Advanced Computing Lab

Course:	Internet of Things (Lab)
https://aclab-hcmut.github.io/	

IoT Lab 2: Core-IoT Platform

Student name: Nguyễn Thanh Hiền
Student ID: 2111203
Class: L01
GitHub repository: https://github.com/amyranotamirror/CO3038-iot/tree/lab-2

1. Overview

1.1. Introduction

This lab introduces students to Core IoT, a platform used for device management, data visualization, and real-time analytics. By the end of this lab, students will understand how to connect IoT devices to ThingsBoard, send and receive telemetry data, and create interactive dashboards for monitoring IoT systems.

1.2. Requirements

- Create a project with PlatformIDE in VSCode and integrate the ESP32 board into the project (you can clone available projects at: https://github.com/ACLAB-HCMUT/PlatformIO_Arduino-Framework_ESP32_Template
- Get the temperature and humidity data from DHT20 and display this data in
- CoreIOT server with template dashboard.
- Complete all tasks in this lab with the RTOS standard.
- Create a scheduler

2. System design

2.1. System Architecture

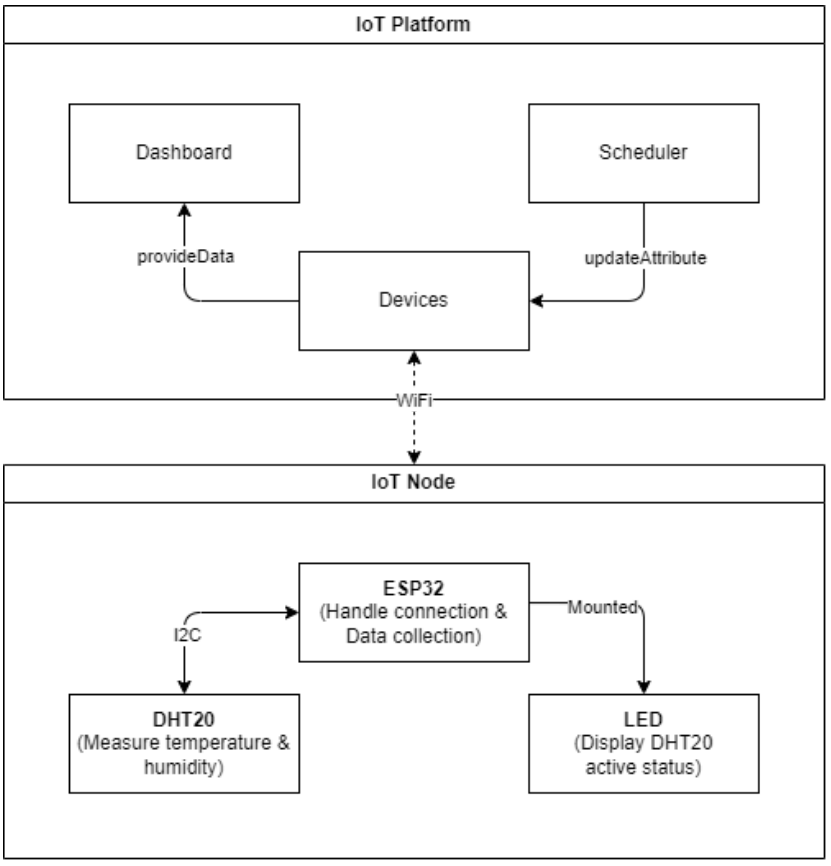

Figure 1: System Architecture

This IoT system follows a three-layer architecture:

1. **Perception Layer** (Data collection)
    - DHT20 sensors periodically collect temperature & humidity data.
    - LED indicates DHT20 sensor active status.
    - ESP32 reads data and processes it before transmitting.
2. **Network Layer** (Data transmission)
    - ESP32 sends data to the IoT platform over WiFi (e.g., via MQTT).
    - Uses shared attributes to sync sensor state (e.g., LED status) between device and platform.
3. **Application Layer** (Dashboard)
    - The dashboard displays real-time sensor data.
    - Scheduler updates device settings using shared attributes via the Devices module.

2.2. Hardware components

- Temperature & Humidity Sensor: **DHT20** [1]
  - Input voltage: 3.3VDC
  - Humidity range: 0 ~ 100% RH
  - Humidity accuracy: ± 3% RH (25 ℃)
  - Temperature range: -40 ~ + 80 ℃
  - Temperature accuracy: ± 0,5 ℃
- Microcontroller board: **Kit Wifi BLE SoC ESP32 WeMos D1 R32** [2]
  - Power supply:
  - 5VDC via USB port
  - 5~12V DC via round DC power jack or Vin pin.
  - Central Processing Unit: Wireless Module – ESP-WROOM-32, based on Espressif ESP32 dual-core Tensilica LX6 processor with 802.11 b/g/n WiFi and Bluetooth 4.2 LE.
  - Pin Configuration & Compatibility: Full ESP32 pinout, designed with GPIO pin headers and dimensions similar to Arduino Uno.
  - Integrated Components: Built-in CH340 UART communication and programming circuit.
- Extended shield: **Arduino UNO** [3]
  - Port: 4 Analog, 4 I2C, 4 Digital
  - 11 servo channel
  - Switch between 3.3V to 5V

