TUGAS MATA KULIAH SISTEM KONTROL TERDISTRIBUSI

"Edge Gateway & Cloud Integration Project using DWSIM, Influx DB and Thinksboard in Hydroponic/Houseplant Factory"

Dosen: Ahmad Radhy, S.Si., M.Si

Oleh:

Muhammad Sahal Rajendra
2042231036

Muhammad Rifal Faiz Arivito
2042231067

PRODI D-4 TEKNOLOGI REKAYASA INSTRUMENTASI
DEPARTEMEN TEKNIK INSTRUMENTASI
FAKULTAS VOKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2025

Daftar Isi

1 Dasar Teori

1.1 Sensor SHT20

Sensor SHT20 merupakan transduser digital terintegrasi yang berfungsi untuk melakukan kuantifikasi terhadap dua parameter fisis secara simultan, yaitu temperatur dan kelembaban relatif (Relative Humidity, RH). Sensor ini, yang dikembangkan oleh Sensirion, beroperasi berdasarkan prinsip sensorik kapasitif untuk pengukuran kelembaban dan sensor band-gap untuk pengukuran temperatur. Data keluaran dari SHT20 telah melalui proses kalibrasi dan linearisasi internal pabrikan, yang kemudian ditransmisikan dalam format digital melalui antarmuka komunikasi serial I2C (Inter-Integrated Circuit). Karakteristik utama yang meliputi akurasi tinggi, waktu respons yang cepat, serta stabilitas jangka panjang yang andal menjadikan SHT20 sebagai komponen yang relevan untuk aplikasi sistem pemantauan lingkungan yang memerlukan presisi tinggi, serta karakteristik tersebut, sensor SHT20 sangat sesuai untuk diaplikasikan dalam sistem edge gateway berbasis ESP32-S3 yang mengirimkan data ke platform cloud seperti ThingSpeak untuk pemantauan kondisi lingkungan secara real-time.

Gambar 1: Sensor SHT20

1.2 Mikrokontroller ESP32 S3

Mikrokontroler ESP32-S3 adalah sebuah System on Chip (SoC) performa tinggi yang dikembangkan oleh Espressif Systems, dirancang secara spesifik untuk aplikasi Internet of Things (IoT) dan komputasi tepi (edge computing). Arsitektur perangkat ini ditenagai oleh prosesor dual-core Xtensa® LX7 dengan frekuensi operasi yang dapat mencapai 240 MHz, yang dilengkapi dengan kapabilitas konektivitas nirkabel terintegrasi, mencakup Wi-Fi (IEEE 802.11 b/g/n) dan Bluetooth Low Energy (BLE) 5.0.

Gambar 2: ESP32S3

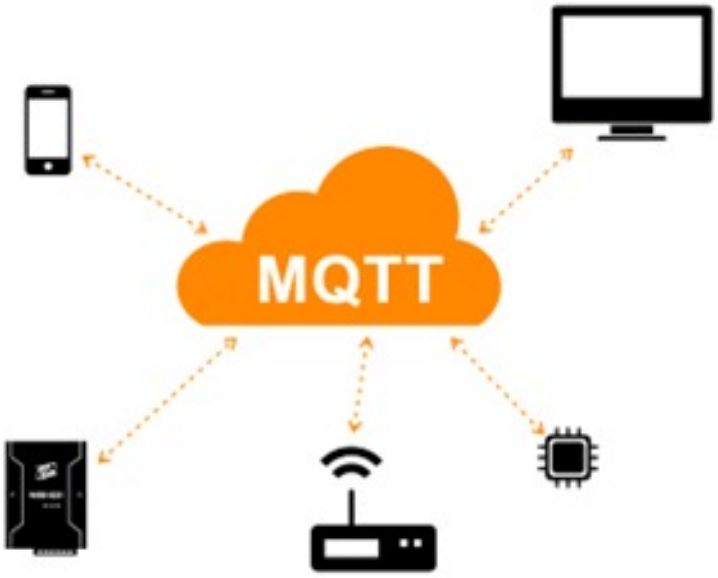
Selain itu, ESP32-S3 memiliki akselerator perangkat keras untuk instruksi vektor yang dapat dimanfaatkan untuk aplikasi kecerdasan buatan dan machine learning (AI/ML). Dalam arsitektur sistem yang diusulkan, ESP32-S3 memegang peranan krusial sebagai edge gateway, yang bertugas melakukan akuisisi data dari sensor melalui protokol Modbus RTU dan mendistribusikannya ke platform hilir.

1.3 Protokol Komunikasi Modbus RTU

Modbus RTU (Remote Terminal Unit) merupakan sebuah protokol komunikasi serial yang telah menjadi standar de facto dalam otomasi industri untuk interoperabilitas antar-perangkat elektronik. Protokol ini mengimplementasikan arsitektur komunikasi master-slave, di mana satu perangkat master (dalam penelitian ini adalah ESP32-S3) dapat menginisiasi permintaan data atau mengirim perintah ke beberapa perangkat slave (sensor Modbus). Representasi data dalam Modbus RTU menggunakan format biner yang efisien, yang umumnya ditransmisikan melalui lapisan fisik RS-485. Penggunaan RS-485 memungkinkan komunikasi yang reliabel pada jarak jauh, hingga 1.200 meter, dengan laju data yang memadai untuk aplikasi kontrol dan monitoring.

1.4 MQTT

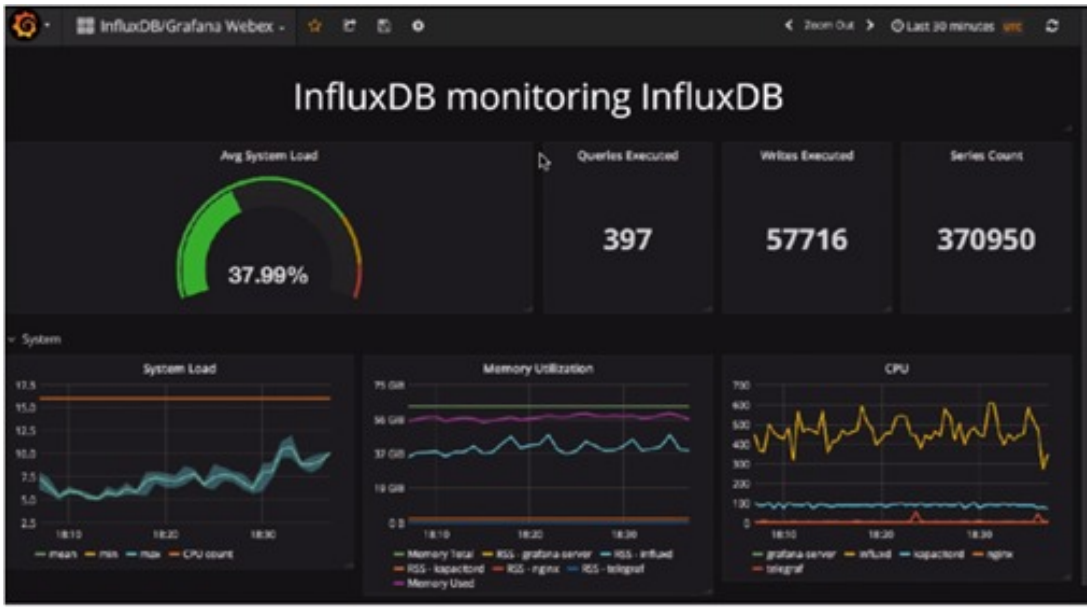
Message Queuing Telemetry Transport (MQTT) adalah sebuah protokol pertukaran pesan berbasis model publikasi-langganan (publish-subscribe) yang dirancang untuk efisiensi dan keringkasan, sehingga optimal untuk perangkat dengan sumber daya terbatas dan jaringan dengan latensi tinggi atau bandwidth rendah. Arsitekturnya terdiri dari tiga entitas utama: publisher (pengirim pesan), subscriber (penerima pesan), dan broker (server perantara). Publisher mengirimkan pesan ke sebuah "topik" pada broker tanpa perlu mengetahui identitas subscriber. Selanjutnya, broker bertugas memfilter dan mendistribusikan pesan tersebut kepada semua subscriber yang terdaftar pada topik yang relevan. Paradigma ini memisahkan (decouple) antara produsen dan konsumen data, sehingga menghasilkan sistem yang sangat skalabel dan fleksibel untuk transmisi data telemetri dari edge device ke cloud server.



