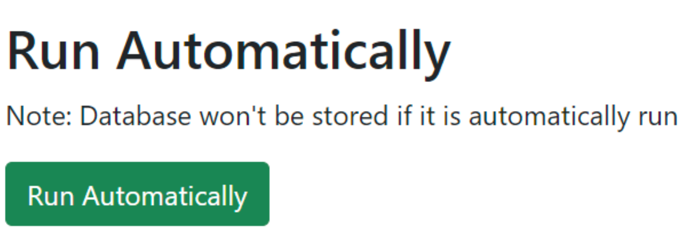


Figure 30: Automatically run command.

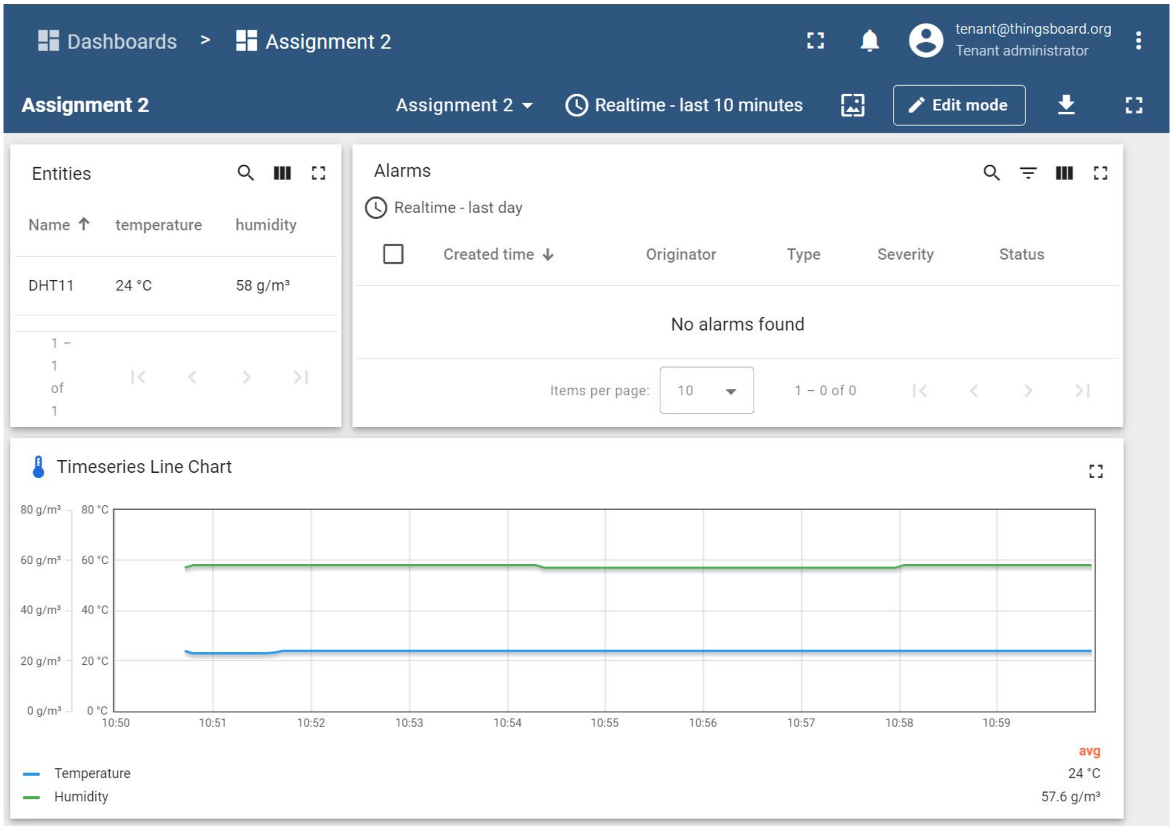


Figure 31: Data visualization

- Get data and store data into database. (URL/storeData)
    - This will include getting current data and storing it into the database. It also will write the Serial input 2 into the Arduino board and trigger green light.

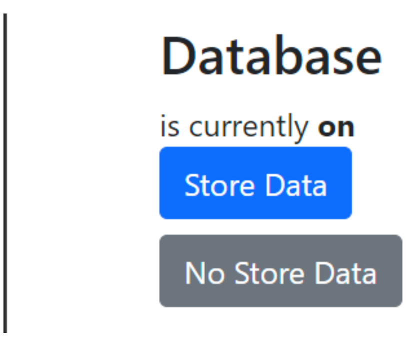


Figure 31: Store database is currently on

- Get data and no store data into database. (URL/noStoreData)

- Get the current data and turn the mode of database to off.

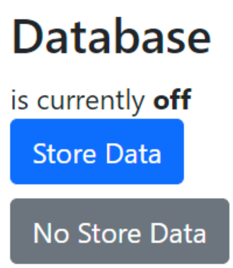
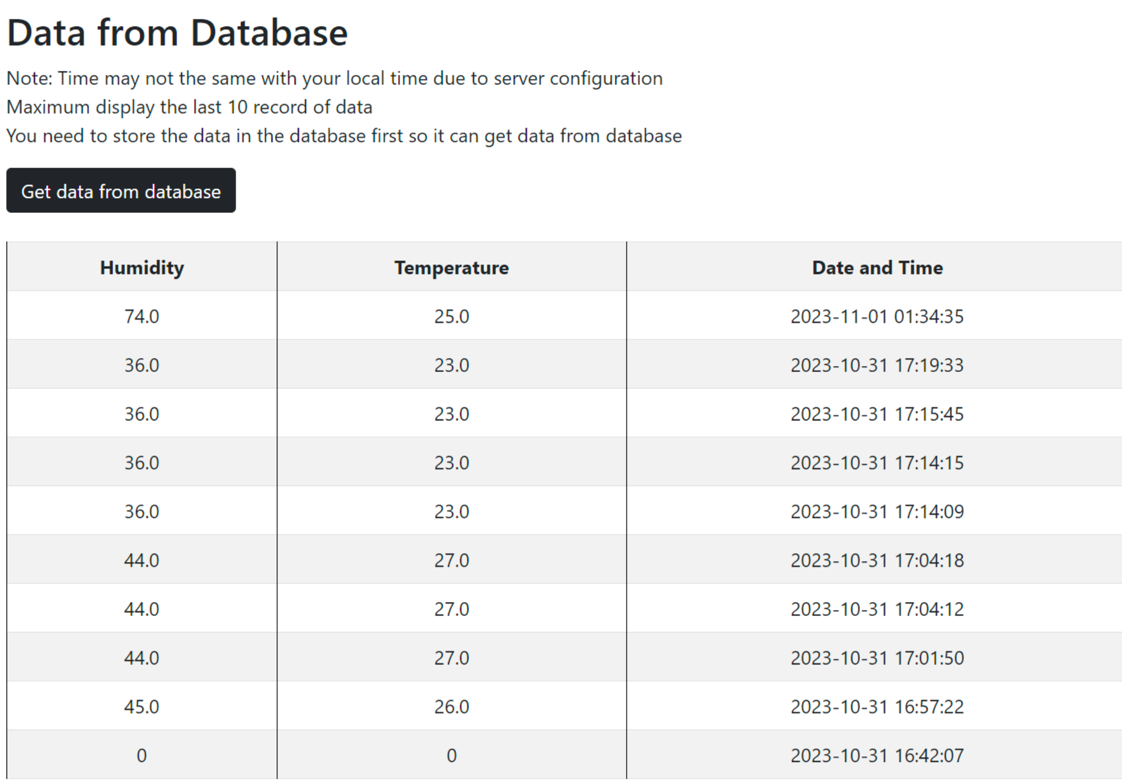


Figure 32: Current database is off.

- Get data from database. (URL/getDatabase)
  - Get the records from the database and display them on the web page.



Data from Database

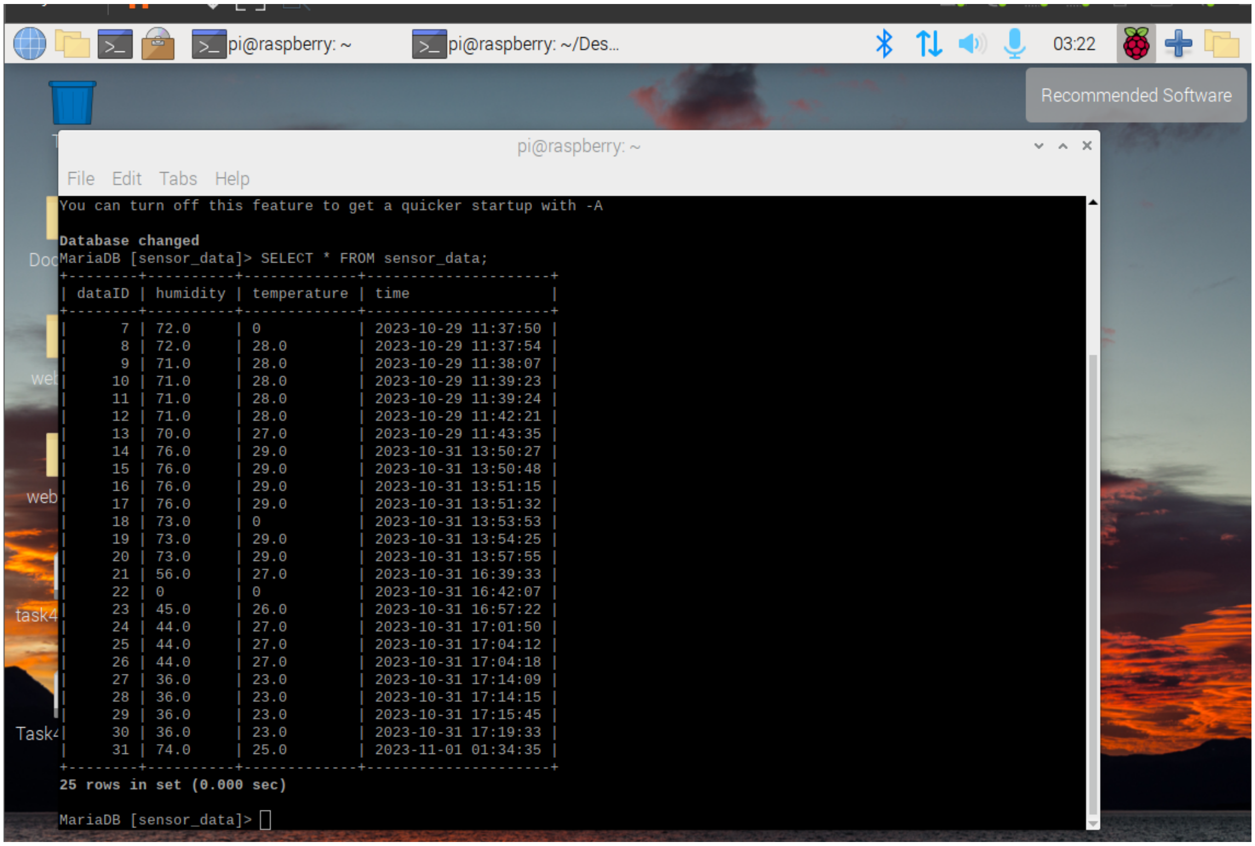
Note: Time may not the same with your local time due to server configuration
Maximum display the last 10 record of data
You need to store the data in the database first so it can get data from database

Get data from database

Humidity	Temperature	Date and Time
74.0	25.0	2023-11-01 01:34:35
36.0	23.0	2023-10-31 17:19:33
36.0	23.0	2023-10-31 17:15:45
36.0	23.0	2023-10-31 17:14:15
36.0	23.0	2023-10-31 17:14:09
44.0	27.0	2023-10-31 17:04:18
44.0	27.0	2023-10-31 17:04:12
44.0	27.0	2023-10-31 17:01:50
45.0	26.0	2023-10-31 16:57:22
0	0	2023-10-31 16:42:07

Figure 33: Table for displaying records.

Database of records:



dataID	humidity	temperature	time
7	72.0	0	2023-10-29 11:37:50
8	72.0	28.0	2023-10-29 11:37:54
9	71.0	28.0	2023-10-29 11:38:07
10	71.0	28.0	2023-10-29 11:39:23
11	71.0	28.0	2023-10-29 11:39:24
12	71.0	28.0	2023-10-29 11:42:21
13	70.0	27.0	2023-10-29 11:43:35
14	76.0	29.0	2023-10-31 13:50:27
15	76.0	29.0	2023-10-31 13:50:48
16	76.0	29.0	2023-10-31 13:51:15
17	76.0	29.0	2023-10-31 13:51:32
18	73.0	0	2023-10-31 13:53:53
19	73.0	29.0	2023-10-31 13:54:25
20	73.0	29.0	2023-10-31 13:57:55
21	56.0	27.0	2023-10-31 16:39:33
22	0	0	2023-10-31 16:42:07
23	45.0	26.0	2023-10-31 16:57:22
24	44.0	27.0	2023-10-31 17:01:50
25	44.0	27.0	2023-10-31 17:04:12
26	44.0	27.0	2023-10-31 17:04:18
27	36.0	23.0	2023-10-31 17:14:09
28	36.0	23.0	2023-10-31 17:14:15
29	36.0	23.0	2023-10-31 17:15:45
30	36.0	23.0	2023-10-31 17:19:33
31	74.0	25.0	2023-11-01 01:34:35

Figure 34: Show the records from database with query.

Command line to run server:

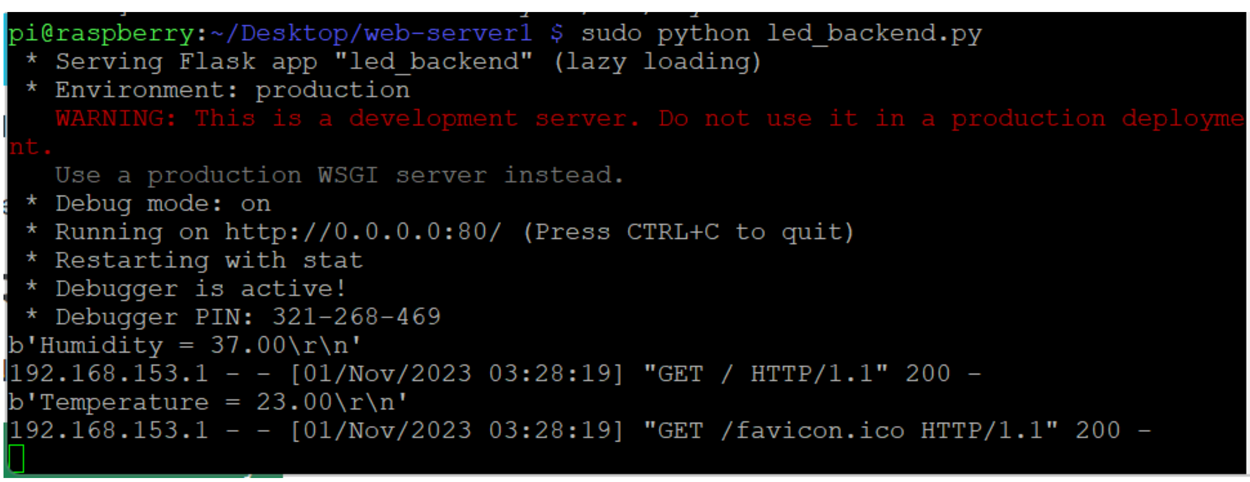


Figure 35: Run server.