2.3. Dashboard design

With the dashboard, the writer requires features to monitor temperature and humidity

- Display historical data of temperature and humidity:
  - Range: Last 5 minutes
  - Aggregation: 10 seconds
- Display the current data of temperature and humidity
- Create / Edit / Delete scheduled events
  - Turn on the LED & sensor every 2 minutes
  - Turn off the LED & sensor every 2 minutes
- Display the location of sensors

---

[1] Ohstem. 1. Cảm biến nhiệt độ độ ẩm DHT20. Access via: https://docs.ohstem.vn/en/latest/module/cam-bien/dht20.html (Access date: March 20th, 2025).

[2] HSHOP. Kit Wifi BLE SoC ESP32 WeMos D1 R32 (Arduino Compatible). Access via: https://hshop.vn/kit-arduino-wifi-ble-soc-esp32-wemos-d1-r32 (Access date: March 20th, 2025).

[3] Ohstem. Mạch mở rộng cho Arduino UNO. https://ohstem.vn/product/mach-mo-rong-cho-arduino/ (Access date: March 20th, 2025).

- Devices: CS1 - DHT20 sensor, CS2 - DHT20 sensor
- Server data: Name, Latitude, Longitude
- Telemetry: Temperature, Humidity

2.4. Data Collection & Transmission

2.4.1. Data Collection

- Measurement frequency: Every 5 seconds (both temperature and humidity)
- Serial print format: “Data read: ${temperature} °C | ${humidity}”.

2.4.2. Data Transmission

- Protocol: MQTT
- Payload format: JSON
- Transmission:
  - Frequency: Every 10 seconds (both temperature and humidity)
  - Serial print: “Data sent: ${temperature} °C | ${humidity}”
- Connection monitoring:
  - Check WiFi and CoreIoT connection every 1 second
  - Automatically reconnects if disconnection is detected.

3. Implementation

GitHub repository: https://github.com/amyranotamirror/CO3038-iot/tree/lab-2

3.1. Initial setup

Requirements:

- Download VSCode and install the PlatformIO plugin.
- Build the project with the available source code at the GitHub link and upload the code to the ESP32-S3 board.

Implementation:

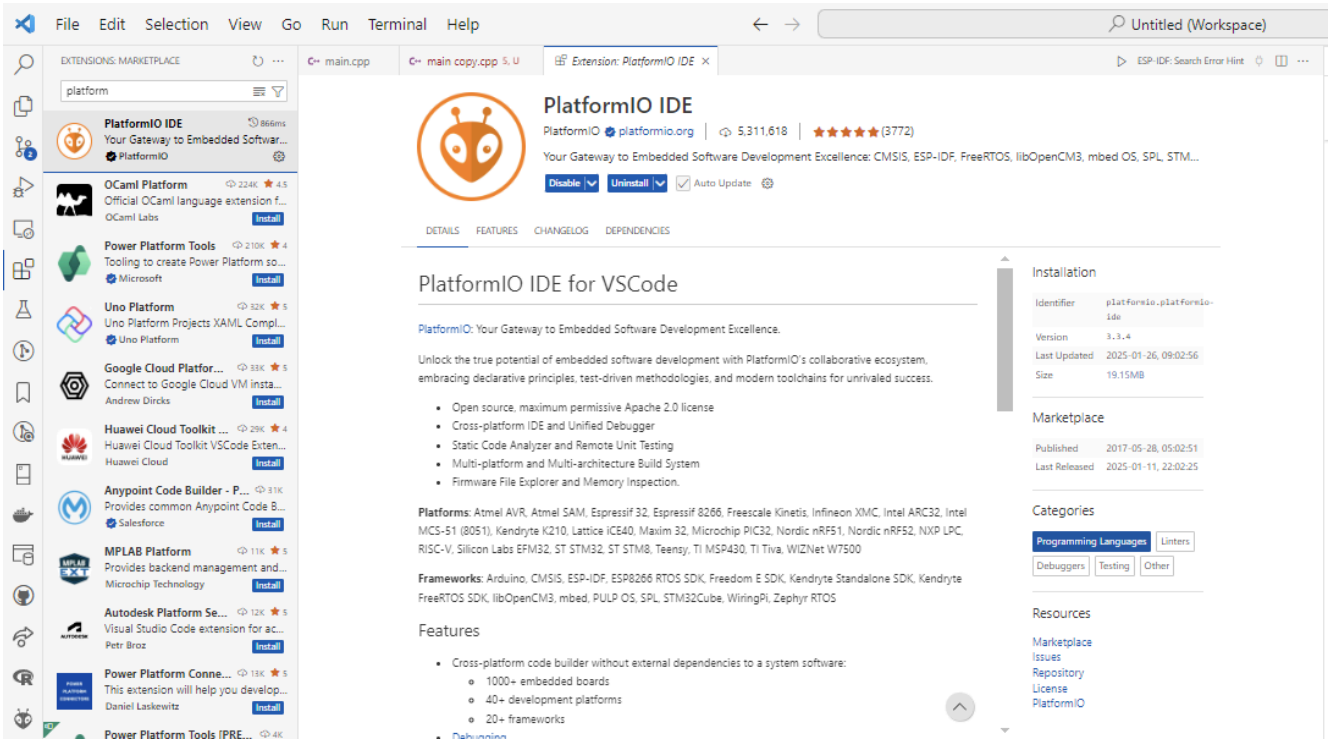

Figure 2. Download PlatformIO on VSCode

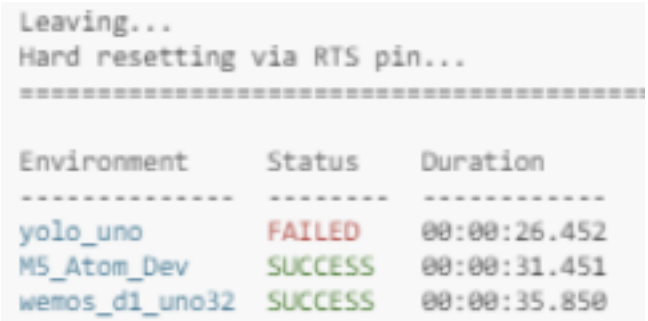

Figure 3. Build project

3.2. Get data from DHT20

Requirements:

- Connect DHT20 to the ESP32-S3 board via the I2C port.
- Measure temperature and humidity data from DHT20 every 5 seconds.

Implementation:

Figure 4. Get data from DHT20 every 5 seconds

3.3. Display data in the CoreIOT server

Requirements

- Create a dashboard in the CoreIOT platform.
- Display this data in the dashboard.
- Implement the Scheduler, follow this doc: https://thingsboard.io/docs/pe/user-guide/scheduler/

Implementation:

3.3.1. Create devices and groups

Device name	Server attribute	Shared attribute
CS1 - DHT20	- **latitude** (double): 10.772338483703717 - **longitude** (double): 106.65788033671267	- ledState (bool)
CS2 - DHT20	- **latitude** (double): 10.880832424162568 - **longitude** (double): 106.805321912137169	- ledState (bool)