Gambar 3: Platform MQTT

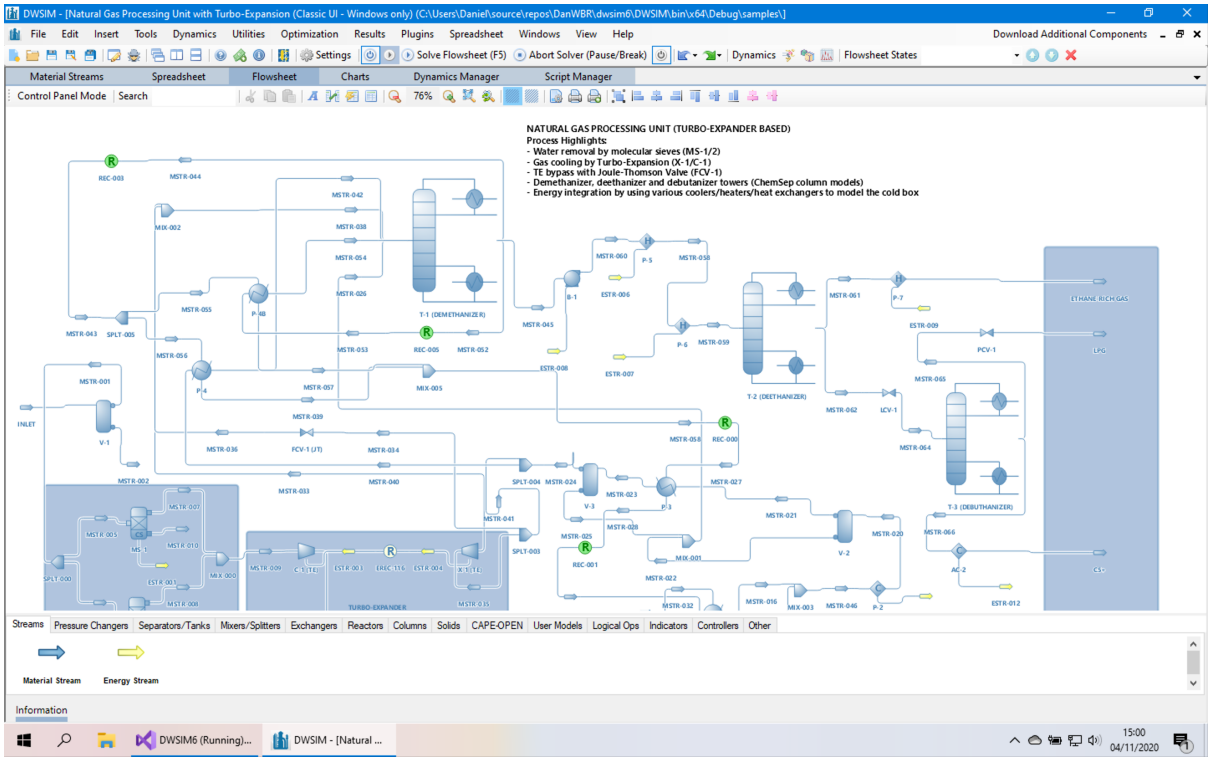
1.5 InfluxDB

InfluxDB merupakan sebuah sistem manajemen basis data (database management system) open-source yang secara khusus dioptimalkan untuk menangani data deret waktu (time-series data). Data deret waktu, yang didefinisikan sebagai sekuens data yang diindeks berdasarkan waktu, merupakan tipe data fundamental dalam aplikasi IoT dan pemantauan industri. InfluxDB didesain untuk memiliki performa ingestion (penulisan) dan kueri yang sangat tinggi, yang esensial untuk menangani volume data sensor yang besar secara real-time. Platform ini menyediakan bahasa kueri (InfluxQL atau Flux) yang fungsionalitasnya menyerupai SQL untuk melakukan agregasi, analisis, dan transformasi data. Dalam sistem ini, InfluxDB berfungsi sebagai data historian atau repositori utama untuk penyimpanan persisten data sensor.



Gambar 4: Tampilan InfluxDB

1.6 DWSIM

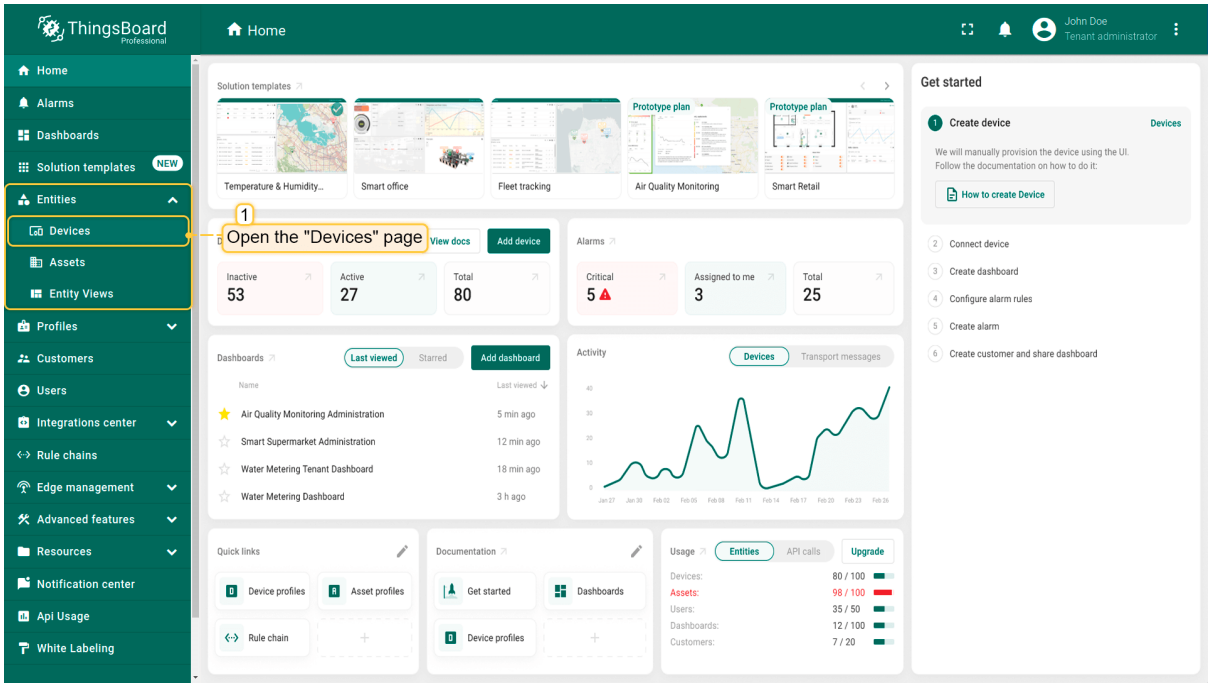


Gambar 5: Software DWSIM

DWSIM adalah sebuah perangkat lunak open-source yang berfungsi sebagai simulator proses kimia (Chemical Process Simulator). DWSIM memungkinkan pemodelan, analisis, dan simulasi proses dinamik yang melibatkan fenomena kimia dan termodinamika. Platform ini menyediakan librari model termodinamika yang ekstensif (misalnya Peng-Robinson, NRTL) serta berbagai unit operasi standar industri, yang memungkinkan konstruksi diagram alir proses secara virtual.

Salah satu fungsionalitas relevan dari DWSIM adalah kemampuannya untuk mengekspor hasil simulasi ke dalam format data terstruktur seperti XML (eXtensible Markup Language). Salah satu keunggulan signifikan DWSIM dalam proyek ini adalah interoperabilitasnya dengan perangkat lunak lain. DWSIM mampu mengekspor hasil simulasi-baik dalam kondisi tunak (steady-state) maupun dinamis-ke dalam format data terstruktur seperti CSV (Comma-Separated Values) dan XML (eXtensible Markup Language). Fitur ekspor ke XML inilah yang menjadi mekanisme kunci dalam arsitektur sistem yang diusulkan. File XML yang dihasilkan oleh DWSIM berisi data numerik dari parameter-parameter simulasi yang relevan, yang kemudian dapat diurai (parsed) secara programatik oleh skrip Python pada gateway. Integrasi ini memungkinkan penggabungan antara data hasil pemodelan matematis dari DWSIM dengan data empiris yang diakuisisi dari sensor fisik, untuk ditampilkan dalam satu ekosistem pemantauan terpadu.

1.7 ThingsBoard



Gambar 6: Dashboard Software DWSIM

ThingsBoard adalah sebuah platform IoT open-source yang menyediakan infrastruktur komprehensif untuk pengumpulan, pemrosesan, visualisasi, dan manajemen perangkat IoT. Platform ini dirancang dengan skalabilitas sebagai pertimbangan utama, sehingga dapat mengakomodasi dari implementasi skala kecil hingga sistem industri berskala besar. ThingsBoard mendukung beragam protokol komunikasi, termasuk MQTT, HTTP, dan CoAP, yang menjamin interoperabilitas dengan berbagai perangkat keras. Data dalam ThingsBoard diorganisasikan dalam sebuah model yang terstruktur, meliputi Devices (representasi perangkat fisik), Assets (representasi entitas logis seperti gedung atau pabrik), Telemetry (data deret waktu dari sensor), dan Attributes (metadata statis atau semi-statis dari perangkat, seperti nomor seri atau konfigurasi).

Fungsionalitas utamanya yang paling menonjol adalah fitur pembuatan dasbor interaktif. Pengguna dapat merancang antarmuka visual (HMI Human-Machine Interface) secara drag-and-drop menggunakan koleksi widget yang ekstensif, seperti grafik deret waktu, pengukur analog dan digital (gauges), tabel, peta geografis, dan saklar kontrol (control widgets). Dasbor ini dapat menampilkan data secara real-time dan historis, serta memungkinkan pengguna mengirimkan perintah kembali ke perangkat (RPC Remote Procedure Call).

2 Tinjauan Pustaka

2.1 Sensor SHT20 RS-485 (Modbus RTU)

Sensor SHT20 merupakan sensor digital yang berfungsi untuk mengukur suhu dan kelembapan relatif dengan tingkat akurasi yang tinggi. Untuk aplikasi industri, sensor ini diintegrasikan dengan modul RS-485 yang memanfaatkan protokol Modbus RTU. Komunikasi RS-485 memungkinkan transmisi data jarak jauh yang andal dan mendukung koneksi multi-drop, menjadikannya ideal untuk pemantauan lingkungan industri terdistribusi.

2.2 ESP32-S3

ESP32-S3 berfungsi sebagai edge gateway yang diprogram menggunakan Embedded Rust, yang menawarkan keamanan memori dan efisiensi. Peran utamanya adalah mengakuisisi data dari sensor SHT20 melalui Modbus RTU (RS-485) dan memprosesnya sebelum dikirim. Selain itu, ESP32-S3 juga dapat mengontrol aktuator (seperti pompa atau katup), sehingga berfungsi ganda sebagai pengumpul data sekaligus pengendali sistem.