VI. Limitations

1. Dependency on Network Stability if implement this system in real life:

The system's heavy reliance on network connectivity raises concerns about its resilience in the face of fluctuating network stability. Any disruptions in the network could jeopardize the real-time monitoring capabilities and hinder the seamless transmission of data between sensors, the edge server, cloud server, and clients.

2. Potential Security Risks:

Security emerges as a critical consideration, particularly with the use of MQTT for data transmission. Without stringent security measures, the system may be susceptible to unauthorized access or tampering, emphasizing the need for robust encryption and secure communication channels.

3. Scalability Challenges:

While the system is envisioned to be scalable, challenges may arise as the number of sensors and actuators increases. Ensuring efficient communication and processing for a larger network of devices requires meticulous planning to maintain scalability without compromising performance.

4. Reliability on Cloud Services:

The system's reliability is contingent on the stability and availability of cloud services. Any downtime or disruptions in cloud services could impede data visualization, alarm triggers, and the overall functionality of the system.

5. Limited Offline Functionality:

A potential limitation arises in the system's lack of robust offline functionality. During network outages, the system may struggle to provide real-time alerts and visualizations, highlighting the importance of developing contingency plans for periods of connectivity loss.

6. **Complex Implementation and Maintenance:**

The integration of various components introduces complexity in both implementation and ongoing maintenance. Regular updates and maintenance efforts are essential to navigate potential complexities and ensure the system's continued smooth operation.

7. **Sensor Limitations:**

The choice of DHT22 sensors for humidity measurement may present challenges in terms of accuracy and responsiveness. Additionally, the reliance on humidity triggers may not offer precise control, as humidity levels may not align perfectly with plant preferences in all scenarios.

8. **Dependence on ThingsBoard:**

The functionality of the system hinges on the seamless integration with ThingsBoard. Any issues with ThingsBoard services or changes in its functionality could impact data visualization and alarm triggers, highlighting a potential point of vulnerability.

9. **Limited Customization for Trigger Circumstances:**

While the system allows for the adjustment of humidity based on trigger circumstances, its capacity for handling diverse and complex scenarios may be limited. Fine-tuning trigger conditions requires careful consideration to ensure optimal system responsiveness.

10. **Data Storage and Retrieval Efficiency:**

The use of MySQL storage for data management raises concerns about the system's efficiency as data volume increases. Optimizing database performance becomes crucial to maintain the system's responsiveness in handling substantial amounts of data.

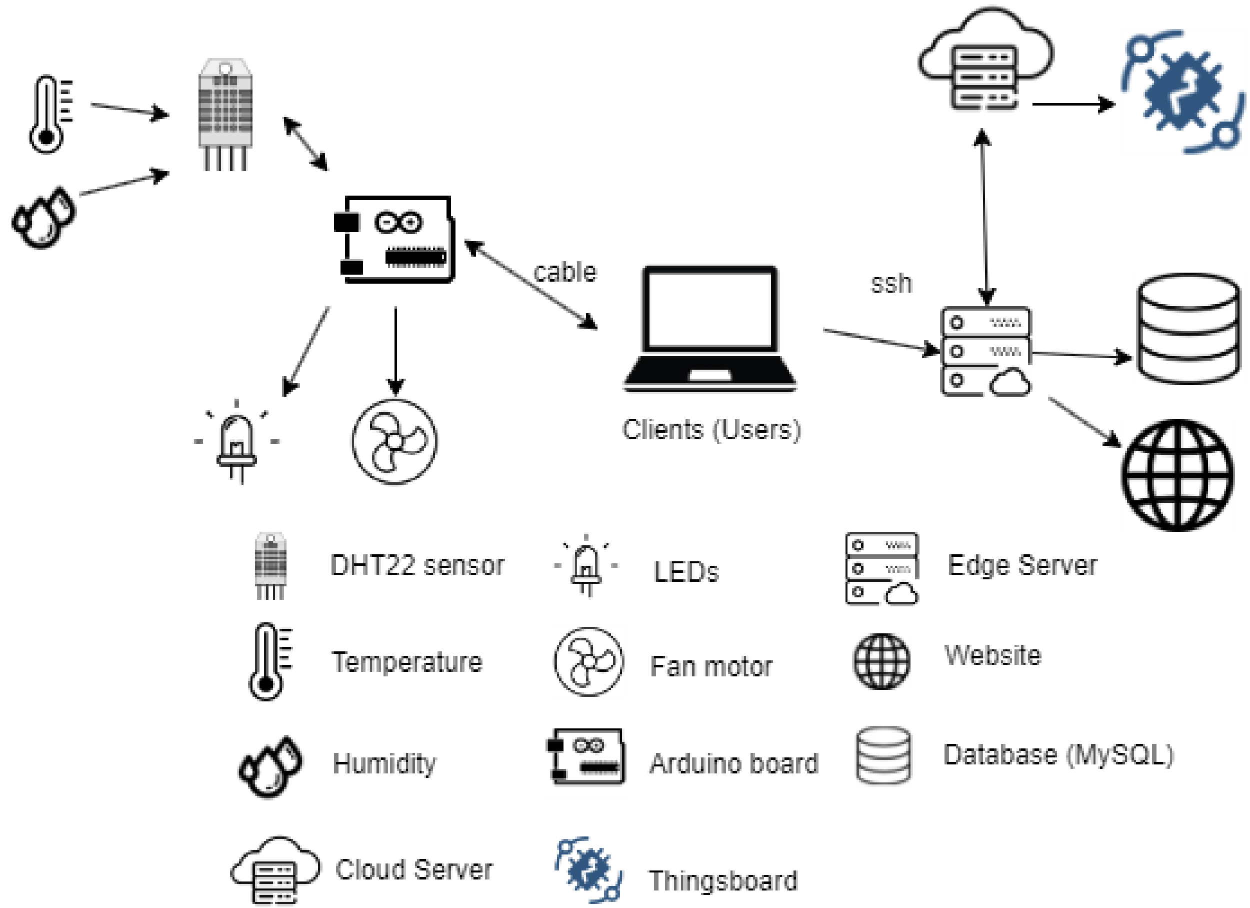
VII. Resources

Circuit Basics (2023). *How to Use a DHT11 Humidity Sensor on the Arduino - Ultimate Guide to the Arduino #38*. *YouTube*. Available at: https://www.youtube.com/watch?v=dJJAQxyryoQ [Accessed 25 Oct. 2023].

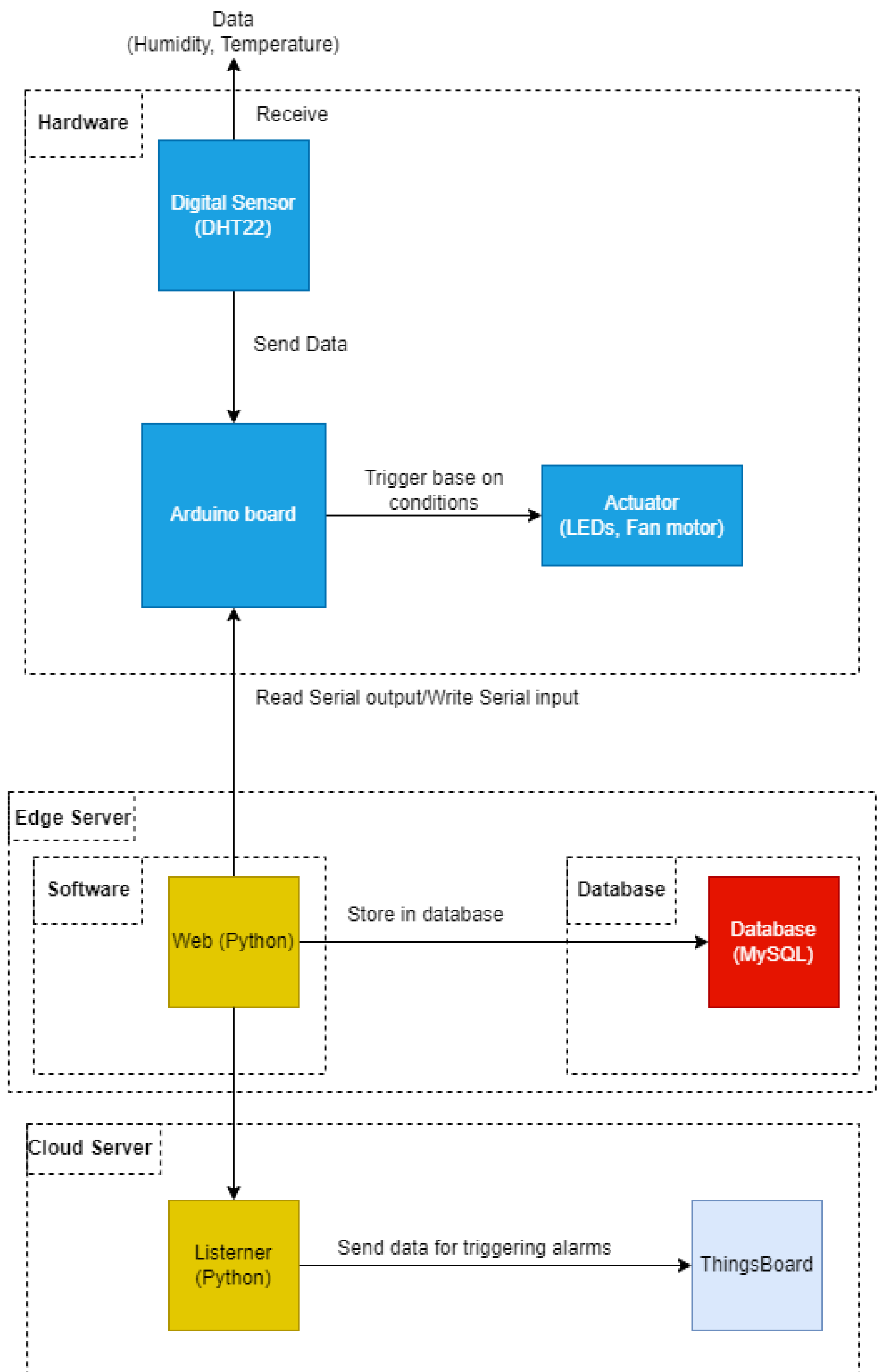
PLNTS.com. (2021). *Temperature and humdity*. [online] Available at: https://plnts.com/en/care/doctor/temperature-and-humidity [Accessed 28 Oct. 2023].