Table 1. Configure devices and groups

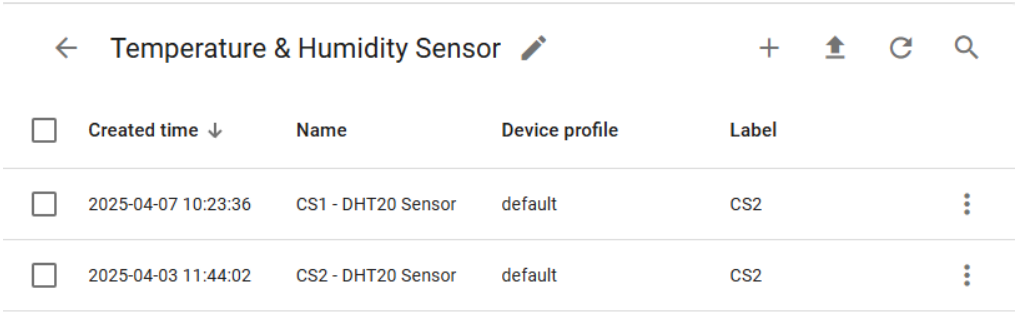

Figure 5. Configure devices and groups on CoreIoT

3.3.2. Create a dashboard

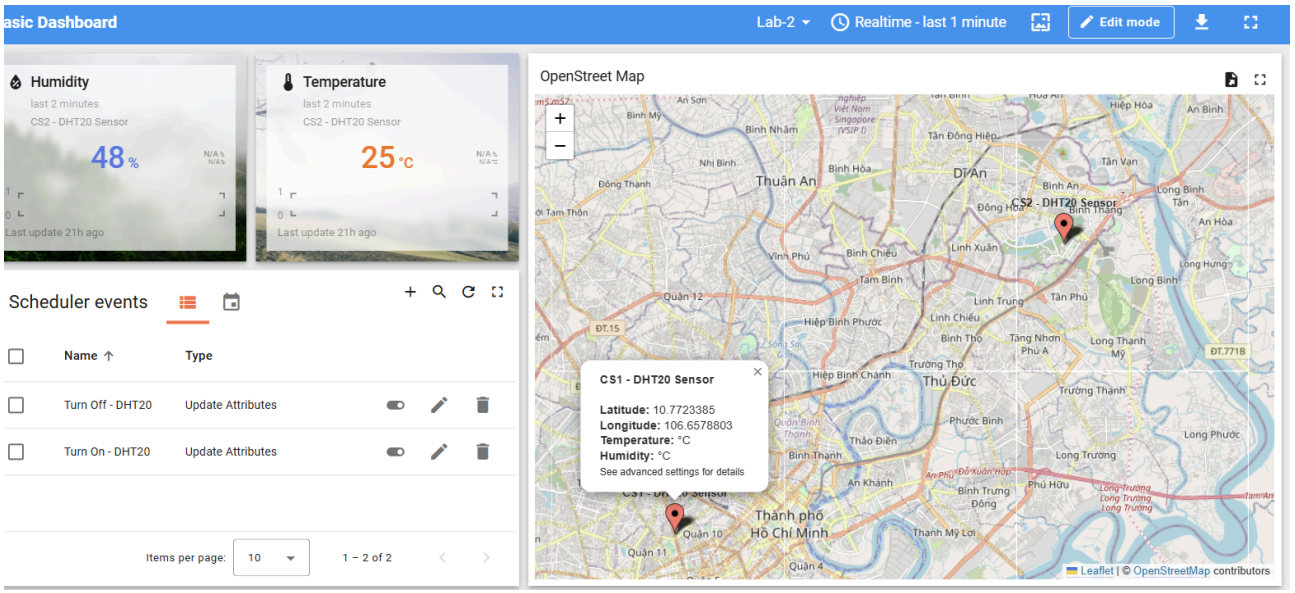

Figure 6. Create dashboard

3.3.3. Create schedules to turn on/off the sensor & LED

Name	Type	Attribute	Target	Schedule
Turn On - DHT20	Update attribute	Shared attribute: {“ledState”: true}	Group entities: Device - Temperature & Humidity sensor	- Start time: 06/04/2025, 11:31 - Repeat: Every 2 minutes
Turn Off - DHT20	Update attribute	Shared attribute: {“ledState”: false}	Group entities: Device - Temperature & Humidity sensor	- Start time: 06/04/2025, 11:31 - Repeat: Every 2 minutes

