EMSIB – IoT & Control Component

Technical Documentation

Authors: Aday García López & Cristian Costa

Course: Software Engineering

University: Lodz University of Technology

Academic Year: 2025/2026

Table of contents

1. Introduction

This document describes the Technical Documentation of the IoT & Control Component developed as part of the EMSIB project. The component corresponds to the Device and Sensor Layer of the system architecture and is responsible for handling communication with physical sensors and devices inside an intelligent building.

This documentation reflects the state of the component after the Second Integration phase. At this stage, the module has evolved from a simple prototype into a structured and modular microservice, ready to be integrated with other EMSIB components such as optimization and analysis modules.

The document focuses on architecture, internal structure, data models, APIs, and deployment aspects of the component.

2. Component Overview

The IoT & Control Component acts as an interface between the EMSIB system and the physical building infrastructure. Its main role is to:

- Receive telemetry data from sensors.
- Manage and control building devices.
- Apply optimization scenarios generated by higher-level modules.
- Maintain a live state of devices and sensor data.

The component is implemented as a RESTful service using FastAPI and is designed to be deployed as an independent microservice.

3. Architecture Description

The component follows a modular architecture based on FastAPI and dependency injection principles.

The main architectural elements are:

- API layer responsible for handling HTTP requests.
- Business logic layer managing telemetry, devices, and optimization logic.
- In-memory storage layer maintaining the current system state.

- Dependency injection layer providing shared resources across the application.

All interactions with the component are performed through REST endpoints. The component does not directly communicate with other EMSIB modules but exposes APIs that allow higher layers to interact with it.

4. Internal Structure

The internal structure of the component is organized into several Python modules inside the src directory.

main.py

Acts as the entry point of the application. It initializes the FastAPI app, registers routers, and configures dependencies.

models.py

Defines data models using Pydantic. These models describe the structure of telemetry data, device states, commands, and API responses.

storage.py

Implements the InMemoryStore, which is responsible for storing:

- Telemetry data.
- Device states.
- Optimization-related information.

This storage simulates a persistent database for integration purposes.

telemetry.py

Handles the ingestion and processing of sensor data received through the API.

devices.py

Manages device registration and device-related operations.

device_control.py

Contains the logic for executing commands on devices. In the current integration, execution is simulated.

optimization.py

Processes optimization scenarios and applies their actions to devices.

deps.py

Defines shared dependencies used across the application, following FastAPI best practices.

5. Data Models

The component uses structured data models to ensure consistency and validation of incoming and outgoing data.

The main types of data handled by the component include:

- Telemetry data from sensors.
- Device command requests.
- Device state representations.
- Optimization scenarios and actions.

All models are validated using Pydantic to prevent invalid or malformed data from entering the system.

6. API Endpoints

The IoT & Control Component exposes several REST endpoints that represent its core functionality.

Telemetry Ingestion

POST /iot/telemetry
Receives telemetry data from sensors and stores it in the in-memory storage.

Device Control

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

Receives a command for a specific device and executes it logically within the component.

Optimization Scenario Application

POST /iot/optimization/apply

Receives optimization scenarios from higher layers and applies their actions.

Device State Retrieval

GET /iot/state/{deviceId}

Returns the current state of a specific device.

Global State Monitoring

GET /iot/state/live

Returns a snapshot of the current state of all devices managed by the component.

7. State Management

A key improvement introduced in the second integration is live state management.

The InMemoryStore maintains an up-to-date view of:

- Latest telemetry values.
- Current device states.
- Results of applied optimization actions.

This allows the component to provide real-time information to other EMSIB modules and supports monitoring use cases.

8. Dependency Injection

The component uses FastAPI dependency injection to manage shared resources such as the in-memory storage.

This design improves:

- Code readability.
- Testability.

- Separation of concerns.

All modules access shared state through injected dependencies rather than global variables.

9. Deployment and Docker Support

The component includes Docker support to allow deployment as a standalone microservice.

A Dockerfile and dockerignore file are provided to:

- Build the application image.
- Install dependencies.
- Expose the service through a containerized environment.

This enables easy integration into container-based deployments and orchestration platforms.

10. Changes from First to Second Integration

Compared to the first integration, the second integration introduces several improvements:

- Structured internal architecture.
- Live state management.
- Dependency injection.
- Expanded API endpoints.
- Docker-based deployment support.

These changes move the component closer to a production-ready microservice.

11. Conclusions

The IoT & Control Component successfully fulfills its responsibilities within the EMSIB system after the second integration phase. The component is modular, extensible, and ready to be integrated with other system components.

The current implementation provides a solid foundation for future extensions, including persistent storage, real MQTT communication, and integration with physical IoT devices.