VIII. Appendix

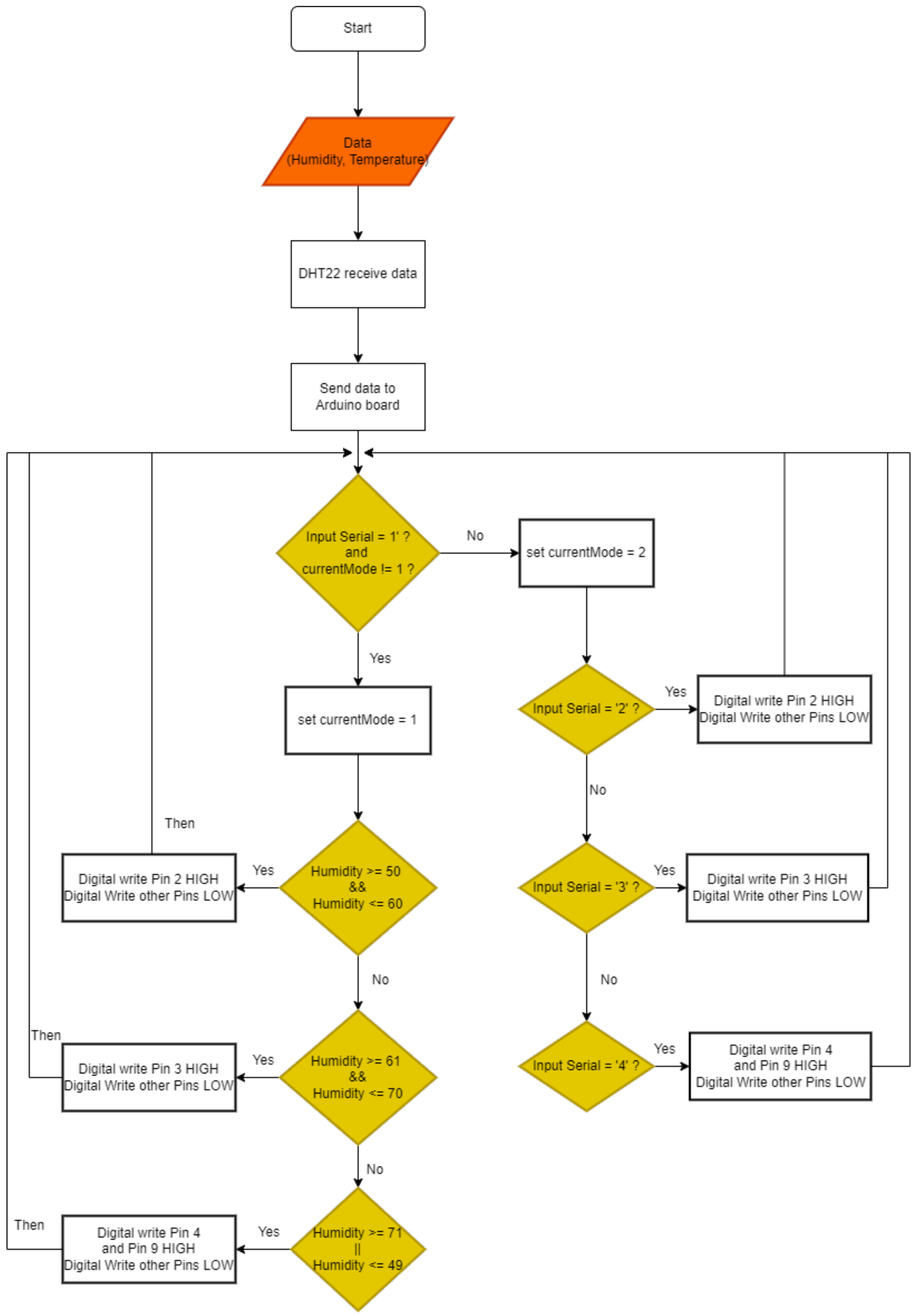
1. Sketch of proposed system



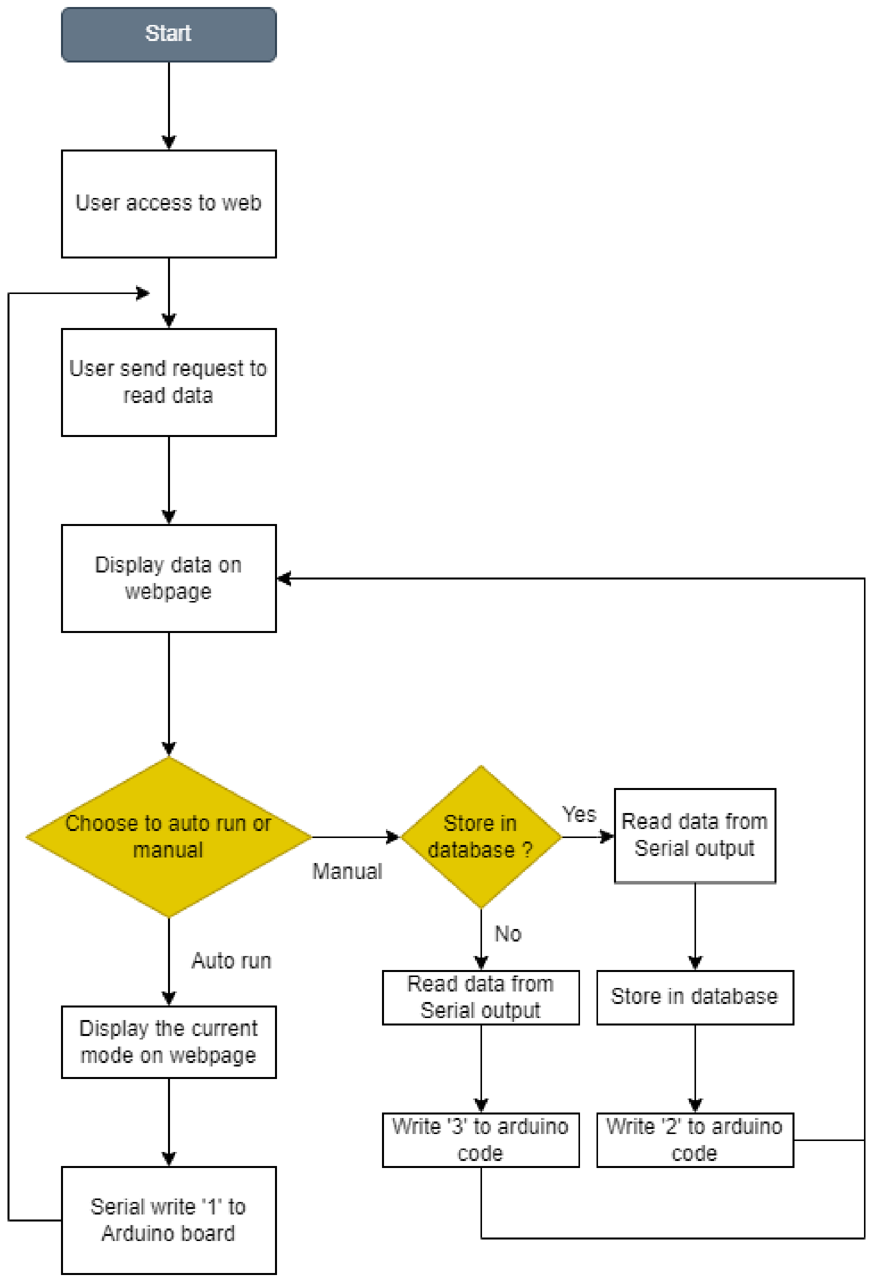
2. Block diagram for proposed system



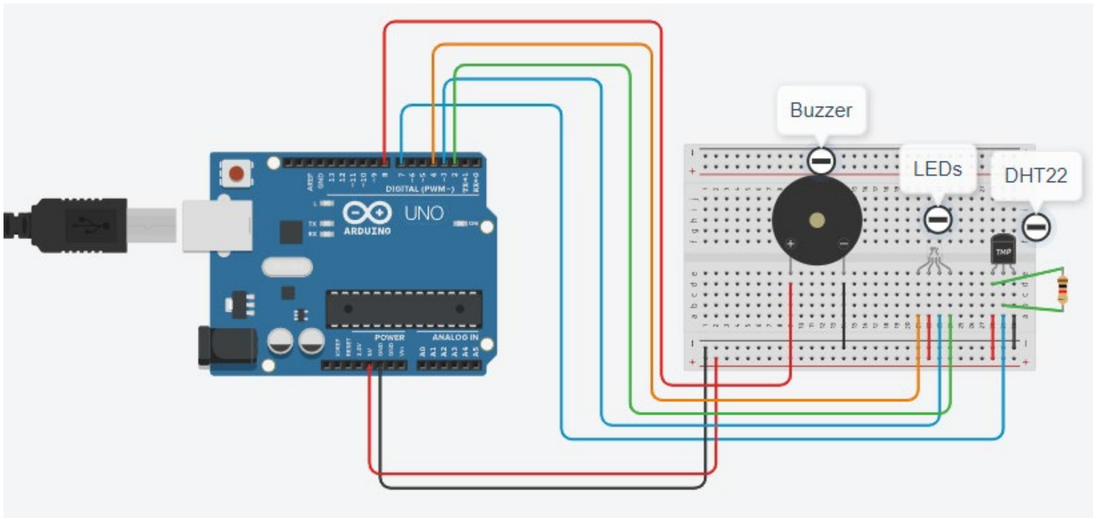
3. Flow chart for how Arduino system



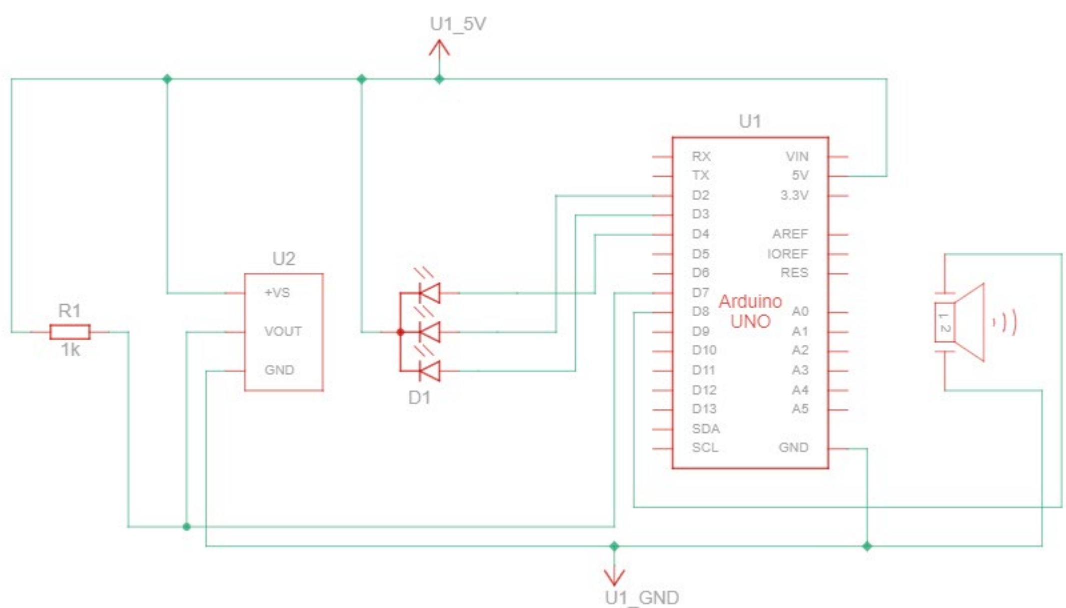
4. Flow chart for Python code and send request.