2.3 Backend Rust

Backend yang dikembangkan dengan Rust dengan Integrasi InfluxDB dan ThingsBoard via MQTT bertindak sebagai jembatan antara edge gateway (ESP32-S3) dan platform cloud. Sistem ini bertugas membaca data yang tersimpan di InfluxDB dan menyalurkannya ke ThingsBoard Cloud menggunakan protokol MQTT. Arsitektur ini memungkinkan data tersimpan secara lokal sekaligus dapat diakses secara real-time di cloud.

2.4 InfluxDB

InfluxDB adalah basis data time-series yang dioptimalkan untuk menyimpan data sensor berkala secara efisien. Database ini dirancang untuk menangani query berbasis waktu dengan cepat. Data suhu dan kelembaban dari SHT20 disimpan di InfluxDB untuk memfasilitasi analisis tren, perhitungan historis, dan pelacakan data jangka panjang.

2.5 ThingsBoard Cloud

ThingsBoard adalah platform IoT cloud yang berfungsi untuk visualisasi data sensor melalui dashboard interaktif. Data dari InfluxDB dikirim secara real-time via MQTT untuk ditampilkan. Platform ini memungkinkan kustomisasi dashboard (grafik, indikator) untuk memudahkan pemantauan dan pengambilan keputusan, serta menyediakan fitur device management dan sistem peringatan (alert).

2.6 DWSIM

DWSIM adalah simulator proses kimia open-source untuk memodelkan, menganalisis, dan mengoptimalkan proses industri. Perangkat lunak ini menyediakan fitur perhitungan termodinamika, neraca massa/energi, dan desain peralatan proses. DWSIM memiliki

antarmuka grafis untuk menggambar flowsheet dan mendukung berbagai model termodinamika (seperti Peng-Robinson). Keunggulan utamanya adalah sifatnya yang gratis dan fleksibel sebagai alternatif dari perangkat lunak komersial seperti Aspen HYSYS.

3 Metodologi

3.1 Alat dan Bahan

Peralatan yang dibutuhkan untuk project ini adalah sebagai berikut:

1. Mikrokontroler ESP32 S3
2. Sensor SHT20 Modbus
3. TTL to RS485 Adapter Module
4. Motor Servo SG90
5. Relay 12v 1 Channel
6. Kabel Jumper
7. Breadboard
8. Software InfluxDB
9. Software Thingsboard
10. Software DWSIM

3.2 Wiring Diagram

1. Catu Daya (Power)
    a) Kabel Positif (V+) / Merah
        - Pin VIN (atau 5V) pada ESP32.
        - Pin VCC pada modul Relay (merah).
        - Pin VCC (kabel merah) pada Servo.
        - Pin VCC pada modul RS485 (biru).
        - Kabel Positif (merah) pada Kipas.

    b) Kabel Negatif (V-) / Ground / Hitam
        - Pin GND pada ESP32.
        - Pin GND pada modul Relay.
        - Pin GND (kabel hitam/coklat) pada Servo.
        - Pin GND pada modul RS485.
        - Pin COM (Common) di terminal output Relay.
2. Kontrol Aktuator (Servo & Kipas)
    - ESP32 mengirim sinyal untuk menggerakkan komponen Motor Servo, dimana:
        - Pin sinyal (kabel kuning) dari Servo terhubung ke GPIO 1 pada ESP32.
    - Kontrol Kipas (melalui Relay):
        - Pin sinyal (IN) dari modul Relay terhubung ke GPIO 2 pada ESP32.

- Relay ini berfungsi sebagai saklar untuk memutus/menyambung jalur negatif kipas.
- Kabel Negatif (hitam) dari Kipas terhubung ke terminal NO (Normally Open) pada Relay.
- Saat GPIO 2 mengaktifkan relay, NO akan terhubung ke COM (yang sudah terhubung ke Ground), sehingga kipas menyala.

3. Komunikasi Sensor (Modbus RS485)

- ESP32 berkomunikasi dengan sensor eksternal (yang tidak terlihat di diagram, misal SHT20) menggunakan modul RS485:
  - GPIO 10 (RX) terhubung ke pin RO (Receiver Output) pada modul RS485.
  - GPIO 11 (TX) terhubung ke pin DI (Driver Input) pada modul RS485.
  - GPIO 12 terhubung ke pin DE (Driver Enable) pada modul RS485.
  - GPIO 13 terhubung ke pin RE (Receiver Enable) pada modul RS485.
  - Terminal A dan B pada modul RS485 akan dihubungkan ke kabel A dan B dari sensor.

