EMSIB – IoT & Control Component

Technical Report – 1st integration

Authors: Aday García López & Cristian Costa

Course: Software Engineering

University: Lodz University of Technology

Academic Year: 2025/2026

Table of contents

1. Introduction

The purpose of this document is to describe the implementation of the IoT & Control Component developed as part of the EMSIB project (Energy Management System in Intelligent Buildings). This module forms one of the lower layers of the system, responsible for interacting with physical sensors and devices in the building.

Our component allows EMSIB to:

- receive telemetry data from sensors,
- control IoT devices such as HVAC or lighting systems,
- apply optimization scenarios generated by the Forecast & Optimization module,
- simulate MQTT-based communication between devices and the EMSIB platform.

The implementation was done in Python using FastAPI, following the API specification provided by the teachers.

2. Component Responsibilities

The IoT & Control Component performs the following key functions:

2.1 Telemetry Ingestion

Receives real-time sensor data (e.g., temperature, humidity, energy consumption) via an HTTP endpoint and stores it for later analysis or forwarding.

2.2 Device Control

Executes control commands, such as changing the temperature of an HVAC system or switching lights on/off.

2.3 Optimization Scenario Application

Receives optimization scenarios from the Forecast & Optimization module and forwards the actions to the required devices.

2.4 MQTT Simulation

Although a full MQTT broker is not implemented, the component simulates MQTT publishing to represent how real IoT communication would occur.

3. Project Structure

The project follows a simple but modular organization:

```
IoT-Control-Component/
        README.md
        requirements.txt
        /src
                main.py
                telemetry.py
                device_control.py
                optimization.py
                storage.py
```

Each file has a specific responsibility:

File	Description
main.py	Main FastAPI server with all implemented endpoints
telemetry.py	Handles telemetry storage logic
device_control.py	Simulates sending commands to IoT devices
optimization.py	Stores and applies optimization scenarios
storage.py	Simulated storage module (simple in-memory list)
README.md	Documentation of the component
requirements.txt	Dependencies required to run the server

The component implements the required endpoints from the “IoT & Control Requirements” document.

4.1 POST /iot/telemetry

Description:

Receives telemetry packets sent by sensors.

Input example:

```
{
  "deviceId": "sensor-01",
  "timestamp": "2024-11-12T10:45:00Z",
  "metrics": {
    "temperature": 22.5,
    "humidity": 55
  }
}
```

Output:

```
{"status": "ok"}
```

Internal behavior:

The telemetry is validated and stored in the TelemetryStorage class.

4.2 POST /iot/device-control/{deviceId}/command

Description:

Executes a control action on a device (simulated MQTT).

Input example:

```
{
  "action": "setTemperature",
  "value": 20
}
```

Output:

{"status": "sent", "deviceId": "hvac-01"}

Internal behavior:

The component prints the control operation and simulates an MQTT publish.

4.3 POST /iot/optimization/apply

Description:

Receives optimization scenarios generated by the Optimization Module.

Input example:


```
{
  "scenarioId": "opt123",
  "actions": [
    {
      "deviceId": "hvac-01",
      "command": {"action": "setTemperature", "value": 19}
    }
  ]
}
```


Output:

{"status": "accepted", "scenarioId": "opt123"}

Internal behavior:

The scenario is stored in memory and actions are executed through the Device Controller.

5. Internal Module Descriptions

5.1 Telemetry Module

Stores incoming telemetry in a simple list.

This simulates a more advanced time-series database.

5.2 Device Controller

Executes device operations and simulates MQTT publishing.

In a real system, this would communicate with the actual devices.

5.3 Optimization Engine

Stores optimization scenarios and applies actions through the controller.

5.4 Storage Module

A generic structure for storing elements. Used mainly for extensibility.

6. How to Run the Component

Install dependencies

pip install -r requirements.txt

Run the server

uvicorn src.main:app –reload

Test the endpoints

Once running, open:

http://127.0.0.1:8000/docs

FastAPI automatically generates interactive API documentation.

7. Testing the Component

The component can be tested using:

- Postman
- cURL
- FastAPI's built-in Swagger UI

Example cURL command for telemetry:

```
curl -X POST "http://localhost:8000/iot/telemetry" \
  -H "Content-Type: application/json" \
  -d '{"deviceId":"sensor-01","metrics":{"temperature":23}}'
```

If everything works, the console will display:

Saving telemetry: {...}

8. Conclusion

The IoT & Control Component successfully implements the required functionality for data ingestion, device control, and optimization scenario execution in the EMSIB system. Although simplified, the component demonstrates the essential logic expected in a real IoT management system and can serve as a foundation for more advanced development.

The work was completed collaboratively by both authors and follows the architecture and API guidelines provided during the course.