5. Tinkercad model of proposed system



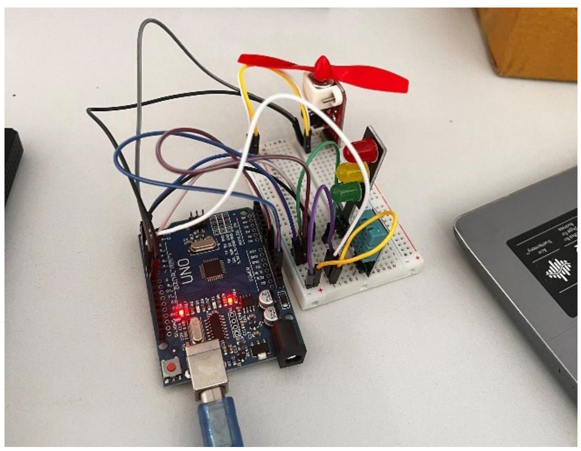
6. Schematic view of model Tinkercad



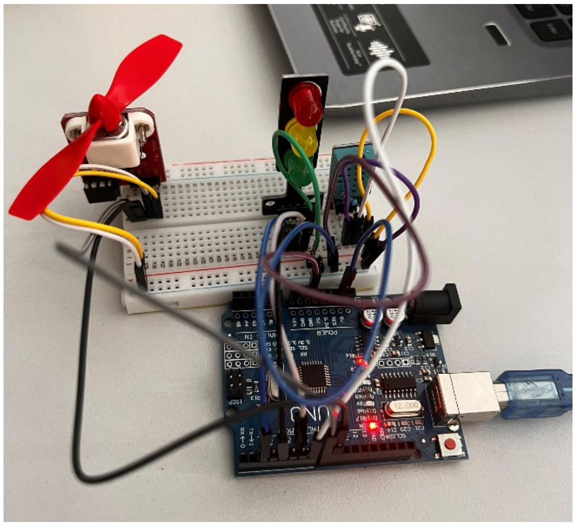
7. SQL query for creating table on Edge Server

```
CREATE TABLE sensor_data (
   dataID INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
   humidity VARCHAR(10) NOT NULL,
   temperature VARCHAR(10) NOT NULL,
   time TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

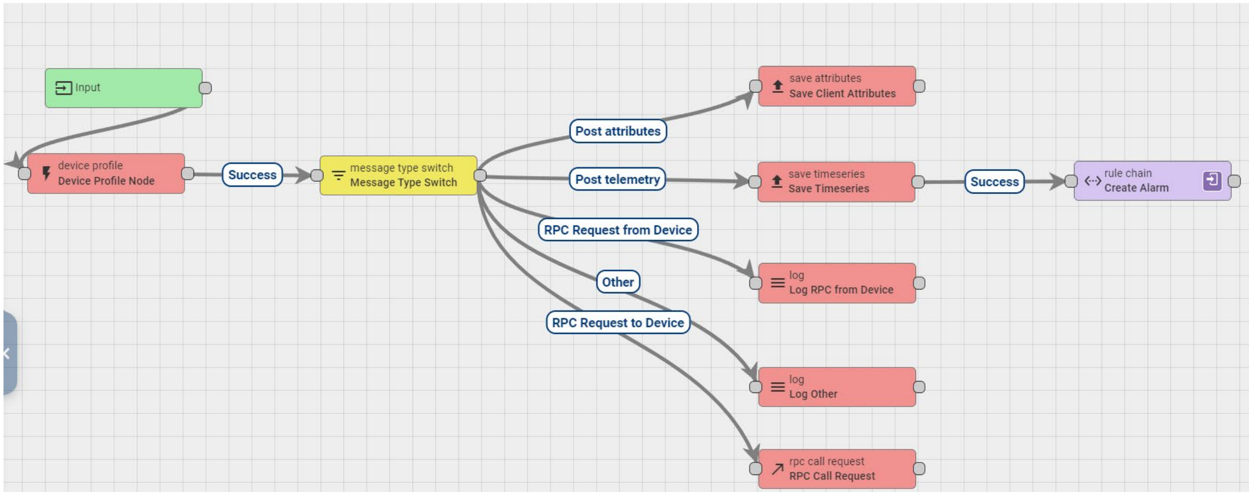
8. Side view of physical system



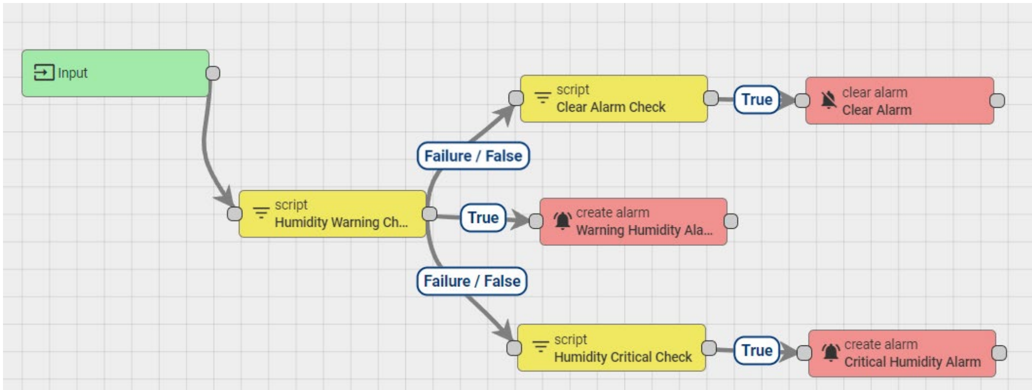
9. Front view of physical system



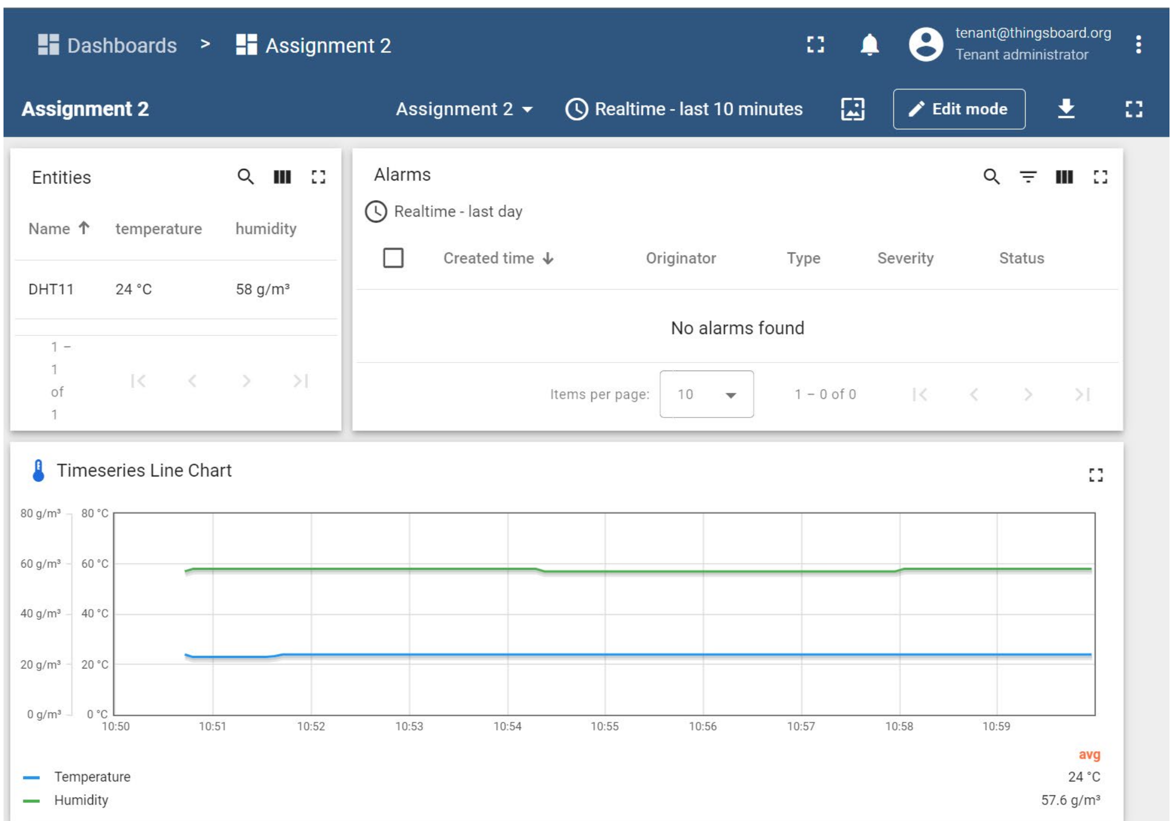
10. Root Rule Chain



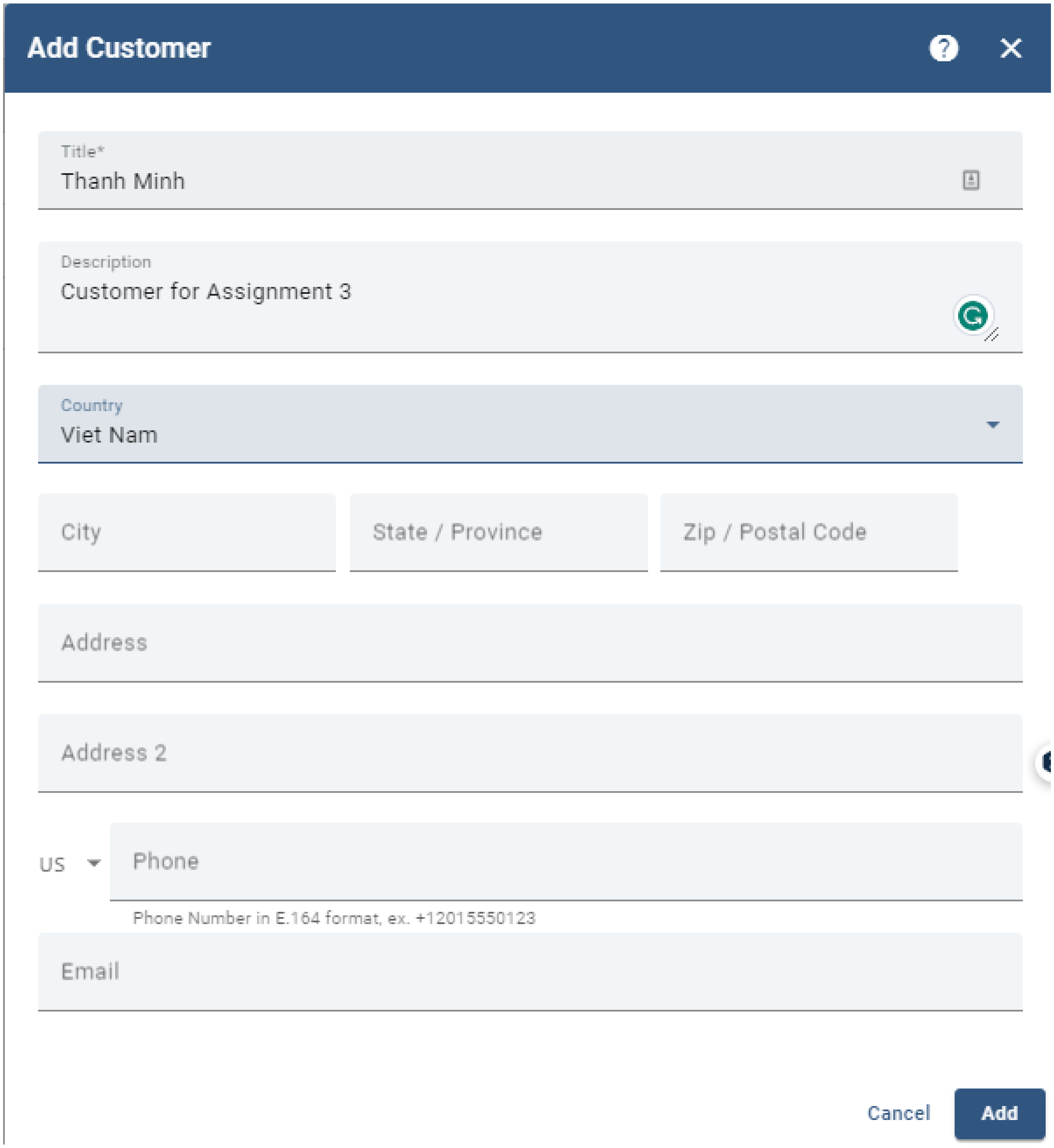
11. Rule Chain for Creating Alarm



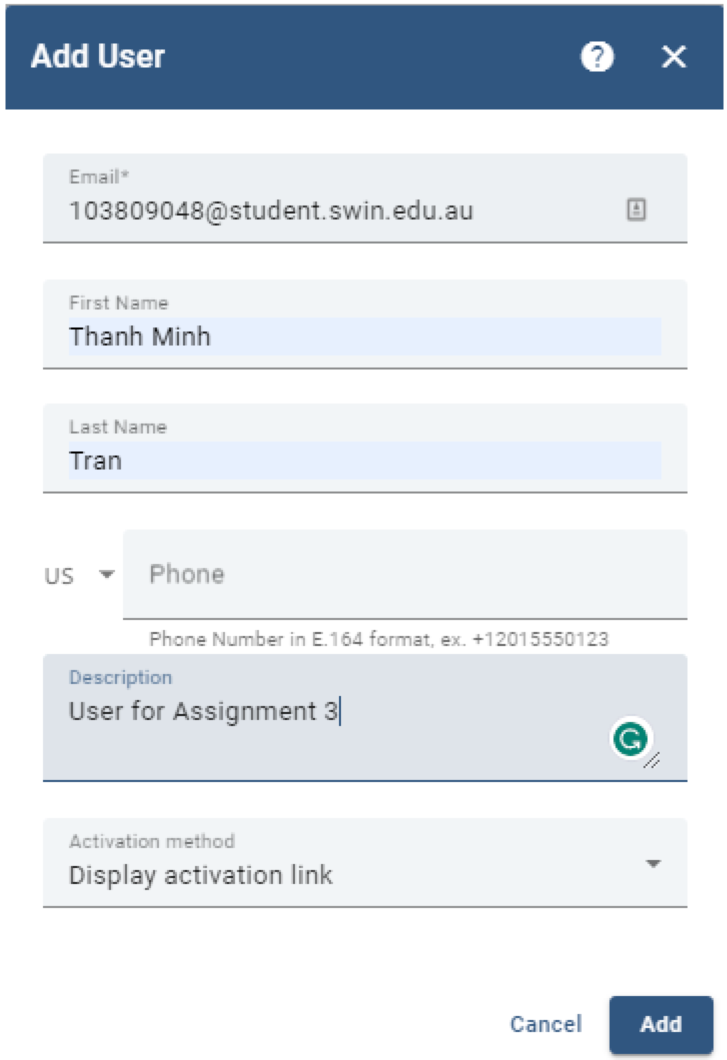
12. Assignment 2 Dashboard



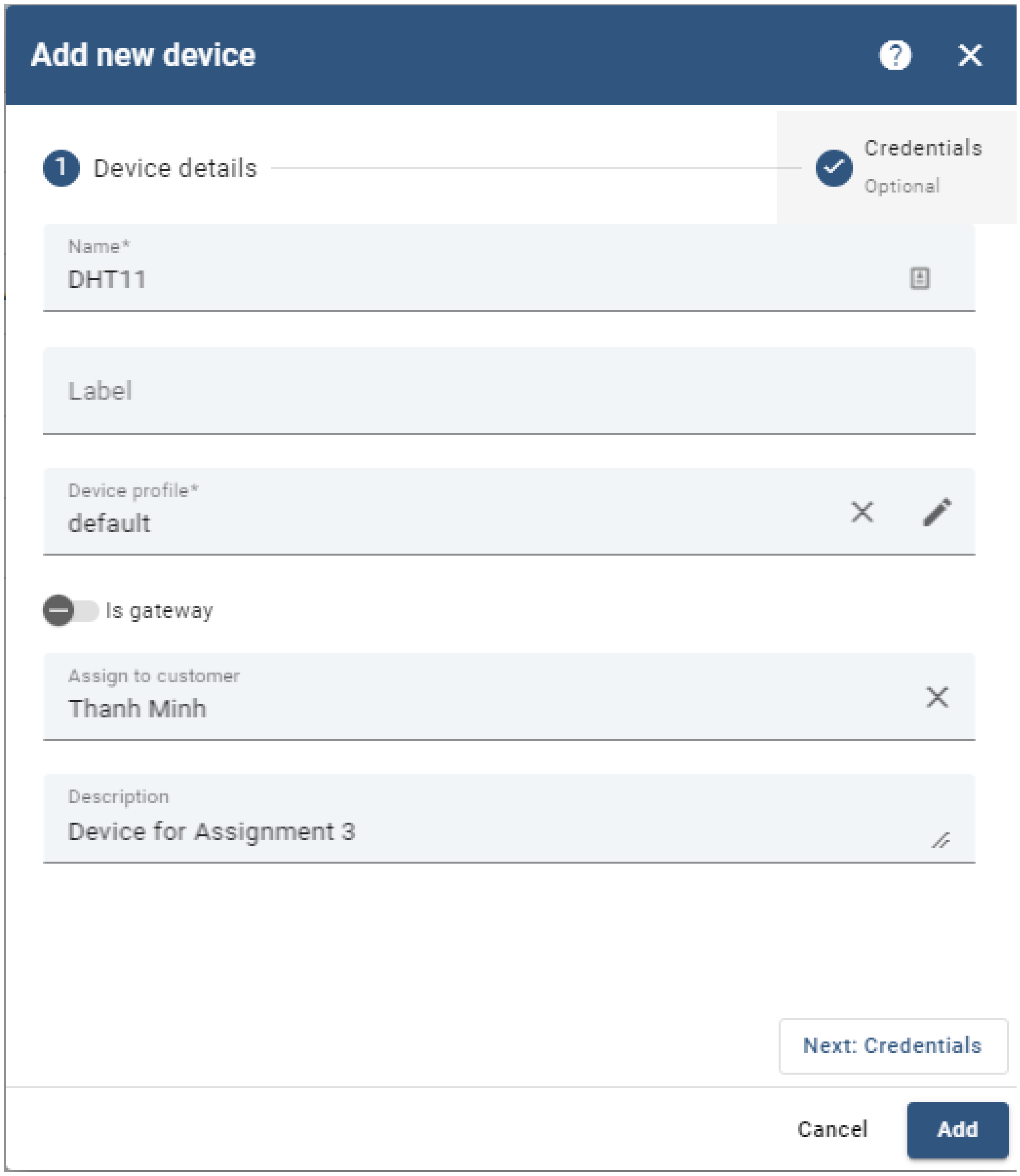