3.3 Perancangan Perangkat Lunak

3.3.1 main.rs

```
//main.rs
extern crate alloc;
use anyhow::{bail, Context, Result};
use log::{error, info, warn};
use esp_idf_sys as sys; // C-API (MQTT & HTTP)
use alloc::ffi::CString;
use alloc::string::String;
use alloc::string::ToString;
use esp_idf_svc::{
    eventloop::EspSystemEventLoop,
    log::EspLogger,
    nvs::EspDefaultNvsPartition,
    wifi::{AuthMethod, BlockingWifi, ClientConfiguration as StaCfg, Configuration as WifiCfg,
    EspWifi},
};
use esp_idf_svc::hal::{
    delay::FreeRtos,
    gpio::{AnyIOPin, Output, PinDriver},
    ledc::{config::TimerConfig, LedcDriver, LedcTimerDriver, Resolution},
    peripherals::Peripherals,
    uart::{config::Config as UartConfig, UartDriver},
    units::Hertz,
};
use serde_json::json;
//===== tambahan untuk TCP server & concurrency =====
use std::io::Write as IoWrite;
use std::net::{TcpListener, TcpStream};
use std::sync::{mpsc, Arc, Mutex};
use std::thread;
// ===== KONFIGURASI =====
// Wi-Fi
const WIFI_SSID: &str = "Xiryus";
const WIFI_PASS: &str = "1qaz2wsx";
//ThingsBoard Cloud (MQTT Basic)
const TB_MQTT_URL: &str = "mqtt://mqtt.thingsboard.cloud:1883";
const TB_CLIENT_ID: &str = "esp32s3-eggdhis";
const TB_USERNAME: &str = "eggdhis";
const TB_PASSWORD: &str = "453621";
// InfluxDB (Cloud atau lokal)
const INFLUX_URL: &str = "https://us-east-1-1.aws.cloud2.influxdata.com";
const INFLUX_ORG_ID: &str = "882113367e216236";
const INFLUX_BUCKET: &str = "skt13eggdhis";
const INFLUX_TOKEN: &str = "w16KRRwJKin17Pn94QudXx8yjhCBkxc-OZIoRp5zGmXrG0oYyRb2d1j7Lhqyw7-e9hyUtj5WshTk0ilyp8DnPQ==";
// Modbus (SHT20 via RS485)
const MODBUS_ID: u8 = 0x01;
const BAUD: u32 = 9_600;
// TCP Server (untuk stream JSON ke klien)
const TCP_LISTEN_ADDR: &str = "0.0.0.0";
const TCP_LISTEN_PORT: u16 = 7878;
// Relay (AKTIF saat RH > 60%)
const RELAY_GPIO_IS_LOW_ACTIVE: bool = true; // true: aktif LOW (umum); false: aktif HIGH
```

```rust
const RH_ON_THRESHOLD: f32 = 59.0; // > 60% RH menyalakan relay
// ===== Util =====
#[inline(always)]
fn ms_to_ticks(ms: u32) -> u32 {
    (ms as u64 * sys::configTICK_RATE_HZ as u64 / 1000) as u32
}
fn looks_like_uuid(s: &str) -> bool {
    s.len() == 36 && s.matches('-').count() == 4
}
// Minimal percent-encoding untuk komponen query (RFC 3986 unreserved: ALNUM-._~)
fn url_encode_component(input: &str) -> String {
    let mut out = String::with_capacity(input.len());
    for b in input.as_bytes() {
        let c = *b as char;
        if c.is_ascii_alphanumeric() || "-._~".contains(c) {
            out.push(c);
        } else {
            let _ = core::fmt::write(&mut out, format_args!("%{:02X}", b));
        }
    }
    out
}
// ===== MQTT client (C-API) ==========
struct SimpleMqttClient {
    client: *mut sys::esp_mqtt_client,
}
impl SimpleMqttClient {
    fn new(broker_url: &str, username: &str, password: &str, client_id: &str) -> Result<Self> {
        unsafe {
            let broker_url_cstr = CString::new(broker_url)?;
            let username_cstr = CString::new(username)?;
            let password_cstr = CString::new(password)?;
            let client_id_cstr = CString::new(client_id)?;
            let mut cfg: sys::esp_mqtt_client_config_t = core::mem::zeroed();
            cfg.broker.address.uri = broker_url_cstr.as_ptr() as *const i8;
            cfg.credentials.username = username_cstr.as_ptr() as *const i8;
            cfg.credentials.client_id = client_id_cstr.as_ptr() as *const u8;
            cfg.credentials.authentication.password = password_cstr.as_ptr() as *const i8;
            cfg.session.keepalive = 30; // detik
            cfg.network.timeout_ms = 20_000; //ms
            let client = sys::esp_mqtt_client_init(&cfg);
            if client.is_null() {
                bail!("Failed to initialize MQTT client");
            }
            let err = sys::esp_mqtt_client_start(client);
            if err != sys::ESP_OK {
                bail!("Failed to start MQTT client, esp_err=0x{:X}", err as u32);
            }
            sys::vTaskDelay(ms_to_ticks(2500));
            Ok(Self { client })
        }
    }
    fn publish(&self, topic: &str, data: &str) -> Result<()> {
        unsafe {
            let topic_c = CString::new(topic)?;
            let msg_id = sys::esp_mqtt_client_publish(
                self.client,
                topic_c.as_ptr(),
                data.as_ptr() as *const i8,
                data.len() as i32,
                1,
                0,
            );
            if msg_id < 0 {
                bail!("Failed to publish message, code: {}", msg_id);
            }
            info!("MQTT published (id={})", msg_id);
            Ok(())
        }
    }
}
impl Drop for SimpleMqttClient {
    fn drop(&mut self) {
        unsafe {
            sys::esp_mqtt_client_stop(self.client);
            sys::esp_mqtt_client_destroy(self.client);
        }
    }
}
// ===== CRC & Modbus util ===
fn crc16_modbus(mut crc: u16, byte: u8) -> u16 {
    crc ^= byte as u16;
    for _ in 0..8 {
        crc = if (crc & 1) != 0 { (crc >> 1) ^ 0xA001 } else { crc >> 1 };
    }
    crc
}
fn modbus_crc(data: &[u8]) -> u16 {
    let mut crc: u16 = 0xFFFF;
    for &b in data { crc = crc16_modbus(crc, b); }
    crc
}
fn build_read_req(slave: u8, func: u8, start_reg: u16, qty: u16) -> heapless::Vec<u8, 256> {
    use heapless::Vec;
    let mut pdu: Vec<u8, 256> = Vec::new();
    pdu.push(slave).unwrap();
    pdu.push(func).unwrap();
    pdu.push((start_reg >> 8) as u8).unwrap();
    pdu.push((start_reg & 0xFF) as u8).unwrap();
```

```
    pdu.push((qty >> 8) as u8).unwrap();
    pdu.push((qty & 0xFF) as u8).unwrap();
    let crc = modbus_crc(&pdu);
    pdu.push((crc & 0xFF) as u8).unwrap();
    pdu.push((crc >> 8) as u8).unwrap();
    pdu
}
fn parse_read_resp(expected_slave: u8, qty: u16, buf: &[u8]) -> Result<heapless::Vec<u16, 64>> {
    use heapless::Vec;
    if buf.len() >= 5 && (buf[1] & 0x80) != 0 {
        let crc_rx = u16::from(buf[4]) << 8 | u16::from(buf[3]);
        let crc_calc = modbus_crc(&buf[..3]);
        if crc_rx == crc_calc {
            let code = buf[2];
            bail!("Modbus exception 0x{:02X}", code);
        } else {
            bail!("Exception frame CRC mismatch");
        }
    }
    let need = 1 + 1 + 1 + (2 * qty as usize) + 2;
    if buf.len() < need { bail!("Response too short: got {}, need {}", buf.len(), need); }
    if buf[0] != expected_slave { bail!("Unexpected slave id: got {}, expected {}", buf[0], expected_slave); }
    if buf[1] != 0x03 && buf[1] != 0x04 { bail!("Unexpected function code: 0x{:02X}", buf[1]); }
    let bc = buf[2] as usize;
    if bc != 2 * qty as usize { bail!("Unexpected byte count: {}", bc); }
    let crc_rx = u16::from(buf[need - 1]) << 8 | u16::from(buf[need - 2]);
    let crc_calc = modbus_crc(&buf[..need - 2]);
    if crc_rx != crc_calc { bail!("CRC mismatch: rx=0x{:04X}, calc=0x{:04X}", crc_rx, crc_calc); }
    let mut out: Vec<u16, 64> = Vec::new();
    for i in 0..qty as usize {
        let hi = buf[3 + 2 * i] as u16;
        let lo = buf[3 + 2 * i + 1] as u16;
        out.push((hi << 8) | lo).unwrap();
    }
    Ok(out)
}
//===== RS485 helpers ==========
fn rs485_write(
    uart: &UartDriver<'_>,
    de: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21, Output>,
    data: &[u8],
) -> Result<()> {
    de.set_high()?;
    FreeRtos::delay_ms(3);
    uart.write(data)?;
    uart.wait_tx_done(200)?;
    de.set_low()?;
    FreeRtos::delay_ms(3);
    Ok(())
}
fn rs485_read(uart: &UartDriver<'_>, dst: &mut [u8], ticks: u32) -> Result<usize> {
    uart.clear_rx()?;
    let n = uart.read(dst, ticks)?;
    use core::fmt::Write as _;
    let mut s = String::new();
    for b in &dst[..n] { write!(&mut s, "{:02X} ", b).ok(); }
    info!("RS485 RX {} bytes: {}", n, s);
    Ok(n)
}
fn try_read(
    uart: &UartDriver<'_>,
    de: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21, Output>,
    func: u8, start: u16, qty: u16, ticks: u32,
) -> Result<heapless::Vec<u16, 64>> {
    let req = build_read_req(MODBUS_ID, func, start, qty);
    rs485_write(uart, de, &req)?;
    let mut buf = [0u8; 64];
    let n = rs485_read(uart, &mut buf, ticks)?;
    parse_read_resp(MODBUS_ID, qty, &buf[..n])
}
fn probe_map(
    uart: &UartDriver<'_>,
    de: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21, Output>,
) -> Option<(u8, u16, u16)> {
    for &fc in &[0x04u8, 0x03u8] {
        for start in 0x0000u16..0x0010u16 {
            for &qty in &[1u16, 2u16] {
                if let Ok(regs) = try_read(uart, de, fc, start, qty, 250) {
                    info!("FOUND: fc=0x{:02X}, start=0x{:04X}, qty={} regs={:04X?}",
                        fc, start, qty, regs.as_slice());
                    return Some((fc, start, qty));
                }
            }
        }
    }
    None
}
fn read_sht20_with_map(
    uart: &UartDriver<'_>,
    de: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21, Output>,
    fc: u8, start: u16, qty: u16,
) -> Result<(f32, f32)> {
    let regs = try_read(uart, de, fc, start, qty, 250)?;
    let (raw_t, raw_h) = if regs.len() >= 2 { (regs[0], regs[1]) } else { (regs[0], 0) };
    let temp_c = (raw_t as f32) * 0.1;
    let rh_pct = (raw_h as f32) * 0.1;
    Ok((temp_c, rh_pct))
}
// ===== Wi-Fi (BlockingWifi) =============
```

```
fn connect_wifi(wifi: &mut BlockingWifi<EspWifi<'static>>) -> Result<()> {
    let cfg = WifiCfg::Client(StaCfg {
        ssid: heapless::String::try_from(WIFI_SSID).unwrap(),
        password: heapless::String::try_from(WIFI_PASS).unwrap(),
        auth_method: AuthMethod::WPA2Personal,
        channel: None,
        ..Default::default()
    });
    wifi.set_configuration(&cfg)?;
    wifi.start()?;
    info!("Wi-Fi driver started");
    wifi.connect()?;
    info!("Wi-Fi connect issued, waiting for netif up...");
    wifi.wait_netif_up()?;
    let ip = wifi.wifi().sta_netif().get_ip_info()?;
    info!("Wi-Fi connected. IP={}", ip.ip);
    unsafe { sys::vTaskDelay(ms_to_ticks(1200)); }
    Ok(())
}
// ===== Influx helpers =============
fn influx_line(measurement: &str, device: &str, t_c: f32, h_pct: f32) -> String {
    format!("{},device={} temperature_c={},humidity_pct={}", measurement, device, t_c, h_pct)
}
fn influx_write(lp: &str) -> Result<()> {
    unsafe {
        let org_q = if looks_like_uuid(INFLUX_ORG_ID) { "orgID" } else { "org" };
        let url = format!(
            "{}/api/v2/write?{}={}&bucket={}&precision=ms",
            INFLUX_URL,
            org_q,
            url_encode_component(INFLUX_ORG_ID),
            url_encode_component(INFLUX_BUCKET)
        );
        let url_c = CString::new(url.as_str())?;
        let mut cfg: sys::esp_http_client_config_t = core::mem::zeroed();
        cfg.url = url_c.as_ptr();
        cfg.method = sys::esp_http_client_method_t_HTTP_METHOD_POST;
        if INFLUX_URL.starts_with("https://") {
            cfg.transport_type = sys::esp_http_client_transport_t_HTTP_TRANSPORT_OVER_SSL;
            cfg.crt_bundle_attach = Some(sys::esp_crt_bundle_attach);
        }
        let client = sys::esp_http_client_init(&cfg);
        if client.is_null() {
            bail!("esp_http_client_init failed");
        }
        // headers
        let h_auth = CString::new("Authorization")?;
        let v_auth = CString::new(format!("Token {}", INFLUX_TOKEN))?;
        let h_ct = CString::new("Content-Type")?;
        let v_ct = CString::new("text/plain; charset=utf-8")?;
        let h_acc = CString::new("Accept")?;
        let v_acc = CString::new("application/json")?;
        let h_conn = CString::new("Connection")?;
        let v_conn = CString::new("close")?;
        sys::esp_http_client_set_header(client, h_auth.as_ptr(), v_auth.as_ptr());
        sys::esp_http_client_set_header(client, h_ct.as_ptr(), v_ct.as_ptr());
        sys::esp_http_client_set_header(client, h_acc.as_ptr(), v_acc.as_ptr());
        sys::esp_http_client_set_header(client, h_conn.as_ptr(), v_conn.as_ptr());
        // body (Line Protocol)
        sys::esp_http_client_set_post_field(client, lp.as_ptr() as *const i8, lp.len() as i32);
        // perform
        let err = sys::esp_http_client_perform(client);
        if err != sys::ESP_OK {
            let e = format!("esp_http_client_perform failed: 0x{:X}", err as u32);
            sys::esp_http_client_cleanup(client);
            bail!(e);
        }
        let status = sys::esp_http_client_get_status_code(client);
        if status != 204 {
            let mut body_buf = [0u8; 256];
            let read = sys::esp_http_client_read_response(client, body_buf.as_mut_ptr() as *mut i8,
                body_buf.len() as i32);
            let body = if read > 0 {
                core::str::from_utf8(&body_buf[..read as usize]).unwrap_or("")
            } else { "" };
            warn!("Influx write failed: HTTP {} Body: {}", status, body);
            sys::esp_http_client_cleanup(client);
            bail!("Influx write HTTP status {}", status);
        } else {