Table 2. Configure schedulers

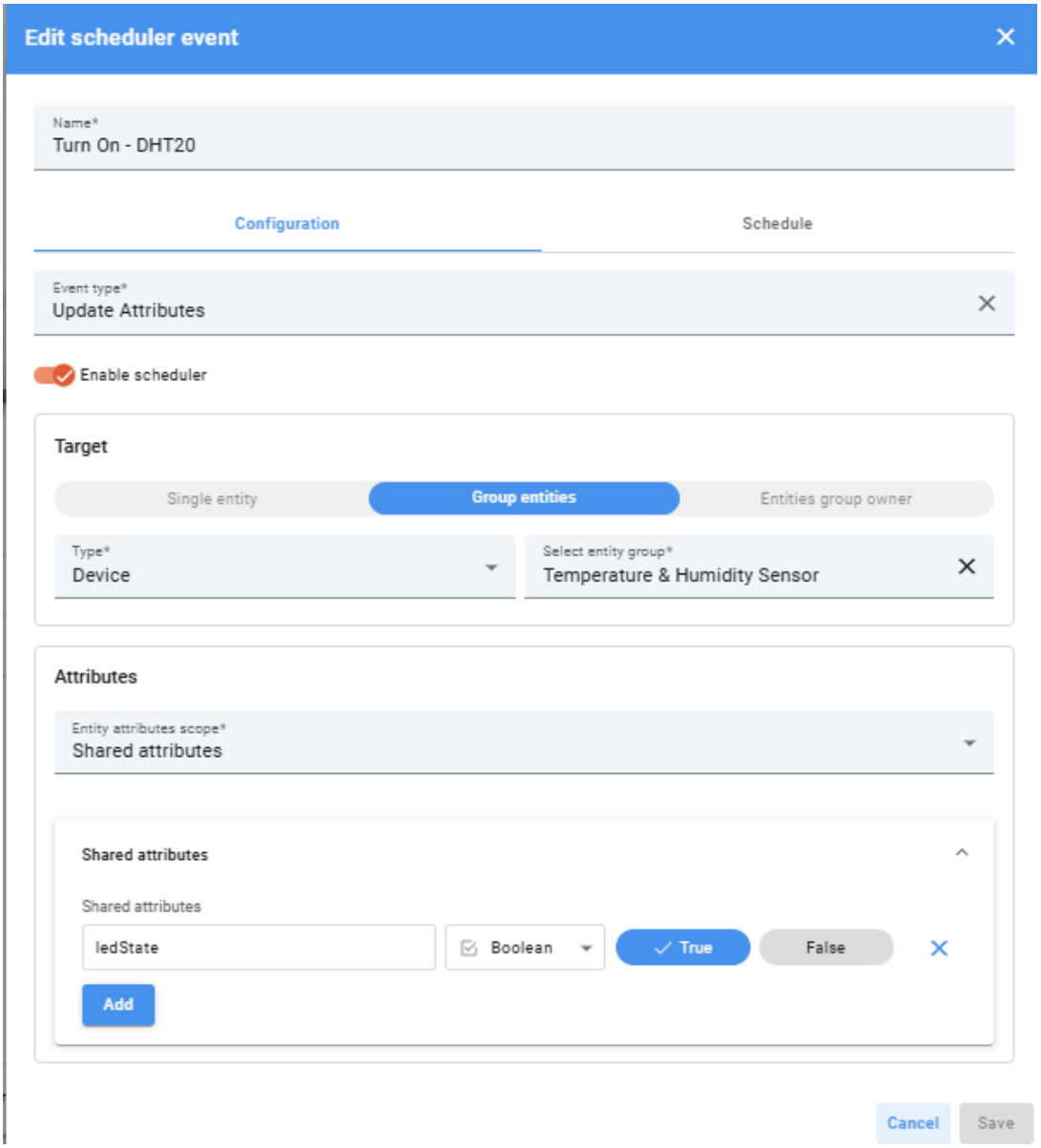

Figure 7. "Turn On - DHT20" configuration

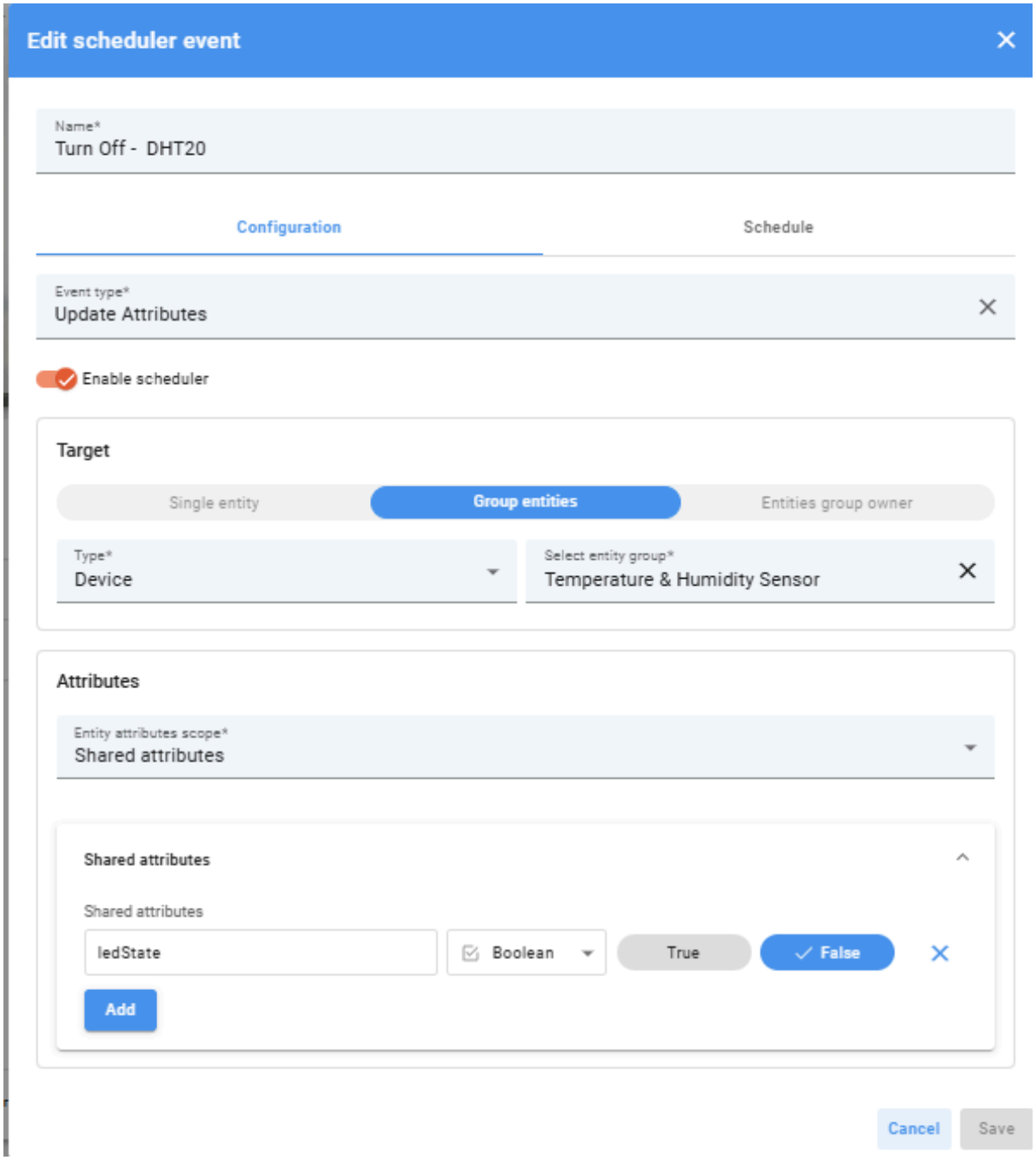

Figure 8. "Turn Off - DHT20" configuration

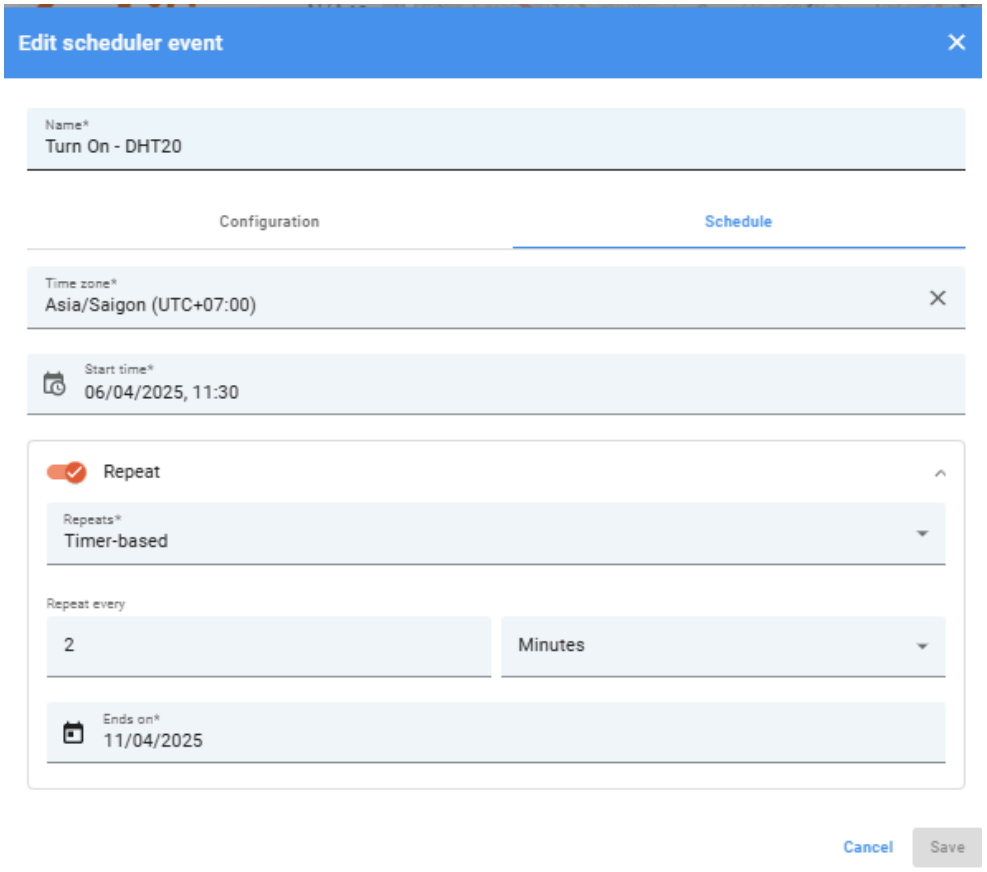

Figure 9. "Turn On - DHT20" schedule

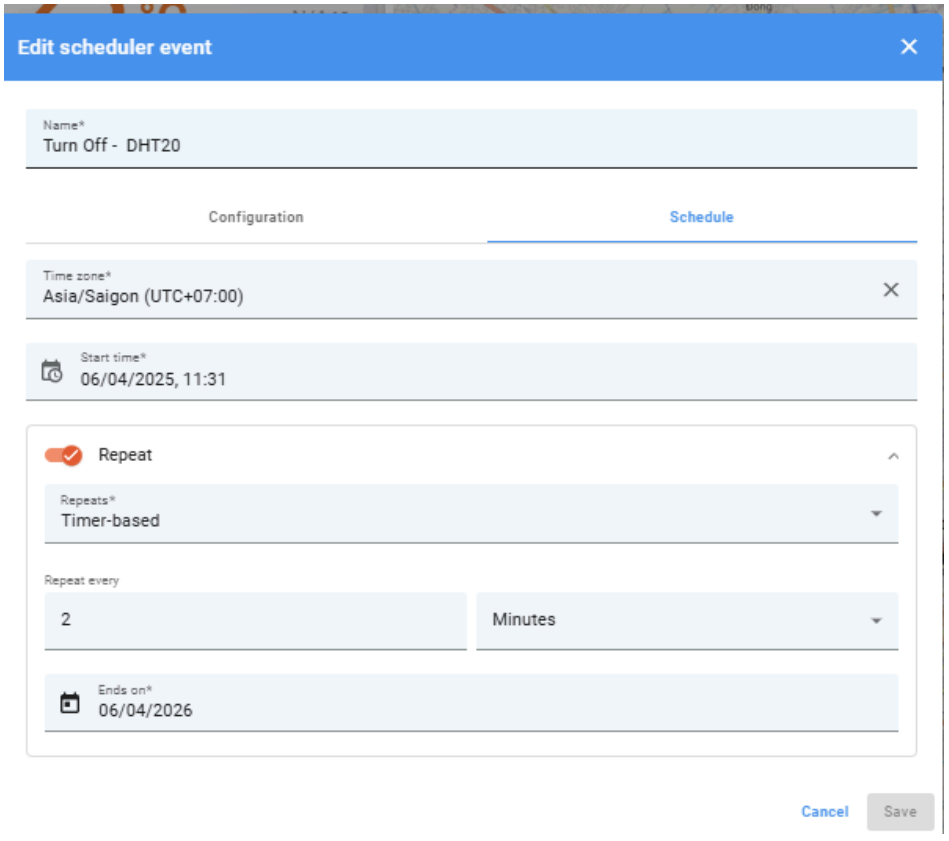

Figure 10. "Turn Off - DHT20" schedule

3.3.4. Validate result

The writer enables the scheduler that:

- Call "Turn On" sensor: Every 2 minutes
- Call "Turn Off" sensor: Every 2 minutes

This means that:

- The sensor reads 13 times and sends telemetry 6 times in 1 minute.
- The sensor is disabled for the following 1 minute.

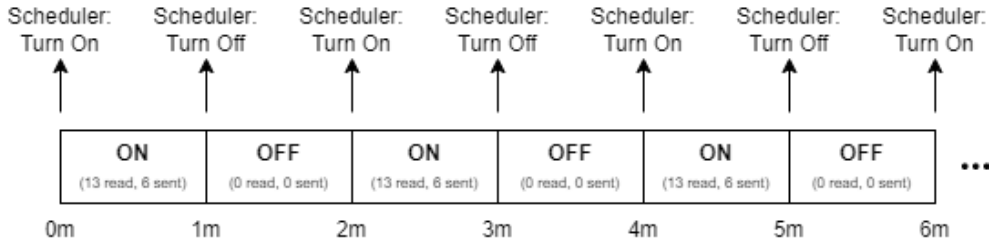

Figure 11. Scheduler timeline

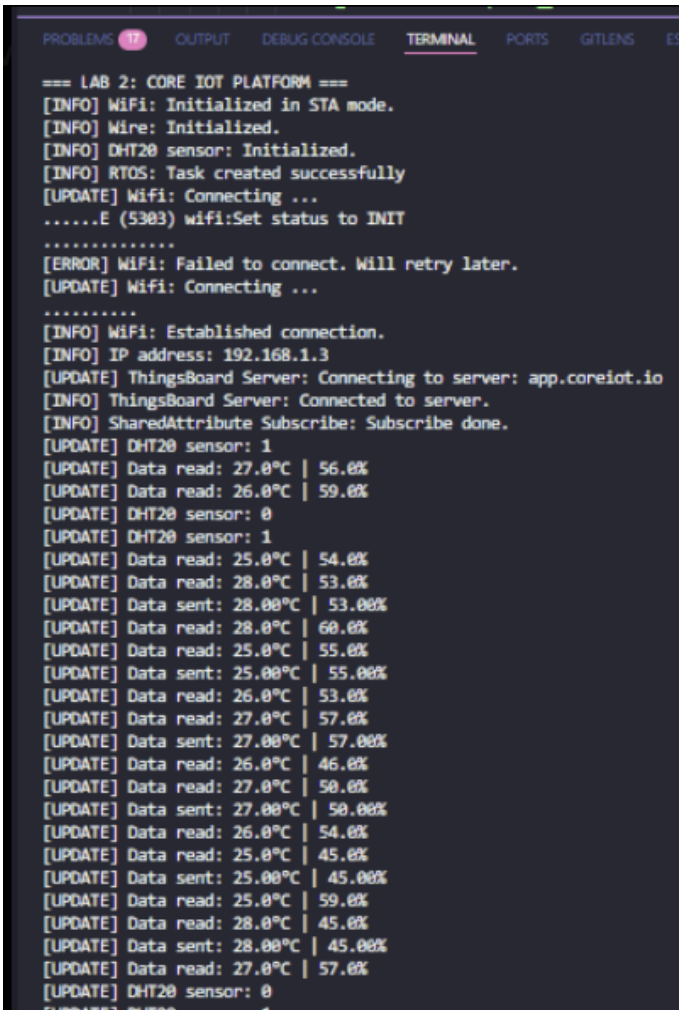

Figure 12. Result

4. Questions

4.1. What protocols can connect devices to ThingsBoard, and what are their advantages?

MQTT (Message Queuing Telemetry Transport)

- **Overview:** MQTT is a lightweight publish-subscribe messaging protocol
- **Advantages:**
  - Lightweight and Efficient: Require minimal resources and are optimized for network bandwidth.
  - Scale to Millions of Things: Can scale to connect with millions of IoT devices.
  - Support for Unreliable Networks: Persistent sessions reduce the time to reconnect the client with the broker.
  - Bi-directional Communications: Allows for messaging between the device and to cloud and the cloud to the device.
  - Reliable Message Delivery: Has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once.
  - Security Enabled: Encrypts messages using TLS and authenticates clients using modern authentication protocols, such as OAuth.