13. Add new customer Thanh Minh



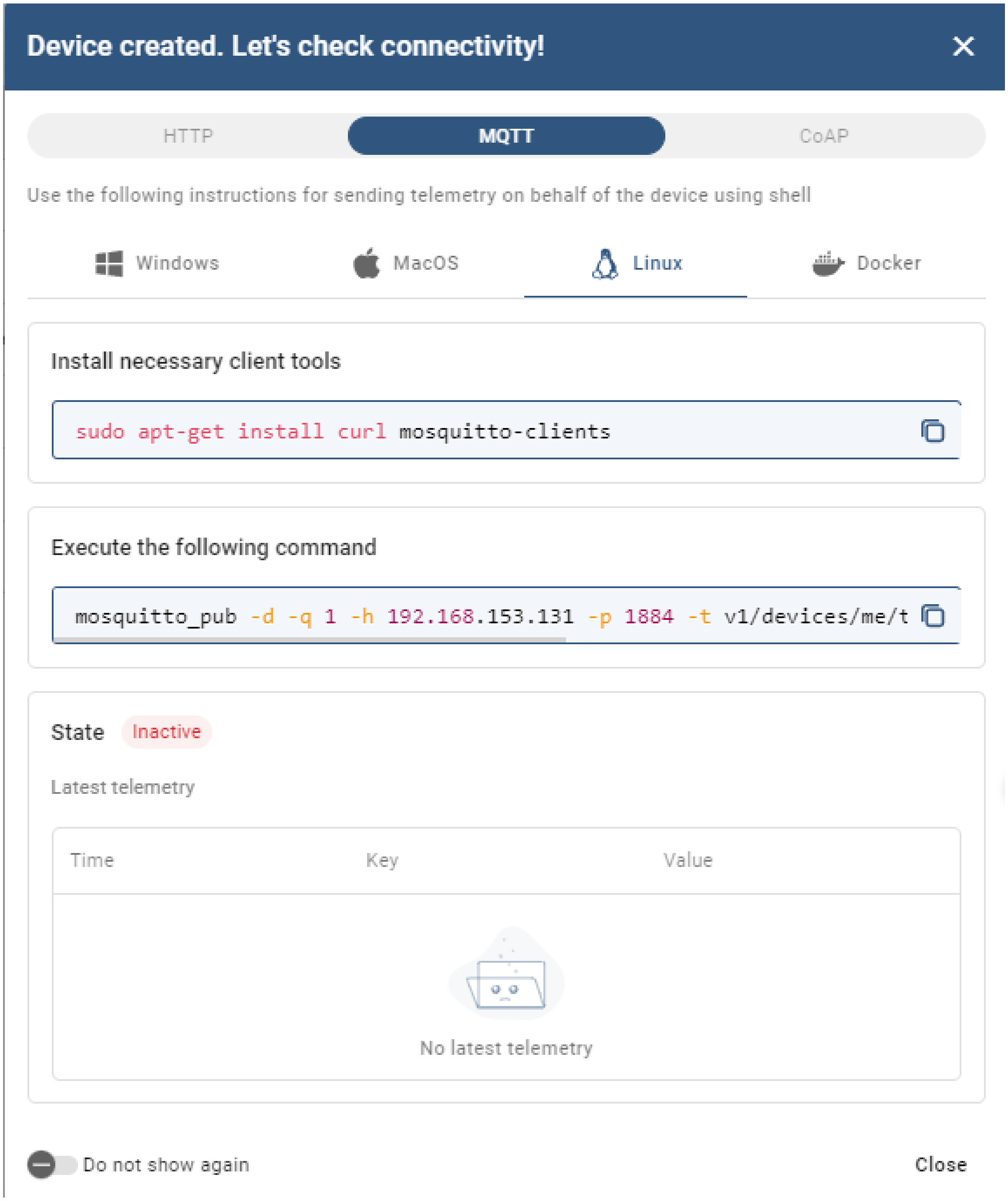
14. Assign user to customer



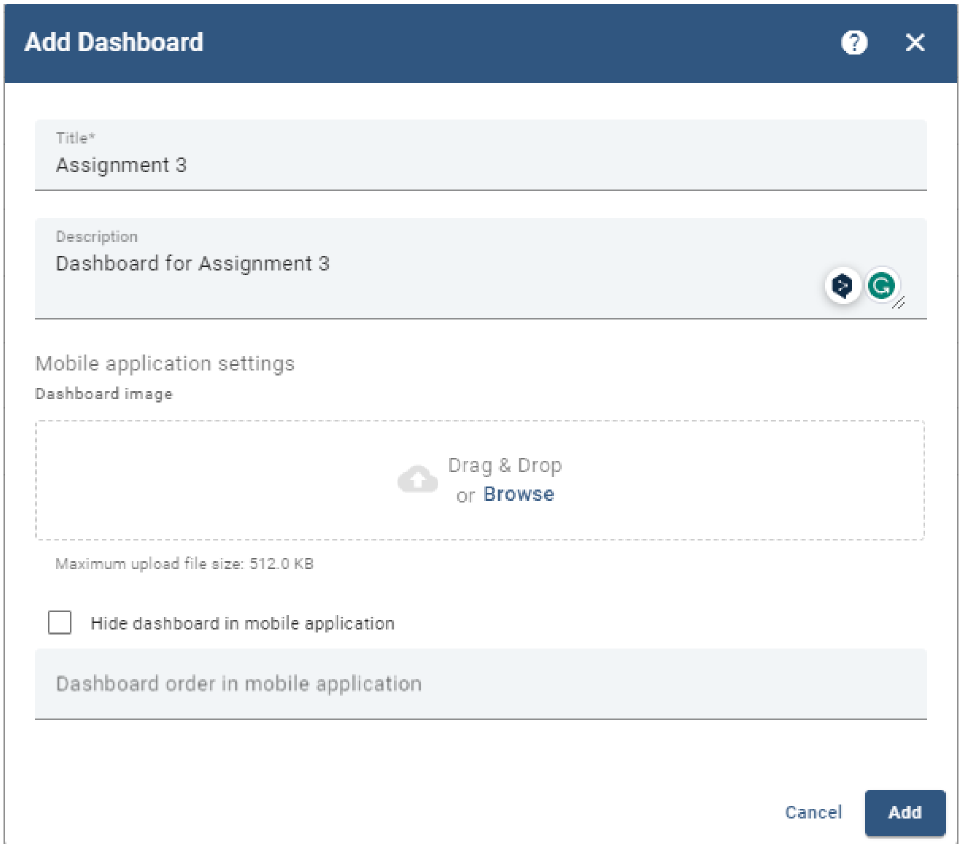
15. DHT11 device for assignment 3



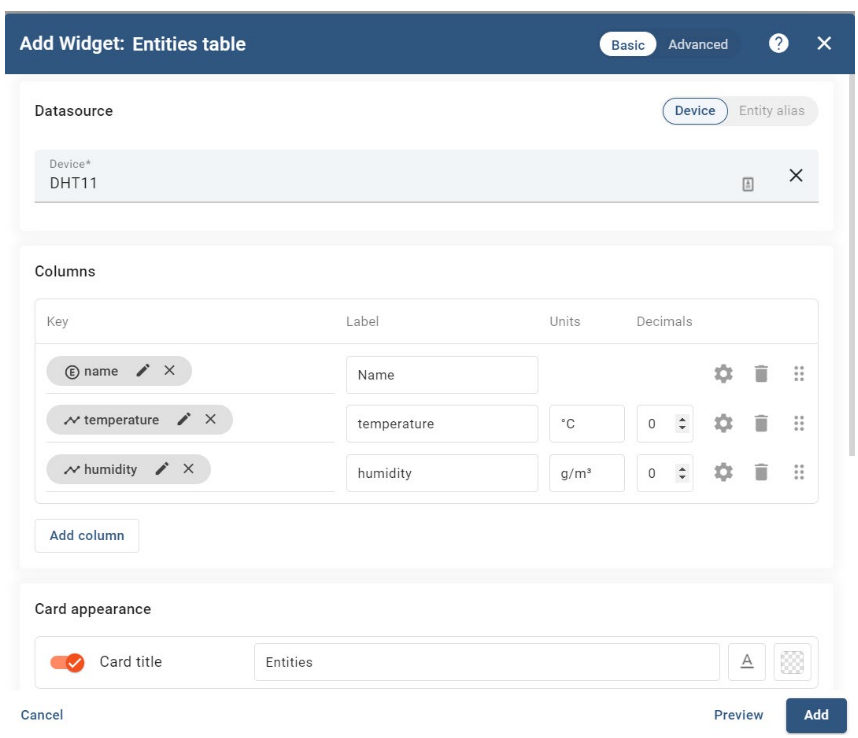
16. Instruction of interacting with Thingsboard



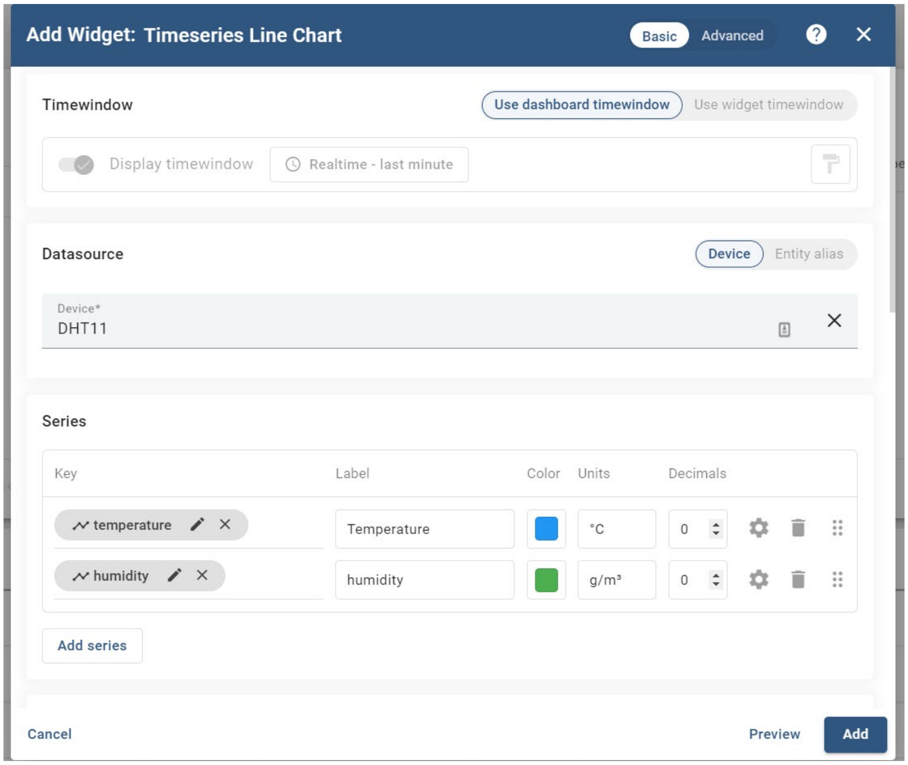
17. Creating Dashboard for Assignment 3



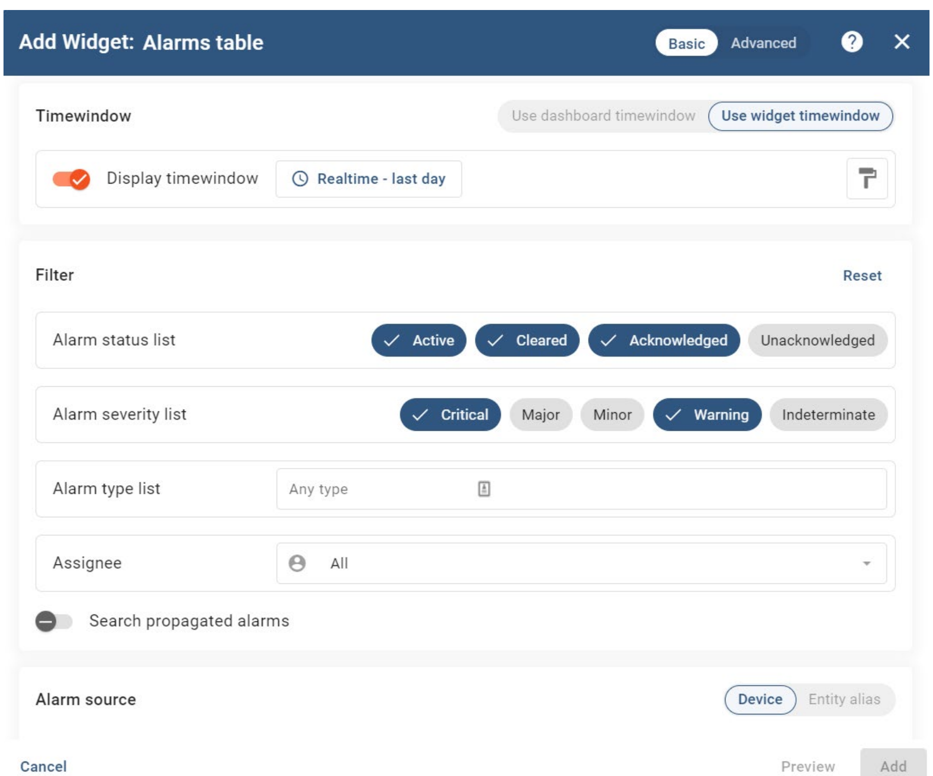
18. Creating Entities table widget



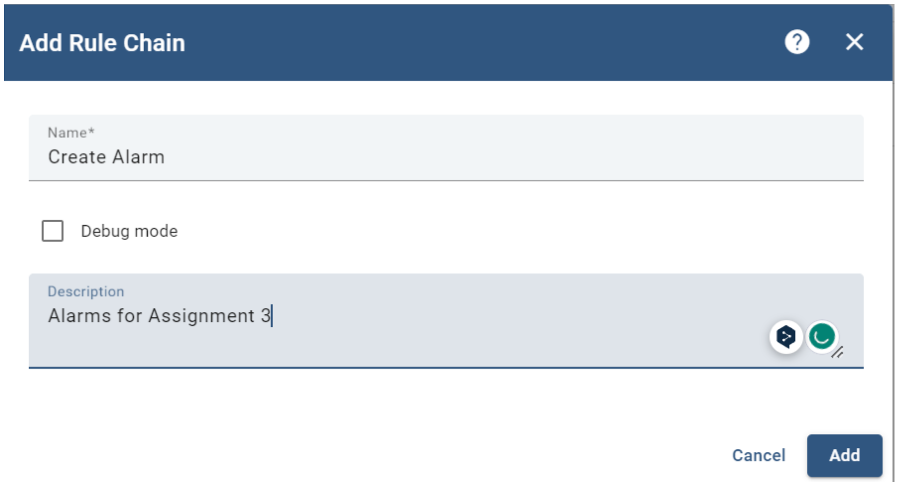
19. Creating Timeseries Line Chart



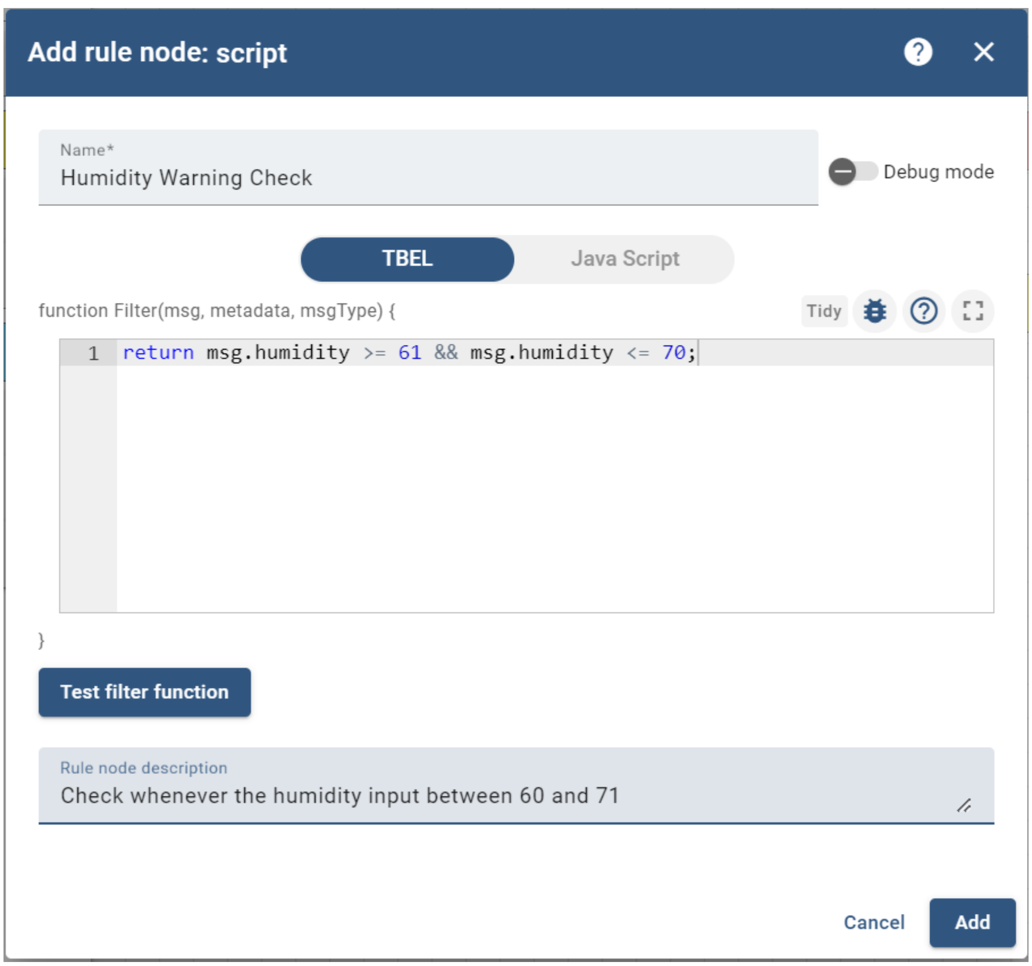
20. Creating Alarm Table widget



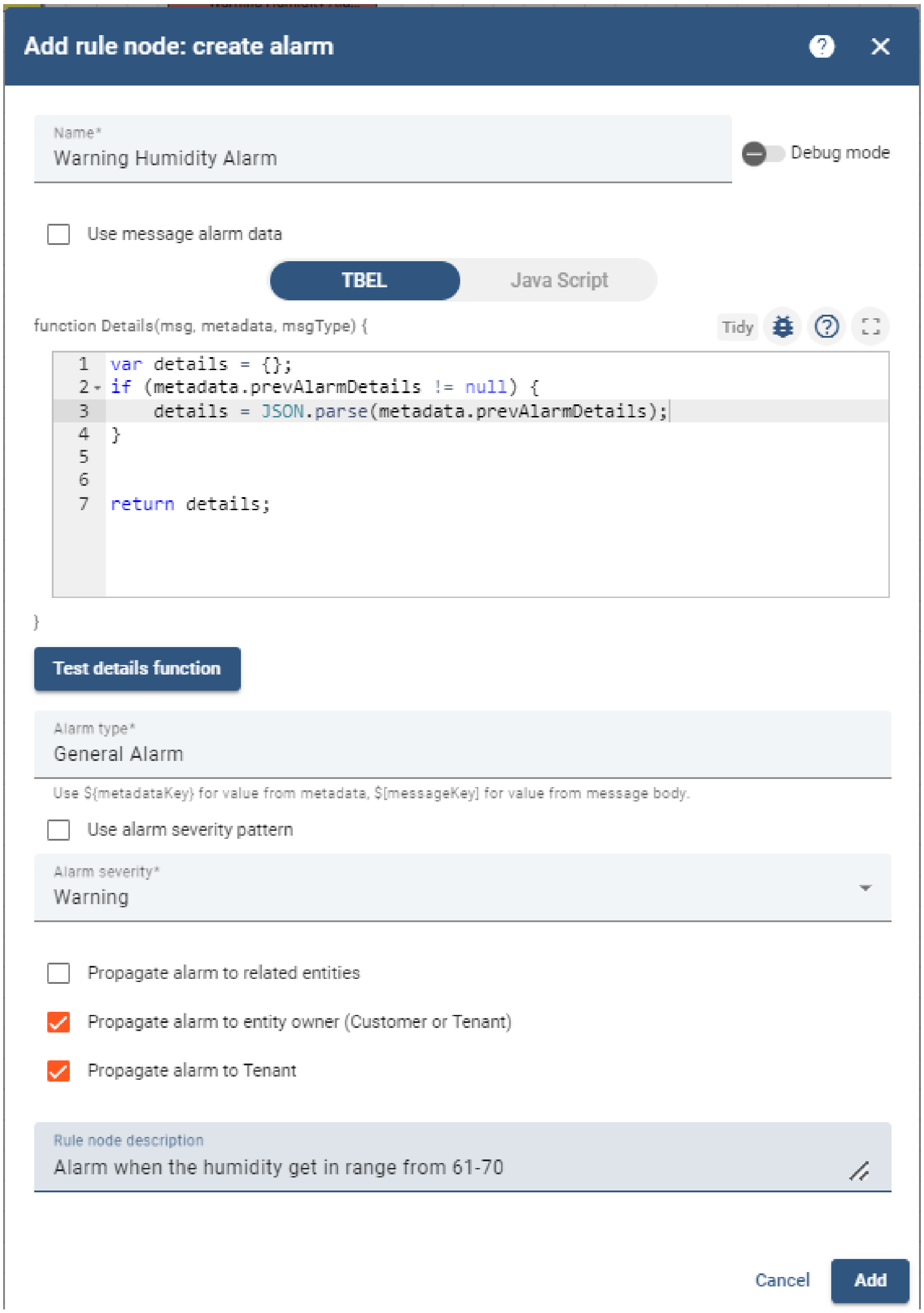
21. Create Rule Chain named Create Alarm



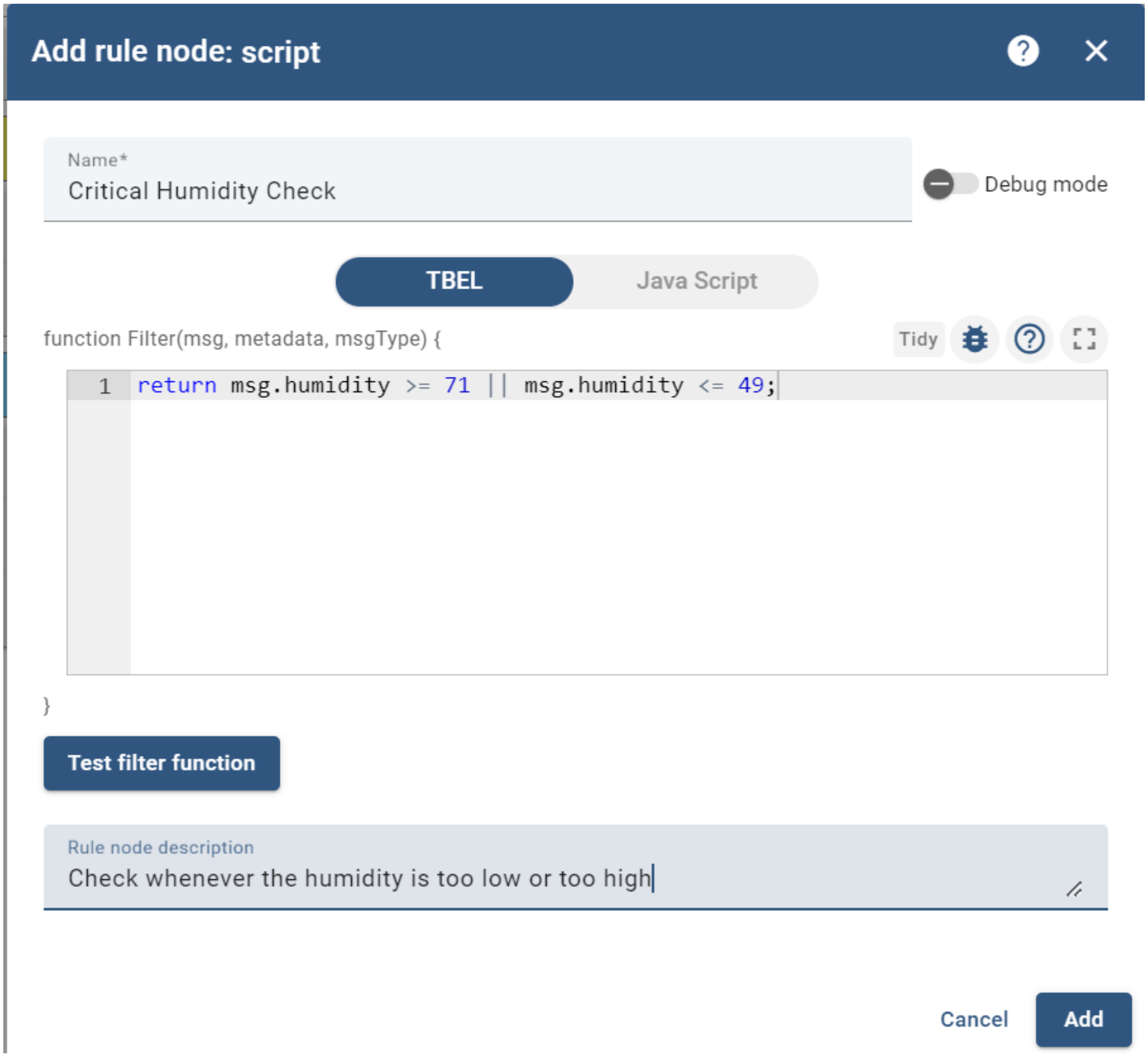
22. Create rule node: script for checking the Humidity Warning range



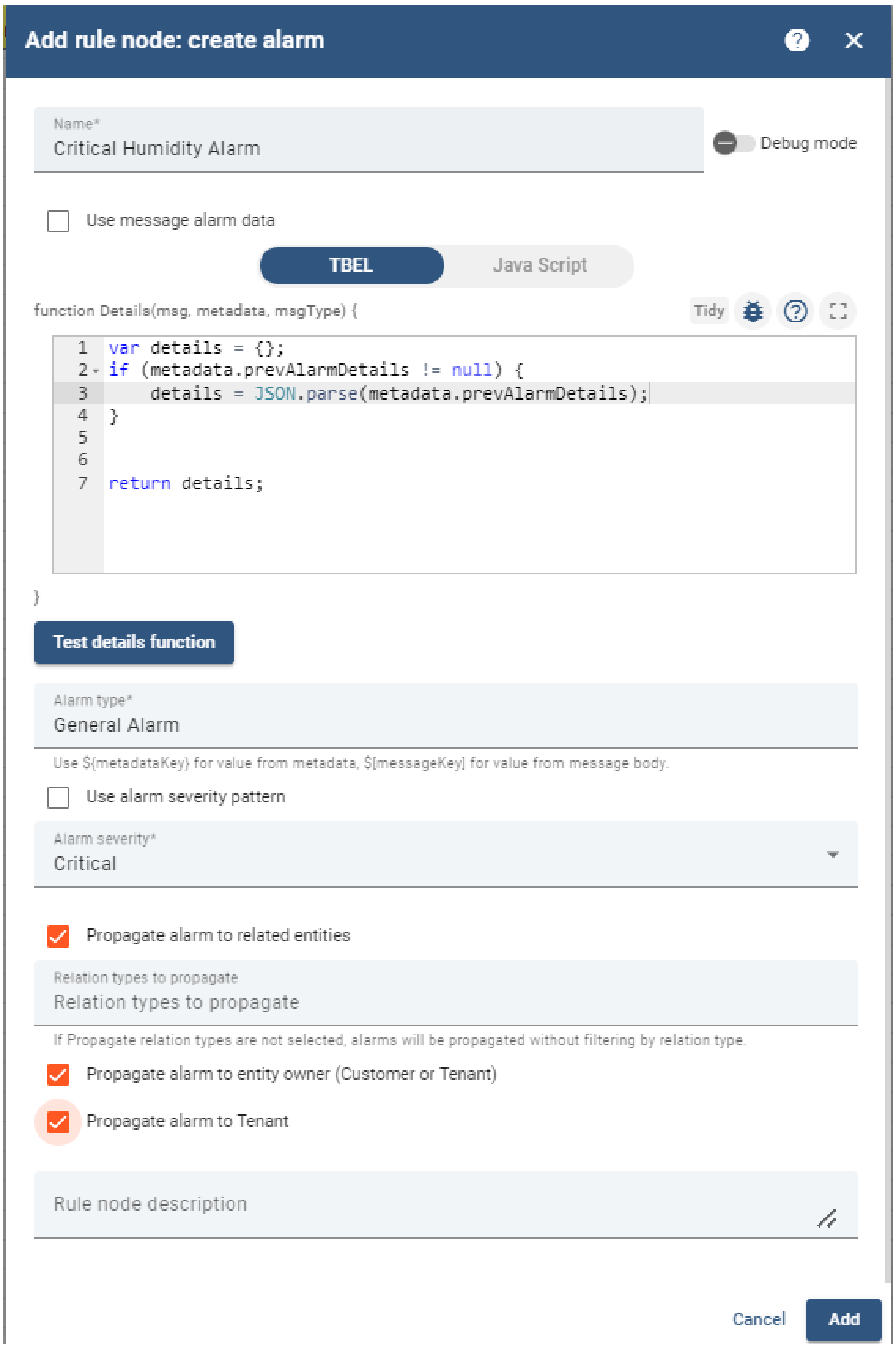