            info!("OK Data Berhasil Dikirim ke InfluxDB");
        }
        sys::esp_http_client_cleanup(client);
        Ok(())
    }
}
// ===== Servo helpers (LEDC) =============
struct Servo {
    ch: LedcDriver<'static>,
    duty_0: u32,
    duty_90: u32,
    duty_180: u32,
}
impl Servo {
    fn new(mut ch: LedcDriver<'static>) -> Result<Self> {
        let max = ch.get_max_duty() as u64; // Bits14 -> 16383
        let period_us = 20_000u64; // 50 Hz -> 20 ms
        let duty_from_us = |us: u32| -> u32 { ((max * us as u64) / period_us) as u32 };
        let duty_0 = duty_from_us(500); // ~0.5 ms (=0deg)
        let duty_90 = duty_from_us(1500); // ~1.5 ms (=90deg)
```

```
        let duty_180 = duty_from_us(2500); // ~2.5 ms (=180deg)
        // Posisi awal: 90deg
        ch.set_duty(duty_90)?;
        ch.enable()?;
        Ok(Self { ch, duty_0, duty_90, duty_180 })
    }
    fn set_0(&mut self) -> Result<()> { self.ch.set_duty(self.duty_0).map_err(Into::into) }
    fn set_90(&mut self) -> Result<()> { self.ch.set_duty(self.duty_90).map_err(Into::into) }
    fn set_180(&mut self) -> Result<()> { self.ch.set_duty(self.duty_180).map_err(Into::into) }
}
#[derive(Copy, Clone, PartialEq, Eq, Debug)]
enum ServoPos { P0, P90, P180 }
// ===== TCP server helpers =================
fn start_tcp_server() -> mpsc::Sender<String> {
    let (tx, rx) = mpsc::channel::<String>();
    let clients: Arc<Mutex<Vec<TcpStream>>> = Arc::new(Mutex::new(Vec::new()));
    // Thread acceptor
    {
        let clients_accept = Arc::clone(&clients);
        thread::spawn(move || {
            let addr = format!("{}:{}", TCP_LISTEN_ADDR, TCP_LISTEN_PORT);
            loop {
                match TcpListener::bind(&addr) {
                    Ok(listener) => {
                        info!("TCP Server listening on {}", addr);
                        listener.set_nonblocking(true).ok(); // non-blocking accept
                        loop {
                            match listener.accept() {
                                Ok((stream, peer)) => {
                                    let _ = stream.set_nodelay(true);
                                    info!("TCP client connected: {}", peer);
                                    if let Ok(mut vec) = clients_accept.lock() {
                                        vec.push(stream);
                                    }
                                }
                                Err(ref e) if e.kind() == std::io::ErrorKind::WouldBlock => {
                                    // tidak ada koneksi baru saat ini
                                    FreeRtos::delay_ms(100);
                                }
                                Err(e) => {
                                    warn!("TCP accept error: {} (rebind)", e);
                                    FreeRtos::delay_ms(1000);
                                    break; // keluar loop dalam -> rebind listener
                                }
                            }
                            // kecilkan beban CPU
                            FreeRtos::delay_ms(10);
                        }
                    }
                    Err(e) => {
                        warn!("TCP bind {} error: {} (retry in 1s)", addr, e);
                        FreeRtos::delay_ms(1000);
                    }
                }
            }
        });
    }
    // Thread broadcaster (writer)
    {
        let clients_write = Arc::clone(&clients);
        thread::spawn(move || {
            while let Ok(line) = rx.recv() {
                if let Ok(mut vec) = clients_write.lock() {
                    vec.retain_mut(|stream| {
                        if writeln!(stream, "{}", line).is_err() {
                            warn!("TCP write to client failed: drop client");
                            false
                        } else {
                            true
                        }
                    });
                }
            }
        });
    }
    tx
}
// ===== Relay helper =============
#[inline(always)]
fn set_relay(
    relay: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio5, Output>,
    on: bool,
    active_low: bool,
) -> anyhow::Result<()> {
    if active_low {
        if on { relay.set_low()?; } else { relay.set_high()?; }
    } else {
        if on { relay.set_high()?; } else { relay.set_low()?; }
    }
    Ok(())
}
// Helper: satu siklus baca -> publish -> tulis Influx -> kirim ke TCP server
fn do_sensor_io(
    uart: &UartDriver<'_>,
    de_pin: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21, Output>,
    fc_use: u8, start_use: u16, qty_use: u16,
    mqtt: &SimpleMqttClient,
    topic_tele: &str,
    tcp_tx: &mpsc::Sender<String>,
```

```
) -> Result<(f32, f32)> {
    let (t, h) = read_sht20_with_map(uart, de_pin, fc_use, start_use, qty_use)?;
    let ts_ms = unsafe { sys::esp_timer_get_time() } / 1000;
    let t_rounded = (t * 10.0).round() / 10.0;
    let h_rounded = (h * 10.0).round() / 10.0;
    let payload = json!({
        "sensor": "sht20",
        "temperature_c": t_rounded,
        "humidity_pct": h_rounded,
        "ts_ms": ts_ms
    }).to_string();
    // 1) log ke stdout
    println!("{}", payload);
    // 2) publish ke ThingsBoard via MQTT
    if let Err(e) = mqtt.publish(topic_tele, &payload) {
        error!("MQTT publish error: {e:?}");
    }
    // 3) tulis ke Influx (HTTP)
    let lp = influx_line("sht20", TB_CLIENT_ID, t_rounded, h_rounded);
    if let Err(e) = influx_write(&lp) {
        warn!("Influx write failed: {e}");
    }
    // 4) kirim ke semua klien TCP yang terhubung
    if let Err(e) = tcp_tx.send(payload) {
        warn!("TCP channel send failed: {e}");
    }
    Ok((t, h))
}
// ===== main ==
fn main() -> Result<()> {
    // ESP-IDF init
    sys::link_patches();
    EspLogger::initialize_default();
    info!("Modbus RS485 + ThingsBoard MQTT + InfluxDB + Servo + TCP Server + Relay");
    // Peripherals & services
    let peripherals = Peripherals::take().context("Peripherals::take")?;
    let pins = peripherals.pins;
    let sys_loop = EspSystemEventLoop::take().context("eventloop")?;
    let nvs = EspDefaultNvsPartition::take().context("nvs")?;
    // Wi-Fi via BlockingWifi
    let mut wifi = BlockingWifi::wrap(
        EspWifi::new(peripherals.modem, sys_loop.clone(), Some(nvs))?,
        sys_loop,
    )?;
    connect_wifi(&mut wifi)?;
    // MQTT ThingsBoard (Basic)
    let mqtt = SimpleMqttClient::new(TB_MQTT_URL, TB_USERNAME, TB_PASSWORD,
        TB_CLIENT_ID)?;
    info!("MQTT connected to {}", TB_MQTT_URL);
    // === Start TCP Server (listener) ===
    let tcp_tx = start_tcp_server();
    info!("TCP server spawned at {}:{}", TCP_LISTEN_ADDR, TCP_LISTEN_PORT);
    // UART0 + RS485 (GPIO43/44, DE: GPIO21)
    let tx = pins.gpio43; // U0TXD
    let rx = pins.gpio44; // U0RXD
    let de = pins.gpio21;
    let cfg = UartConfig::new().baudrate(Hertz(BAUD));
    let uart = UartDriver::new(peripherals.uart0, tx, rx, None::<AnyIOPin>,
        None::<AnyIOPin>, &cfg)
        .context("UartDriver::new")?;
    let mut de_pin = PinDriver::output(de).context("PinDriver::output (DE)")?;
    de_pin.set_low()?; // default RX
    info!("UART0 ready (TX=GPIO43, RX=GPIO44, DE=GPIO21), {} bps", BAUD);
    // ===== Servo init (LEDC 50 Hz, 14-bit) =====
    let ledc = peripherals.ledc;
    let mut servo_timer = LedcTimerDriver::new(
        ledc.timer0,
        &TimerConfig {
            frequency: Hertz(50),
            resolution: Resolution::Bits14,
            ..Default::default()
        },
    )?;
    let servo_channel = LedcDriver::new(ledc.channel0, &mut servo_timer, pins.gpio18)?;
    let mut servo = Servo::new(servo_channel)?;
    let mut servo_pos = ServoPos::P90; // posisi awal 90deg
    // === Relay init (GPIO5) ===
    let mut relay = PinDriver::output(pins.gpio5).context("PinDriver::output (RELAY GPIO5)")?;
    // Pastikan OFF saat mulai
    if RELAY_GPIO_IS_LOW_ACTIVE { relay.set_high()?; } else { relay.set_low()?; }
    info!("Relay siap di GPIO5 (aktif-{}).", if RELAY_GPIO_IS_LOW_ACTIVE { "LOW" } else { "HIGH" });
    // Tanda waktu siklus reset servo (ms)
    let mut next_reset_ms: u64 = unsafe { (sys::esp_timer_get_time() as u64) / 1000 } + 20_000;
    // Probe mapping registri SHT20 (opsional)
    let (mut fc_use, mut start_use, mut qty_use) = (0x04u8, 0x0000u16, 2u16);
    if let Some((fc, start, qty)) = probe_map(&uart, &mut de_pin) {
        (fc_use, start_use, qty_use) = (fc, start, qty);
        info!("Using map: fc=0x{:02X}, start=0x{:04X}, qty={}", fc_use, start_use, qty_use);
    } else {
        warn!("Probe failed. Fallback map: fc=0x{:02X}, start=0x{:04X}, qty={}", fc_use,
            start_use, qty_use);
    }
    // Loop utama
    let topic_tele = "v1/devices/me/telemetry";
    loop {
        let now_ms: u64 = unsafe { (sys::esp_timer_get_time() as u64) / 1000 };
        if now_ms >= next_reset_ms {
            // === Reset siklus 20 detik -> kembali ke 90deg ===
            if servo_pos != ServoPos::P90 {
```

```
                if let Err(e) = servo.set_90() {
                    error!("Servo reset 90deg error: {e:?}");
                } else {
                    info!("Reset siklus 20s: Servo -> 90deg");
                    servo_pos = ServoPos::P90;
                }
            } else {
                info!("Reset siklus 20s: Servo sudah di 90deg");
            }
            // Baca ulang sensor & kirim data
            match do_sensor_io(&uart, &mut de_pin, fc_use, start_use, qty_use, &mqtt, topic_tele,
                &tcp_tx) {
                Ok((t, h)) => {
                    // === Relay logic: ON jika RH > 60%, selain itu OFF ===
                    let want_on = h > RH_ON_THRESHOLD;
                    set_relay(&mut relay, want_on, RELAY_GPIO_IS_LOW_ACTIVE)?;
                    let lvl = relay.is_set_high();
                    info!("Relay {} (RH={:.1}%) | GPIO5 level={}", if want_on { "ON" } else { "OFF" }, h,
                        if lvl { "HIGH" } else { "LOW" });
                    // Jeda 5 detik setelah mendapat data
                    FreeRtos::delay_ms(5_000);
                    // Aturan servo (setelah reset)
                    if t < 25.0 {
                        if servo_pos != ServoPos::P180 {
                            if let Err(e) = servo.set_180() { error!("Servo set 180deg error: {e:?}"); }
                            else { info!("Servo -> 180deg (T={:.1}degC) setelah reset", t); servo_pos = ServoPos::P180
    ; }
                        }
                    } else if t > 25.0 {
                        if servo_pos != ServoPos::P0 {
                            if let Err(e) = servo.set_0() { error!("Servo set 0deg error: {e:?}"); }
                            else { info!("Servo -> 0deg (T={:.1}degC) setelah reset", t); servo_pos = ServoPos::P0; }
                        }
                    } else {
                        info!("T=25.0degC persis -> Servo tetap di {:?}", servo_pos);
                    }
                }
                Err(e) => error!("Modbus read error (after 20s reset): {e:?}"),
            }
            next_reset_ms = now_ms + 20_000;
        } else {
            // === Siklus normal: baca sensor & kirim data ===
            match do_sensor_io(&uart, &mut de_pin, fc_use, start_use, qty_use, &mqtt, topic_tele,
                &tcp_tx) {
                Ok((t, h)) => {
                    // === Relay logic: ON jika RH > 60%, selain itu OFF ===
                    let want_on = h > RH_ON_THRESHOLD;
                    set_relay(&mut relay, want_on, RELAY_GPIO_IS_LOW_ACTIVE)?;
                    let lvl = relay.is_set_high();
                    info!("Relay {} (RH={:.1}%) | GPIO5 level={}", if want_on { "ON" } else { "OFF" }, h,
                        if lvl { "HIGH" } else { "LOW" });
                    // Jeda 5 detik setelah mendapat data
                    FreeRtos::delay_ms(5_000);
                    // Aturan servo (normal)
                    if t < 33.5 {
                        if servo_pos != ServoPos::P180 {
                            if let Err(e) = servo.set_180() { error!("Servo set 180deg error: {e:?}"); }
                            else { info!("Servo -> 180deg (T={:.1}degC)", t); servo_pos = ServoPos::P180; }
                        }
                    } else if t > 33.5 {
                        if servo_pos != ServoPos::P0 {
                            if let Err(e) = servo.set_0() { error!("Servo set 0deg error: {e:?}"); }
                            else { info!("Servo -> 0deg (T={:.1}degC)", t); servo_pos = ServoPos::P0; }
                        }
                    } else {
                        info!("T=33.5degC persis -> Servo tetap di {:?}", servo_pos);
                    }
                }
                Err(e) => error!("Modbus read error: {e:?}"),
            }
        }
        // Delay kecil agar loop tidak terlalu ketat
        FreeRtos::delay_ms(1000);
    }
}
```