CoAp (Constrained Application Protocol)

- **Best for:** Battery-powered or constrained devices over unreliable networks
- **Advantages:**
  - Uses UDP (lighter than TCP) → great for power-sensitive applications
  - Extremely low overhead
  - RESTful interaction model (GET, POST, PUT, DELETE)
- **Disadvantages:**
  - Not reliable by default — needs retries for packet loss (UDP doesn't guarantee delivery)
  - Limited tooling and support compared to MQTT and HTTP
  - Harder to debug and monitor compared to TCP-based protocols

HTTP (HyperText Transfer Protocol)

- **Best for:** Simple one-way communication from devices to the cloud
- **Advantages:**
  - Easy to test/debug (you can even use curl)
  - Works well for periodic updates

    - Widely supported across languages/devices
- **Disadvantages**:
    - High overhead — each request carries headers and a handshake
    - Stateless — doesn't support real-time bi-directional communication
    - Less efficient in low-bandwidth or battery-constrained environments

LwM2M (Lightweight Machine-to-Machine)

- **Best for:** Telecoms and device fleets (e.g., smart meters, smart cities)
- **Advantages:**
    - Built on CoAP but adds device registration, firmware updates, and security
    - Scalable device management out of the box
    - Efficient for constrained networks and devices
- **Disadvantages:**
    - Steeper learning curve
    - More complex integration process
    - Less common in DIY/hobbyist or microcontroller projects

SNMP (Simple Network Management Protocol)

- **Best for:** Monitoring networking and enterprise devices (switches, routers, etc.)
- **Advantages:**
    - Industry standard for IT infrastructure monitoring
    - Devices expose pre-defined metrics (CPU, memory, status)
- **Disadvantages:**
    - Not suited for embedded IoT
    - Limited to polling-based monitoring (not event-driven)
    - Legacy protocol with limited flexibility for modern IoT use cases

4.2. What are the differences between shared and client attributes, and when should each be used?

Characteristics	Shared attribute	Client-side attributes
Similarity	Only for Devices	
Difference	The device firmware/application may request the value of the shared attribute(s) or subscribe to the updates of the attribute(s).	The device firmware/application may send the value of the attributes from the device to the platform.

5. Reference

1. MQTT.org, "MQTT: The Standard for IoT Messaging," [Online]. Available: https://mqtt.org/. [Accessed: April 7, 2025].
2. Open Mobile Alliance, "Lightweight Machine-to-Machine (LwM2M)," Open Mobile Alliance, [Online]. Available: https://www.openmobilealliance.org/lwm2m/. [Accessed: Apr. 7, 2025].
3. R. Fielding, J. Gettys, and J. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1," Request for Comments (RFC) 2616, W3C, 1999. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616.txt. [Accessed: Apr. 7, 2025].
4. ThingsBoard Authors, "MQTT Device API Reference," ThingsBoard Documentation. [Online]. Available: https://thingsboard.io/docs/reference/mqtt-api/. [Accessed: April 7, 2025].
5. ThingsBoard, "CoAP Device API Reference," ThingsBoard Documentation, [Online]. Available: https://thingsboard.io/docs/reference/coap-api/. [Accessed: Apr. 7, 2025].
6. ThingsBoard, "SNMP API," ThingsBoard Documentation, [Online]. Available: https://thingsboard.io/docs/reference/snmp-api/. [Accessed: Apr. 7, 2025].
7. ThingsBoard, "LWM2M Device API Reference," ThingsBoard Documentation, [Online]. Available: https://thingsboard.io/docs/reference/lwm2m-api/. [Accessed: Apr. 7, 2025].
8. ThingsBoard, "HTTP Device API Reference," ThingsBoard Documentation, [Online]. Available: https://thingsboard.io/docs/reference/http-api/. [Accessed: Apr. 7, 2025].
9. ThingsBoard Authors, "Device Connectivity Protocols," ThingsBoard Documentation. [Online]. Available: https://thingsboard.io/docs/reference/protocols/. [Accessed: April 7, 2025].
10. ThingsBoard, "Attributes," ThingsBoard Documentation, [Online]. Available: https://thingsboard.io/docs/user-guide/attributes/. [Accessed: Apr. 7, 2025].
11. Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Internet Engineering Task Force, June 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252