23. Create alarm for warning when humidity get in range warning



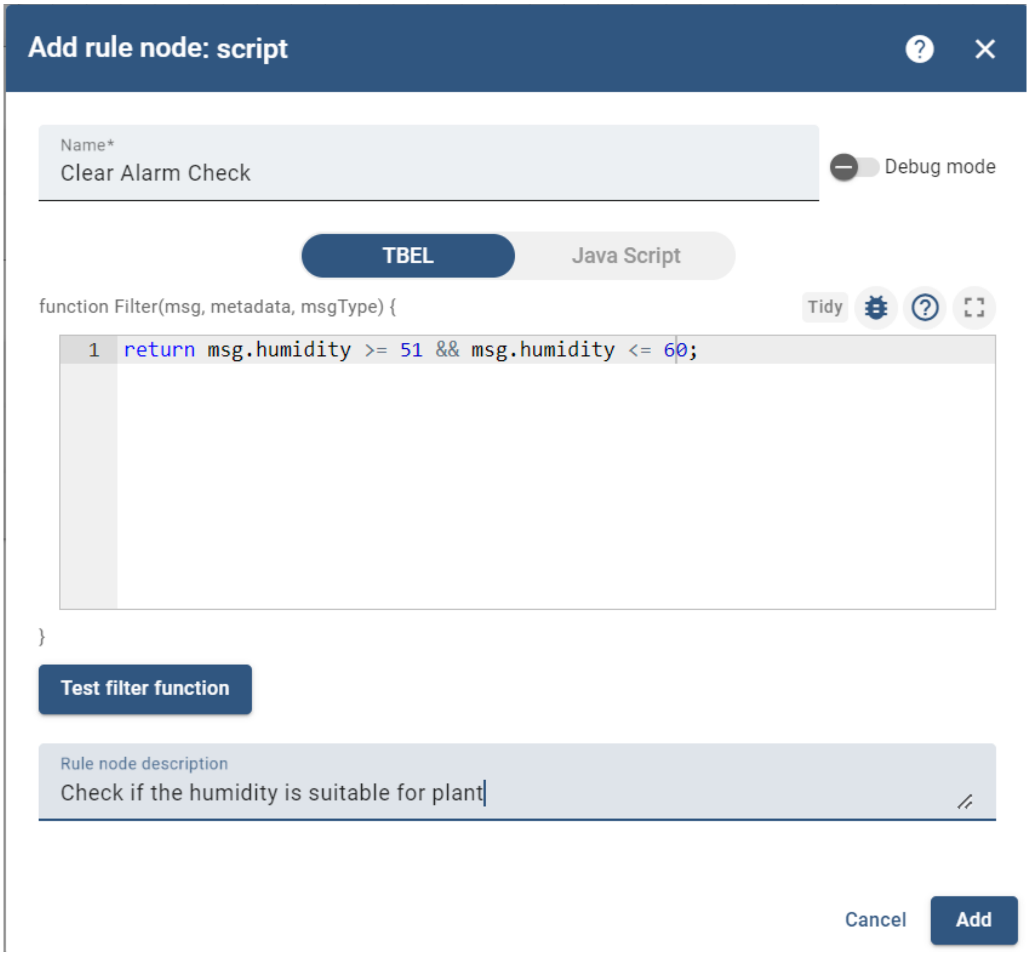
24. Critical Humidity Check node



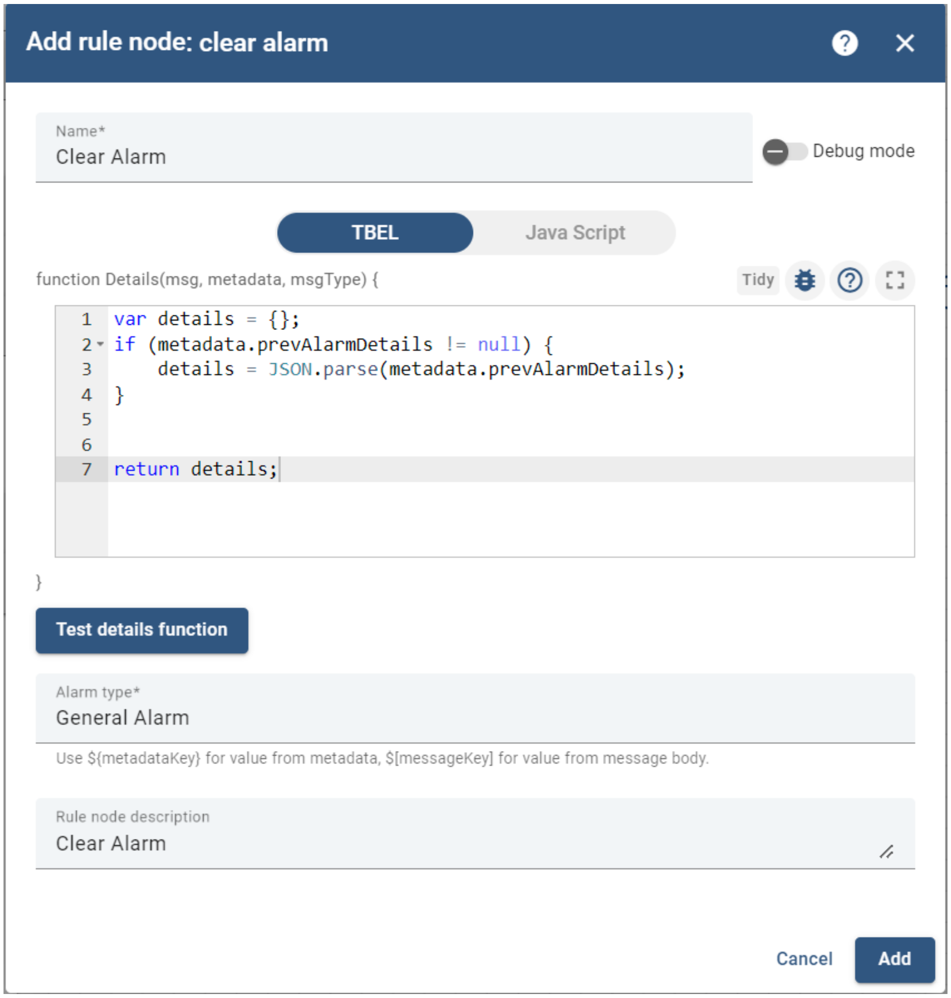
25. Adding the Critical Humidity Alarm



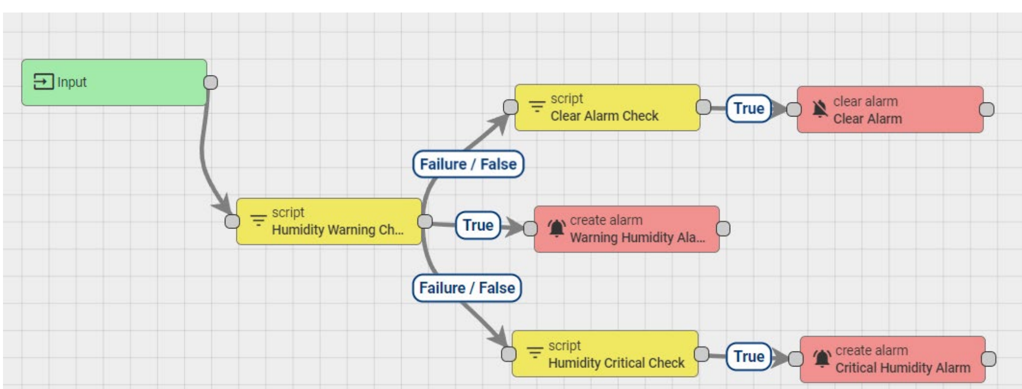
26. Creating node for checking the humidity is suitable for plants



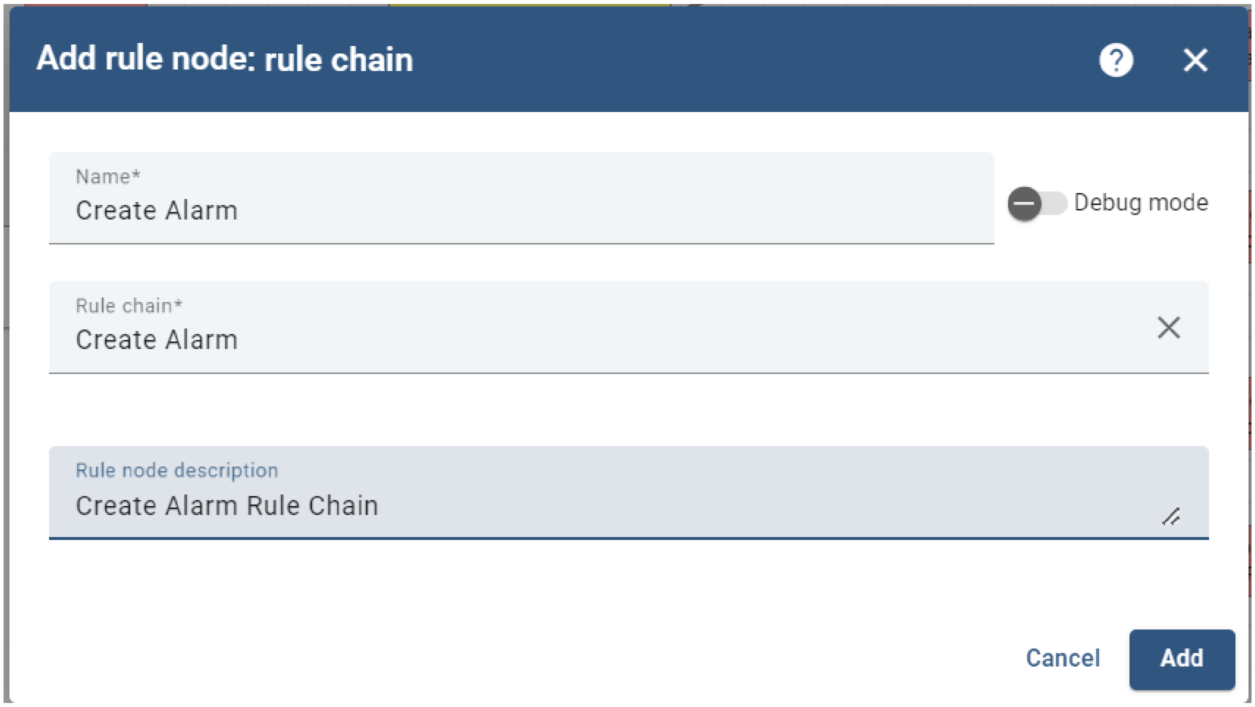
27. Adding the clear alarm node



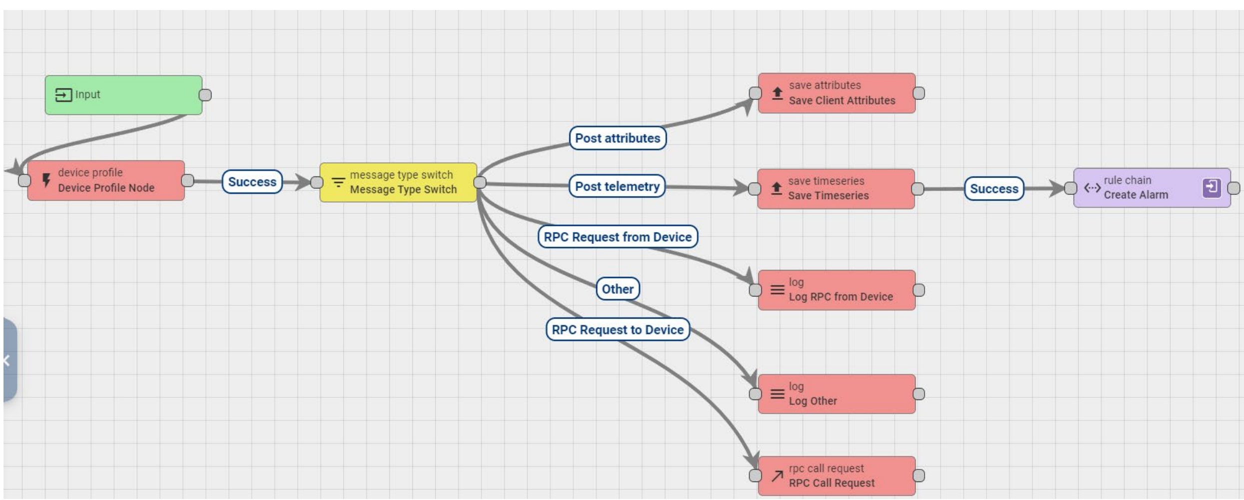
28. Create Alarm Rule Chain



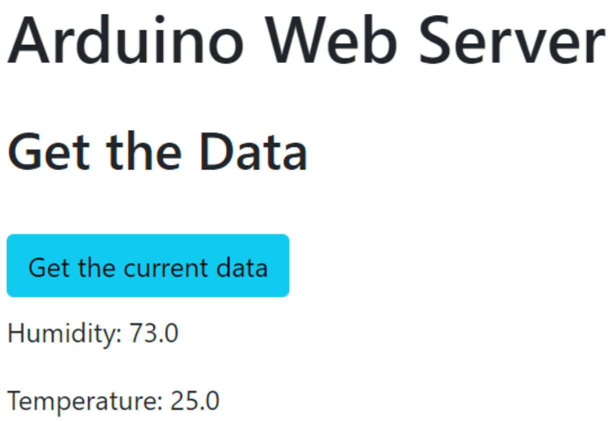
29. Adding the Creating Alarm Rule Chain Node to Root Rule Chain



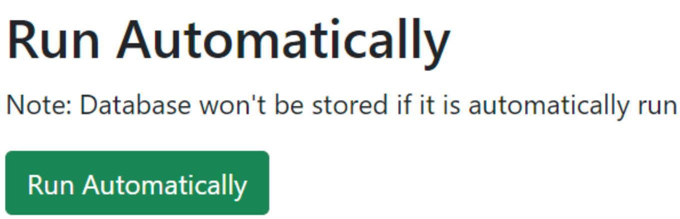
30. Root Rule Chain



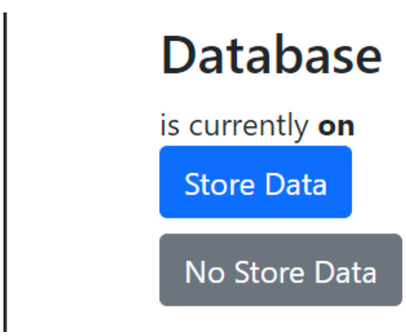
31. Visual for getting data command.



32. Visual for automatically run command



33. Visual for store database command



34. Visual for not storing database command

Database