Listing 1: main.rs

3.3.2 Cargo.toml

```
[package]
name = "skt13b" # atau nama proyekmu
version = "0.1.0"
edition = "2021"
resolver = "2"
rust-version = "1.77"

[[bin]]
```

```toml
name = "skt13b"
harness = false

[features]
default = []
experimental = ["esp-idf-svc/experimental"]

[dependencies]
log = "0.4"
anyhow = "1"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
esp-idf-svc = "=0.51.0"
esp-idf-sys = { version = "=0.36.1", features = ["binstart"] }
heapless = "0.8"

[build-dependencies]
embuild = "0.33"
```

Listing 2: Cargo.toml

3.3.3 ESP32S3 to InfluxDB

```python
#!/usr/bin/env python3
import sys, os, json, ssl, socket, urllib.request, urllib.parse

# ==== KONFIG INFLUXDB v2 ====
INFLUX_URL = os.getenv("INFLUX_URL", "http://localhost:8086").rstrip("/")
INFLUX_TOKEN = os.getenv("INFLUX_TOKEN", "ZISIdQAqG9EPJK_mgaW5q0IfN15-M1npTE5-ouPpDiP5PnxHchCTgAFdE1g2YFa50uoRCd9U7LFVWGmV9lomgQ==")
INFLUX_ORG = os.getenv("INFLUX_ORG", "ITS")
INFLUX_BUCKET = os.getenv("INFLUX_BUCKET", "skt13esp")

# ==== Kunci JSON yang dikirim ESP ====
MEASUREMENT = os.getenv("MEASUREMENT", "sht20")
FIELD_TEMP = os.getenv("FIELD_TEMP", "esp32s3 Temperature")
FIELD_HUM = os.getenv("FIELD_HUM", "esp32s3 Humidity")
FIELD_SENSOR = os.getenv("FIELD_SENSOR", "sensor")
TS_FIELD = os.getenv("TS_FIELD", "ts_ms") # epoch ms

def log(msg): print(msg, flush=True)

def _esc_tag(v: str) -> str:
    return v.replace(",", r"\,_").replace(" ", r"\_").replace("=", r"\_=")

def build_line_protocol(sensor: str, t_c: float, h_pct: float, ts_ms: int = None):
    tagpart = f"sensor={_esc_tag(sensor)}" if sensor else ""
    fieldpart = f"{FIELD_TEMP}={float(t_c)},{FIELD_HUM}={float(h_pct)}"
    if ts_ms is None:
        return f"{MEASUREMENT},{tagpart} {fieldpart}", "s"
    else:
        return f"{MEASUREMENT},{tagpart} {fieldpart} {int(ts_ms)}", "ms"

```

```
def influx_write(lines: str, precision: str):
    if not INFLUX_TOKEN: raise RuntimeError("INFLUX_TOKEN kosong")
    url = f"{INFLUX_URL}/api/v2/write?" + urllib.parse.urlencode({
        "org": INFLUX_ORG, "bucket": INFLUX_BUCKET, "precision":
    precision
    })
    req = urllib.request.Request(url, data=lines.encode("utf-8"),
    method="POST",
        headers={
            "Authorization": f"Token {INFLUX_TOKEN}",
            "Content-Type": "text/plain; charset=utf-8",
            "Accept": "application/json",
        }
    )
    ctx = ssl.create_default_context() if url.startswith("https://")
    else None
    with urllib.request.urlopen(req, context=ctx, timeout=15) as resp:
        if resp.status not in (200, 204):
            raise RuntimeError(f"HTTP {resp.status} {resp.read().decode
    ('utf-8', 'ignore')}")

def handle_line(line: str):
    obj = json.loads(line)
    if FIELD_TEMP not in obj or FIELD_HUM not in obj:
        raise ValueError(f"JSON harus punya {FIELD_TEMP} dan {FIELD_HUM
    }")
    sensor = str(obj.get(FIELD_SENSOR, "")).strip()
    t_c = float(obj[FIELD_TEMP])
    h_pct = float(obj[FIELD_HUM])
    ts_ms = obj.get(TS_FIELD)
    # Perbaikan penting: abaikan ts_ms yang bukan epoch ms wajar
    # Epoch ms yang wajar (tahun 2017+)
    if isinstance(ts_ms, (int, float)) and int(ts_ms) >= 1
    _500_000_000_000:
        ts_ms = int(ts_ms)
    else:
        ts_ms = None # biar Influx pakai timestamp "now" dari server
    lp, prec = build_line_protocol(sensor, t_c, h_pct, ts_ms)
    influx_write(lp, precision=prec)
    print(f"[OK] {sensor or ''} T={t_c:.3f}deg C RH={h_pct:.2f}% ts={'
    now' if ts_ms is None else ts_ms}", flush=True)

def main():
    if len(sys.argv) < 3:
        print("usage: influx_tcp_client.py <ESP_IP> <PORT>", file=sys.
    stderr)
        sys.exit(1)
    host = sys.argv[1]
    port = int(sys.argv[2])
    log(f"[CLIENT] connecting to {host}:{port} ...")
    # auto-reconnect sederhana
    while True:
        try:
            with socket.create_connection((host, port), timeout=10) as
    sock:
                log("[CLIENT] connected")
                f = sock.makefile(mode="r", encoding="utf-8", newline="
    \n")
```

```
                for line in f:
                    line = line.strip()
                    if not line:
                        continue
                    try:
                        handle_line(line)
                    except Exception as e:
                        log(f"[ERR] {e} | {line}")
        except Exception as e:
            log(f"[CLIENT] connect/read error: {e} (retry in 2s)")
            try:
                import time; time.sleep(2)
            except KeyboardInterrupt:
                break

if __name__ == "__main__":
    main()
```