is currently **off**

Store Data

No Store Data

35. Visual for displaying data from database

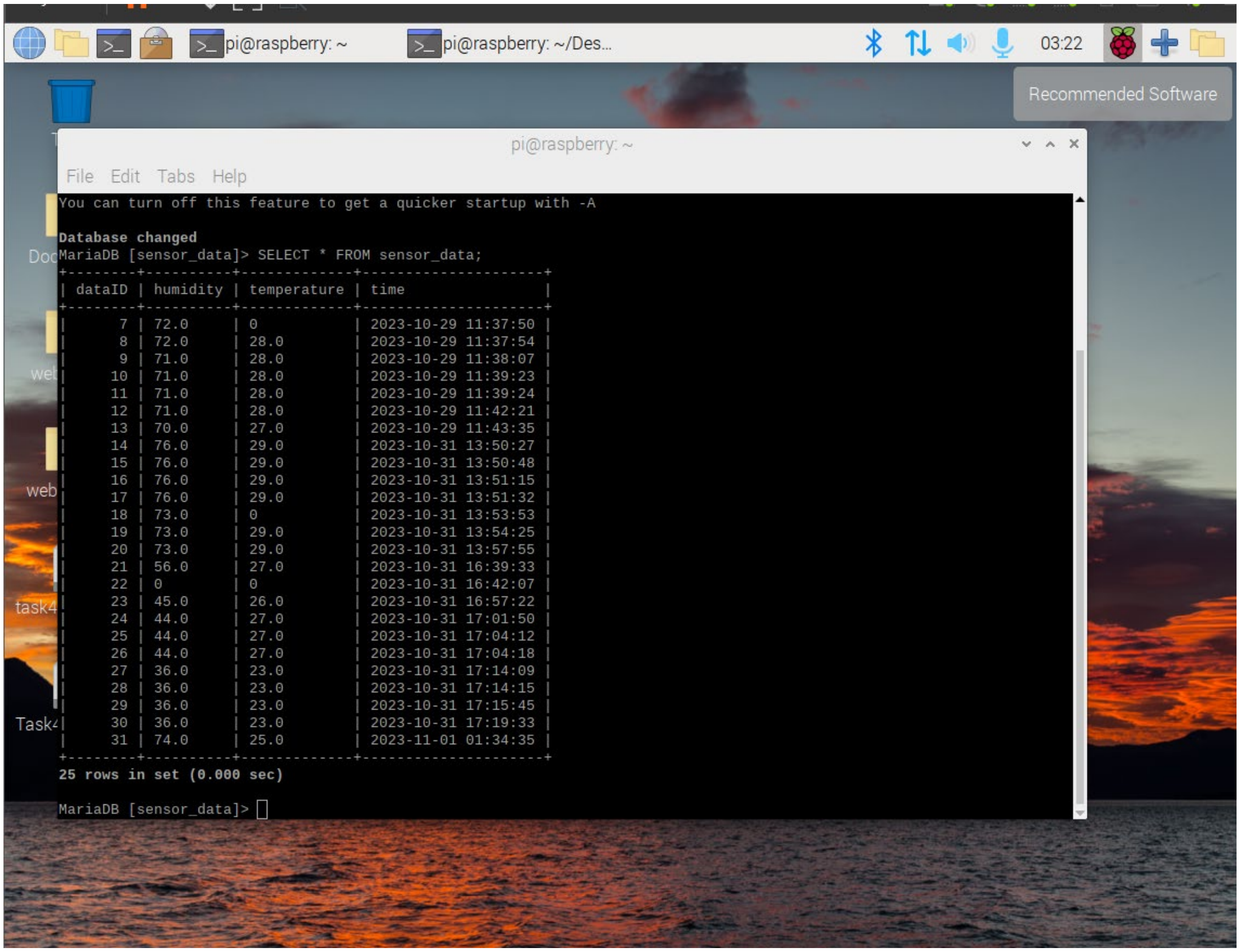
Data from Database

Note: Time may not the same with your local time due to server configuration
Maximum display the last 10 record of data
You need to store the data in the database first so it can get data from database

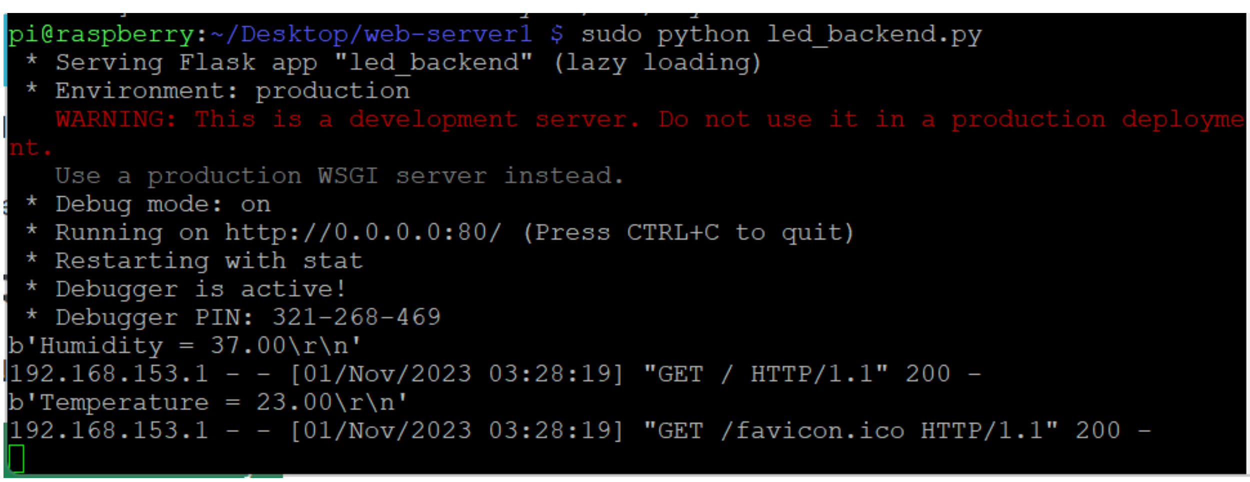
Get data from database

Humidity	Temperature	Date and Time
74.0	25.0	2023-11-01 01:34:35
36.0	23.0	2023-10-31 17:19:33
36.0	23.0	2023-10-31 17:15:45
36.0	23.0	2023-10-31 17:14:15
36.0	23.0	2023-10-31 17:14:09
44.0	27.0	2023-10-31 17:04:18
44.0	27.0	2023-10-31 17:04:12
44.0	27.0	2023-10-31 17:01:50
45.0	26.0	2023-10-31 16:57:22
0	0	2023-10-31 16:42:07

36. Display data from database in edge server



37. Command line to run server



38. C++ code for Arduino

```
#include "dht.h"  //include library
dht DHT;
#define DHT11_PIN 7 //set Pin 7 (LED)
int pin2 = 2; //set Pin 2 (LED)
int pin3 = 3; //set Pin 3 (LED)
int pin4 = 4; //set Pin 4 (LED)
int INA = 9; // set Pin 9(Fan motor)
int INB = 8; //set Pin 8(Fan motor)

int currentMode = 0; // default of mode is 0
void setup() {
  pinMode(pin2, OUTPUT);
  pinMode(pin3, OUTPUT);
  pinMode(pin4, OUTPUT);
  pinMode(INA, OUTPUT);
  pinMode(INB, OUTPUT);
  //Set default of Fan to low
  digitalWrite(INA, LOW);
  digitalWrite(INB, LOW);
  Serial.begin(19200);
}

void loop() {
  int value = Serial.read(); //Read the Serial input
  if (value == '1' && currentMode != 1) { // if the Serial input is 1
and current mode is not 1
    currentMode = 1; // set current mode = 1 (automatically)
  } else if ((value == '2' || value == '3') && currentMode != 2) {
    currentMode = 2; // set the current mode = 2 (For displaying LED
customly)
  } else if (value == '4' && currentMode != 4){
```

```
    currentMode = 4; // set current mode = 2 (For displaying LED
customly and run fan motor)
  }
  int chk = DHT.read11(DHT11_PIN);
  Serial.print("Humidity = ");
  Serial.println(DHT.humidity); //Display humidity from sensor
  Serial.print("Temperature = ");
  Serial.println(DHT.temperature); //Display temperature from sensor
  if (currentMode == 1){ // the code will run automatically if
currentMode is 1
    if (DHT.humidity >= 50 && DHT.humidity <= 60) {
      digitalWrite(pin2, HIGH); //trigger green light
      digitalWrite(pin3, LOW);
      digitalWrite(pin4, LOW);
      digitalWrite(INA, LOW);
      digitalWrite(INB, LOW);
    } else if (DHT.humidity >= 61 && DHT.humidity <= 70) {
      digitalWrite(pin2, LOW);
      digitalWrite(pin3, HIGH); //trigger yellow light
      digitalWrite(pin4, LOW);
      digitalWrite(INA, LOW);
      digitalWrite(INB, LOW);
    } else if (DHT.humidity >= 71 || DHT.humidity <= 49) {
      digitalWrite(pin2, LOW);
      digitalWrite(pin3, LOW);
      digitalWrite(pin4, HIGH); //trigger red light
      //run Fan motor clockwise
      digitalWrite(INA, HIGH);
      digitalWrite(INB, LOW);
    }
  } else {
```

```
    if (value == '2') {
      digitalWrite(pin2, HIGH); //trigger green led
      digitalWrite(pin3, LOW);
      digitalWrite(pin4, LOW);
      digitalWrite(INA, LOW);
      digitalWrite(INB, LOW);
    } else if (value == '3') {
      digitalWrite(pin2, LOW);
      digitalWrite(pin3, HIGH); // trigger yellow led
      digitalWrite(pin4, LOW);
      digitalWrite(INA, LOW);
      digitalWrite(INB, LOW);
    } else if (value == '4') {
      digitalWrite(pin2, LOW);
      digitalWrite(pin3, LOW);
      digitalWrite(pin4, HIGH); //trigger red led
      digitalWrite(INA, HIGH); //trigger fan motor
      digitalWrite(INB, LOW);
    }
  }
  while (currentMode == 4) { // keep the fan run if current mode is 4
    digitalWrite(INA, HIGH);
    digitalWrite(INB, LOW);
  }
  delay(2000); // delay 2s
}
```

39. Python code

```
import serial
import MySQLdb
from flask import Flask, render_template
```

```
import paho.mqtt.publish as publish
from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.triggers.interval import IntervalTrigger

app = Flask(__name__)
topic = "v1/devices/me/telemetry"
device='/dev/ttyUSB0'
device_id = "OwvtaAzFoOYeUU1WSILH"
ser = serial.Serial(device, 9600, timeout=1)  # Establish the
connection on a specific port
hostname = "192.168.153.131"
port = 1884
sensors = { # Dictionary with the sensors
    1 : {'name' : 'Humidity', 'state' : 0 },
    2 : {'name' : 'Temperature', 'state' : 0 },
}
database = { # Dictionary with the database
    1 : {'name' : 'Database', 'state' : 0 },
    2 : [],
}
def read_sensor_data(): # Update the sensor data
    global sensors # Access the global variable sensors
    global database # Access the global variable database
    sensor = ser.readline() # Read the data from the Arduino
    sensor_str = sensor.decode('utf-8')  # Decode the bytes to a
string
    for line_sensor in sensor_str.split('\n'): # Split the string into
lines
        if line_sensor.startswith('Humidity'): # Check if the line
starts with Humidity
```

```
            sensors[1]['state'] =
float(line_sensor.split('=')[1].strip()) # Get the value after the =
        elif line_sensor.startswith('Temperature'): # Check if the
line starts with Temperature
            sensors[2]['state'] =
float(line_sensor.split('=')[1].strip()) # Get the value after the =
def connect_to_database():
    try:
        return MySQLdb.connect("localhost", "pi", "", "sensor_data")
    except Exception as e:
        ser.write(b"4")  # write serial input for Arduino code to
trigger red light
        print(f"Error connecting to the database: {e}")
        return None
def send_to_cloud():
    try:
        # mosquitto_pub -d -q 1 -h 192.168.153.131 -p 1884 -t
v1/devices/me/telemetry -u OwvtaAzFoOYeUU1WSILH -m
"{temperature:29,humidity:50}"
        # Use subprocess to run mosquitto_pub command
        temperature = sensors[2]['state']
        humidity = sensors[1]['state']
        publish.single(topic,
f'{{"temperature":{temperature},"humidity":{humidity}}}',
hostname=hostname, port=port, qos=1, auth={'username': device_id})
    except Exception as e:
        ser.write(b"4")  # write serial input for Arduino code to
trigger red light
        print(f"Error sending data to the cloud: {e}")
def print_sensor_data():
    print(f"Humidity: {sensors[1]['state']}")
```

```
    print(f"Temperature: {sensors[2]['state']}")
def update_sensor_data_and_send_to_cloud():
    read_sensor_data()
    send_to_cloud()

# Main function when accessing the website
@app.route("/") # This is the main page
def index(): # This function will be executed when the main page is
accessed
    read_sensor_data()
    templateData = { 'sensors' : sensors, 'database' :database } #
Create a dictionary with the data to be sent
    return render_template('index.html', **templateData) # Return the
template

# Function with buttons to toggle to store the data into the database
or not
@app.route("/<toggleDatabase>")
def toggle_store_data(toggleDatabase): # This function will be
executed when the main page is accessed
    update_sensor_data_and_send_to_cloud()  # Update and send data
immediately
    dbConn = None
    try:
        dbConn = connect_to_database() # Connect to the database
        if dbConn:
            cursor = dbConn.cursor() # Create a cursor
            if toggleDatabase == "storeData": # Check if the button is
storeData
                database[1]['state'] = 1 # Turn on the database
```

```
                ser.write(b"2") # write serial input for Arduino code
to trigger green light
                # Insert the data into the database
                cursor.execute("INSERT INTO sensor_data (humidity,
temperature) VALUES (%s, %s)",
                               (str(sensors[1]['state']),
str(sensors[2]['state'])))
                dbConn.commit() # Commit the changes
                print("Data stored in the database") # Print a message
                cursor.execute("SELECT * FROM sensor_data ORDER BY
dataID DESC LIMIT 10") # Get the last 10 data
                rows = cursor.fetchall() # Fetch the rows
                for row in rows: # Loop through the rows
                    database[2].append({'humidity': row[1],
'temperature': row[2], 'time_stamp': row[3]}) # Append the data to the
database
            elif toggleDatabase == "noStoreData": # Check if the
button is noStoreData
                database[1]['state'] = 0 # Turn off the database
                ser.write(b"3") # write serial input for Arduino code
to trigger red light

            elif toggleDatabase == "getDatabase":  # Check if the
button is getDatabase
                cursor.execute("SELECT * FROM sensor_data ORDER BY
dataID DESC LIMIT 10") # Get the last 10 data
                rows = cursor.fetchall()
                for row in rows:
                    database[2].append({'humidity': row[1],
'temperature': row[2], 'time_stamp': row[3]})
    except paho.mqtt.MQTTException as mqtt_error:
```

```
        ser.write(b"4")  # write serial input for Arduino code to
trigger red light
        print(f"MQTT Exception: {mqtt_error}")
        # Handle the MQTT connection error here (e.g., log, send
notification, etc.)

    finally:
        while len(database[2]) > 10:  # Check if the database is
greater than 10
            if database[2]:
                try:
                    database[2].pop(0)  # Remove the first element
                except IndexError:
                    break  # Break out of the loop if the list is
empty or index is out of range
            else:
                break  # Break out of the loop if the list is empty
        if dbConn: # Close the database
            dbConn.close()
    templateData = { 'sensors' : sensors, 'database' : database } #
Create a dictionary with the data to be sent
    return render_template('index.html', **templateData) # Return the
template
@app.route("/automatically") #URL/automatically
def automatic():
    scheduler = BackgroundScheduler()
    scheduler.add_job(update_sensor_data_and_send_to_cloud,
trigger=IntervalTrigger(seconds=2))
    scheduler.add_job(print_sensor_data,
trigger=IntervalTrigger(seconds=2))
    scheduler.start()
```

```
    ser.write(b"1") #write serial input for Arduino code to trigger
automatic code
    templateData = { 'sensors' : sensors, 'database' : database }
    return render_template('index.html', **templateData)
#get data directly from arduino and send to the website
@app.route("/getData") #URL/getData
def getData():
    read_sensor_data()
    templateData = { 'sensors' : sensors, 'database' : database } #
Create a dictionary with the data to be sent
    return render_template('index.html', **templateData) # Return the
template
# Main function when accessing the website
if __name__ == '__main__':
    ser = serial.Serial(device, 9600, timeout = 1) # Establish the
connection on a specific port
    ser.flush() # Clear the serial buffer
    app.run(host='0.0.0.0', port = 80, debug = True) # Run the app
```

40. HTML code

```
<!DOCTYPE html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1">
        <!-- Bootstrap CSS -->
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.
min.css" rel="stylesheet">
```

```
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bu
ndle.min.js"></script>
        <meta name="author" content="Thanh Minh" />
        <meta name="description" content="SWE30011 - IOT Programming"
/>

        <title> Arduino Web Server </title>
        <style>
            table, th, td {
                border:1px solid black;
            }
        </style>
    </head>
    <body class="container">
        <h1 class="text-center"> Arduino Web Server </h1>
        <p>
            <h2 class="my-4">
                Get the Data
            </h2>
            <!-- Display the buttons to get the data -->
            <a href="/getData" class="btn btn-info my-2">Get the
current data</a>
            {% for sensor in sensors %} <!-- Loop through the sensors
dictionary -->
                <p>{{sensors[sensor]['name']}}:
{{sensors[sensor]['state']}}</p>
            {% endfor %}
        </p>
        <p>
            <h2>Run Automatically</h2>
```

```
            <p>
                Note: Database won't be stored if it is automatically
run
            </p>
            <a href="/automatically" class="btn btn-success">Run
Automatically</a> <!-- Display the button to run automatically -->
        </p>
        <p>
            <!-- Display the buttons to toggle the database -->
            <h3>{{database[1]['name']}}</h3>
            {% if database[1]['state'] == 1 %} <!-- Check if the
database is on or off -->
                is currently <strong>on</strong></h2>
            {% else %}
                is currently <strong>off</strong></h2>
            {% endif %}
            <br />
            <a href="/storeData" class="btn btn-primary mb-2">Store
Data</a> <!-- Display the button to store data -->
            <br />
            <a href="/noStoreData" class="btn btn-secondary">No Store
Data</a> <!-- Display the button to no store data -->
        </p>
        <p>
            <h2>Data from Database</h2>
            <p>
                Note: Time may not the same with your local time due
to server configuration
                <br />
                Maximum display the last 10 record of data
                <br />
```

```
                You need to store the data in the database first so it
can get data from database
            </p>
            <a href="/getDatabase" class="btn btn-dark mb-4">Get data
from database</a> <!-- Display the button to get data from database --
>
            <table class="table table-striped table-bordered">
                <th class="text-center" scope="col">Humidity</th>
                <th class="text-center" scope="col">Temperature</th>
                <th class="text-center" scope="col">Date and Time</th>
                {% for entry in database[2] %} <!-- Loop through the
database dictionary -->
                <tr>
                    <td class="text-center">{{ entry['humidity']
}}</td>
                    <td class="text-center">{{ entry['temperature']
}}</td>
                    <td class="text-center">{{ entry['time_stamp']
}}</td>
                </tr>
                {% endfor %}
            </table>

        </p>

        <p>
            <h2>All Commands</h2> <!-- Display all the buttons -->
            <ol>
                <li class="my-1"><a href="/getData" class="btn btn-
info">Get the current data</a></li>
```

```
                <li class="my-1"><a href="/automatically" class="btn
btn-success">Run Automatically</a></li>
                <li class="my-1"><a href="/storeData" class="btn btn-
primary">Store Data</a></li>
                <li class="my-1"><a href="/noStoreData" class="btn
btn-secondary">No Store Data</a></li>
                <li class="my-1"><a href="/getDatabase" class="btn
btn-dark mb-4">Get data from database</a></li>
            </ol>
        </p>
    </body>
</html>
```