Listing 3: ESP32S3 to InfluxDB (python)

3.3.4 DWSIM

```
import xml.etree.ElementTree as ET
import os
import time
import threading
import customtkinter as ctk
from tkinter import filedialog, messagebox
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

#--- KONFIGURASI INFLUXDB CLOUD ---
INFLUXDB_URL = "https://us-east-1-1.aws.cloud2.influxdata.com"
INFLUXDB_TOKEN = "4iveQHIkSl7m-
   EArzJZKxVLXB5G9FP21VTcbujdk_pfm7NHBGel2DZDECdLt-
   LLNtTMvQXFI1qlx2clomBSeOw=="
INFLUXDB_ORG = "skt"
INFLUXDB_BUCKET = "skt1"

running = False # Flag loop pengiriman

#--- FUNGSI LOGIKA ---
def extract_temperatures_from_xml(file_path):
    """ Membaca file XML DWSIM dan mengembalikan dictionary stream_name
   : temperature ( C ) """
    temperatures = {}
    try:
        tree = ET.parse(file_path)
        root = tree.getroot()
        # Pemetaan ID stream ke tag
        stream_tags = {}
        for graphic_object in root.findall(".//GraphicObject[ObjectType
   ='MaterialStream']"):
            stream_id = graphic_object.find('Name').text
            stream_tag = graphic_object.find('Tag').text
```

```
            if stream_id and stream_tag:
                stream_tags[stream_id] = stream_tag
        # Ambil temperatur
        for sim_object in root.findall(".//SimulationObject[Type='DWSIM.Thermodynamics.Streams.MaterialStream']"):
            stream_id = sim_object.find('Name').text
            if stream_id in stream_tags:
                temp_kelvin_element = sim_object.find(".//Phase[ComponentName='Mixture']/Properties/temperature")
                if temp_kelvin_element is not None:
                    temp_kelvin = float(temp_kelvin_element.text)
                    temp_celsius = temp_kelvin - 273.15
                    stream_name = stream_tags[stream_id]
                    temperatures[stream_name] = temp_celsius
        return temperatures
    except Exception as e:
        messagebox.showerror("Error", f"Gagal membaca XML: {e}")
        return None

def send_to_influxdb(client, data):
    """ Mengirim data temperatur ke InfluxDB Cloud """
    try:
        write_api = client.write_api(write_options=SYNCHRONOUS)
        points = []
        for stream_name, temp_celsius in data.items():
            point = (
                Point("heat_exchanger_monitoring")
                .tag("stream_name", stream_name)
                .field("temperature_celsius", float(f"{temp_celsius:.2f}"))
            )
            points.append(point)
        write_api.write(bucket=INFLUXDB_BUCKET, org=INFLUXDB_ORG, record=points)
        return True
    except Exception as e:
        messagebox.showerror("Error", f"Gagal mengirim ke InfluxDB Cloud: {e}")
        return False

#--- FUNGSI UI ---
def browse_file():
    file_path = filedialog.askopenfilename(filetypes=[("XML Files", "*.xml")])
    if file_path:
        xml_path_entry.delete(0, ctk.END)
        xml_path_entry.insert(0, file_path)

def start_monitoring():
    global running
    running = True
    start_button.configure(state="disabled")
    stop_button.configure(state="normal")
    thread = threading.Thread(target=monitor_loop)
    thread.daemon = True
    thread.start()

def stop_monitoring():
```

```
    global running
    running = False
    start_button.configure(state="normal")
    stop_button.configure(state="disabled")

def monitor_loop():
    global running
    file_path = xml_path_entry.get().strip()
    interval = int(interval_entry.get())
    if not os.path.exists(file_path):
        messagebox.showerror("Error", "File XML tidak ditemukan.")
        return
    client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)
    while running:
        data = extract_temperatures_from_xml(file_path)
        if data:
            update_display(data)
            update_graph(data)
            success = send_to_influxdb(client, data)
            if success:
                status_label.configure(text=f"Data dikirim ke InfluxDB Cloud @ {time.strftime('%H:%M:%S')}", text_color="lightgreen")
            else:
                status_label.configure(text="Gagal membaca data dari XML.", text_color="red")
        for i in range(interval):
            if not running:
                break
            time.sleep(1)

def update_display(data):
    text_output.delete("1.0", ctk.END)
    for name, temp in data.items():
        text_output.insert(ctk.END, f"{name}: {temp:.2f}  C \n")

#--- GRAFIK ---
def update_graph(data):
    ax.clear()
    streams = list(data.keys())
    temps = list(data.values())
    colors = ["#5DADE2" if "Cold" in s else "#2874A6" if "Hot" in s else "#3498DB" for s in streams]
    ax.bar(streams, temps, color=colors)
    for i, v in enumerate(temps):
        ax.text(i, v + 0.5, f"{v:.1f} C ", ha='center', color='white', fontsize=10, fontweight='bold')
    ax.set_title("Grafik Temperatur Heat Exchanger", fontsize=13, color="white", fontweight="bold")
    ax.set_ylabel("Temperature ( C )", color="white")
    ax.set_ylim(min(temps) - 5, max(temps) + 10)
    ax.grid(True, linestyle="--", alpha=0.3)
    ax.tick_params(colors="white")
    canvas.draw()

#--- SETUP DASHBOARD ---
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")
```

```
136
137 app = ctk.CTk()
138 app.title("Project SKT Kelompok 5")
139 app.geometry("900x720")
140
141 # Judul Utama
142 title_label = ctk.CTkLabel(
143     app,
144     text=" Project SKT Kelompok 5 ",
145     font=("Segoe UI", 22, "bold"),
146     text_color="#5DADE2"
147 )
148 title_label.pack(pady=15)
149
150 #Frame utama
151 frame = ctk.CTkFrame(app, corner_radius=10, fg_color="#0d3b66")
152 frame.pack(padx=20, pady=10, fill="x")
153
154 # Path XML
155 ctk.CTkLabel(frame, text="Path File XML DWSIM:", text_color="white").
    pack(anchor="w")
156 xml_path_entry = ctk.CTkEntry(frame, width=500)
157 xml_path_entry.pack(side="left", padx=5, pady=5)
158 browse_btn = ctk.CTkButton(frame, text="Browse", command=browse_file,
    fg_color="#1f77b4")
159 browse_btn.pack(side="left", padx=5, pady=5)
160
161 # Interval
162 interval_frame = ctk.CTkFrame(app, fg_color="#0d3b66")
163 interval_frame.pack(pady=10)
164 ctk.CTkLabel(interval_frame, text="Interval Kirim (detik):", text_color
    ="white").pack(side="left")
165 interval_entry = ctk.CTkEntry(interval_frame, width=60)
166 interval_entry.insert(0, "60")
167 interval_entry.pack(side="left", padx=5)
168
169 # Tombol kontrol
170 btn_frame = ctk.CTkFrame(app, fg_color="#0d3b66")
171 btn_frame.pack(pady=10)
172 start_button = ctk.CTkButton(btn_frame, text="Start", fg_color="#1E88E5
    ", command=start_monitoring)
173 start_button.pack(side="left", padx=10)
174 stop_button = ctk.CTkButton(btn_frame, text="Stop", fg_color="#D32F2F",
     command=stop_monitoring, state="disabled")
175 stop_button.pack(side="left", padx=10)
176
177 # Output data
178 ctk.CTkLabel(app, text="Data Temperatur:", text_color="#AED6F1").pack(
    anchor="w", padx=20)
179 text_output = ctk.CTkTextbox(app, height=150)
180 text_output.pack(padx=20, pady=10, fill="x")
181
182 # Grafik
183 ctk.CTkLabel(app, text="Grafik Temperatur:", text_color="#AED6F1", font
    =("Segoe UI", 13, "bold")).pack(anchor="w", padx=20)
184 graph_frame = ctk.CTkFrame(app, corner_radius=10, fg_color="#0d3b66")
185 graph_frame.pack(padx=20, pady=10, fill="both", expand=True)
186 fig, ax = plt.subplots(figsize=(7, 4))
```

```
fig.patch.set_facecolor("#0d3b66")
ax.set_facecolor("#0d3b66")
ax.tick_params(colors="white")
ax.yaxis.label.set_color("white")
ax.xaxis.label.set_color("white")
ax.title.set_color("white")
canvas = FigureCanvasTkAgg(fig, master=graph_frame)
canvas.get_tk_widget().pack(fill="both", expand=True)

# Status
status_label = ctk.CTkLabel(app, text="Menunggu aksi...", text_color="#F7DC6F")
status_label.pack(pady=10)

# Jalankan
app.mainloop()
```

Listing 4: DWSIM (python)

3.3.5 InfluxDB DWSIM dan Kirim ke Thingsboard

```
#!/usr/bin/env python3
# InfluxDB v2 -> ThingsBoard (MQTT Basic), JSON-query mode + perbaikan range (start)
# Default tersambung ke demo.thingsboard.io (non-TLS, port 1883)
import os, time, json, csv, ssl, re
import urllib.request, urllib.parse
from datetime import datetime, timezone
from typing import List, Tuple, Optional
from urllib.parse import urlparse

# ===== Influx v2 =====
INFLUX_URL = os.getenv("INFLUX_URL", "http://localhost:8086").rstrip("/")
INFLUX_TOKEN = os.getenv("INFLUX_TOKEN", "CnwVt5c90vNVJskPQ9W9QPMge6wWZyX4Gfdj8L_Yo77wbJJDuYDISCYL51jioxcXxaFRaPp-tsZX8A==")
INFLUX_ORG = os.getenv("INFLUX_ORG", "ITS")
INFLUX_BUCKET = os.getenv("INFLUX_BUCKET", "skt13dwsim")
MEASUREMENT = os.getenv("MEASUREMENT", "water_temperature_c")
FIELD = os.getenv("FIELD", "value")
TAG_FILTERS = os.getenv("TAG_FILTERS", "stream=water_temp") # kosongkan jika tidak perlu
QUERY_LOOKBACK = os.getenv("QUERY_LOOKBACK", "1h")

# ===== ThingsBoard (MQTT Basic) =====
# Kamu bisa set salah satu:
# TB_MQTT_URL="mqtt://demo.thingsboard.io:1883"
# atau terpisah TB_MQTT_HOST/TB_MQTT_PORT
TB_MQTT_URL = os.getenv("TB_MQTT_URL", "")
TB_MQTT_HOST = os.getenv("TB_MQTT_HOST", "demo.thingsboard.io")
TB_MQTT_PORT = int(os.getenv("TB_MQTT_PORT", "1883")) # 1883 non-TLS; 8883 TLS
TB_CLIENT_ID = os.getenv("TB_CLIENT_ID", "esp32s3-eggdhis")
TB_USERNAME = os.getenv("TB_USERNAME", "SKTkel13")
TB_PASSWORD = os.getenv("TB_PASSWORD", "453621")
TB_TOPIC = os.getenv("TB_TOPIC", "v1/devices/me/telemetry")
```

```
TB_KEY = os.getenv("TB_KEY", "DWsim Temperature")
# Opsional: pakai TLS untuk MQTT jika perlu (mis. saat port 8883)
TB_MQTT_TLS = int(os.getenv("TB_MQTT_TLS", "0")) # 1 untuk TLS, default
    0 (non-TLS)
POLL_INTERVAL_S = int(os.getenv("POLL_INTERVAL_S", "10"))
STATE_FILE = os.getenv("STATE_FILE", os.path.join(os.path.dirname(
    __file__), ".influx_tb_mqtt_state.json"))
DEBUG = bool(int(os.getenv("DEBUG", "0")))

from paho.mqtt import client as mqtt

def _ensure_cfg():
    missing = []
    if not INFLUX_TOKEN: missing.append("INFLUX_TOKEN")
    if not INFLUX_ORG: missing.append("INFLUX_ORG")
    if not INFLUX_BUCKET: missing.append("INFLUX_BUCKET")
    if missing:
        raise SystemExit(f"Wajib set: {', '.join(missing)}")

def _load_state() -> dict:
    try:
        with open(STATE_FILE, "r") as f:
            return json.load(f)
    except Exception:
        return {}

def _save_state(state: dict):
    try:
        with open(STATE_FILE, "w") as f:
            json.dump(state, f)
    except Exception as e:
        print(f"[WARN] save state: {e}")

def _is_duration(s: str) -> bool:
    """ Contoh valid: -1h30m, -2d, -15s, -1w """
    s = s.strip().lower()
    return bool(re.fullmatch(r"-\d+[smhdw]", s))

def _range_start_clause(start_expr: str) -> str:
    s = start_expr.strip()
    if _is_duration(s):
        return s # tanpa kutip untuk durasi relatif
    # asumsikan RFC3339 absolut
    return f'time(v: "{s}")'

def _build_flux(start_rfc3339_or_duration: str) -> str:
    filters = [
        f'r["_measurement"] == "{MEASUREMENT}"',
        f'r["_field"] == "{FIELD}"',
    ]
    if TAG_FILTERS:
        for pair in TAG_FILTERS.split(","):
            if "=" in pair:
                k, v = pair.split("=", 1)
                k, v = k.strip(), v.strip()
                if k:
                    filters.append(f'r["{k}"] == "{v}"')
    flt = " and ".join(filters)
```

```
    start_clause = _range_start_clause(start_rfc3339_or_duration)
    flux = f"""from(bucket: "{INFLUX_BUCKET}")
    |> range(start: {start_clause})
    |> filter(fn: (r) => {flt})
    |> keep(columns: ["_time", "_value"])
    |> sort(columns: ["_time"], desc: false)"""
    return flux

def _http_post(url: str, headers: dict, data: bytes, timeout: int = 25):
    req = urllib.request.Request(url, data=data, headers=headers, method="POST")
    ctx = None
    if url.startswith("https://"):
        ctx = ssl.create_default_context()
    try:
        with urllib.request.urlopen(req, context=ctx, timeout=timeout) as resp:
            return resp.status, resp.read()
    except urllib.error.HTTPError as e:
        return e.code, e.read()
    except Exception as e:
        raise RuntimeError(f"HTTP POST error {url}: {e}")

def query_influx_since(start_expr: str) -> List[Tuple[int, float]]:
    """Return list (ts_ms, value)"""
    flux = _build_flux(start_expr)
    if DEBUG:
        print("\n=== FLUX ===\n")
        print(flux)
        print("\n============\n")
    url = f"{INFLUX_URL}/api/v2/query?org={urllib.parse.quote(INFLUX_ORG)}"
    body = json.dumps({"query": flux, "type": "flux"}).encode("utf-8")
    headers = {
        "Authorization": f"Token {INFLUX_TOKEN}",
        "Content-Type": "application/json",
        "Accept": "application/csv"
    }
    status, resp = _http_post(url, headers, body)
    if status not in (200, 204):
        text = resp.decode("utf-8", "ignore")
        raise RuntimeError(f"Query Influx error HTTP {status}: {text}")

    rows: List[Tuple[int, float]] = []
    text = resp.decode("utf-8", "replace")
    reader = csv.DictReader(text.splitlines())
    for r in reader:
        if "_time" in r and "_value" in r:
            try:
                dt = datetime.fromisoformat(r["_time"].replace("Z", "+00:00"))
                ts_ms = int(dt.timestamp() * 1000)
                val = float(r["_value"])
                rows.append((ts_ms, val))
            except Exception:
                continue
    return rows
```

```
140
141 class TBMqtt:
142     def __init__(self, host: str, port: int, use_tls: bool):
143         # clean_session dipertahankan untuk kompatibilitas Paho < 2.0
144         self.client = mqtt.Client(client_id=TB_CLIENT_ID, clean_session
    =True)
145         self.client.username_pw_set(TB_USERNAME, TB_PASSWORD)
146         self.client.on_connect = self.on_connect
147         self.client.on_disconnect = self.on_disconnect
148         self.connected = False
149         self.host = host
150         self.port = port
151         self.use_tls = use_tls
152         if self.use_tls:
153             # TLS default (cert sistem). Untuk custom CA: tls_set(
    ca_certs="path")
154             self.client.tls_set()
155             # Opsi verifikasi sertifikat bisa disesuaikan bila perlu:
156             # self.client.tls_insecure_set(True)
157
158     def on_connect(self, client, userdata, flags, rc):
159         self.connected = (rc == 0)
160         print(f"[MQTT] Connected" if self.connected else f"[MQTT]
    Connect failed rc={rc}")
161
162     def on_disconnect(self, client, userdata, rc):
163         self.connected = False
164         print(f"[MQTT] Disconnected rc={rc}")
165
166     def connect(self):
167         self.client.connect(self.host, self.port, keepalive=60)
168         self.client.loop_start()
169         for _ in range(50):
170             if self.connected:
171                 return
172             time.sleep(0.1)
173         raise RuntimeError("MQTT tidak tersambung")
174
175     def ensure_connected(self):
176         if not self.connected:
177             try:
178                 self.client.reconnect()
179             except Exception:
180                 self.connect()
181
182     def publish(self, ts_ms: int, key: str, value: float):
183         self.ensure_connected()
184         payload = json.dumps({"ts": ts_ms, "values": {key: value}},
    separators=(",", ":"))
185         r = self.client.publish(TB_TOPIC, payload, qos=1, retain=False)
186         if r.rc != mqtt.MQTT_ERR_SUCCESS:
187             raise RuntimeError(f"MQTT publish rc={r.rc}")
188
189 def _resolve_mqtt_host_port_tls() -> Tuple[str, int, bool]:
190     """
191     Tentukan host, port, dan TLS berdasarkan:
192     1) TB_MQTT_URL (jika diset), contoh: mqtt://host:1883 atau mqtts://
    host:8883
```

```
    2) TB_MQTT_HOST/TB_MQTT_PORT
    3) TB_MQTT_TLS (0/1)
    """
    host = TB_MQTT_HOST
    port = TB_MQTT_PORT
    use_tls = bool(TB_MQTT_TLS)
    if TB_MQTT_URL:
        u = urlparse(TB_MQTT_URL)
        if not u.hostname:
            raise SystemExit(f"TB_MQTT_URL tidak valid: {TB_MQTT_URL}")
        host = u.hostname
        if u.port:
            port = int(u.port)
        # scheme mqtts => TLS; mqtt => non-TLS
        if u.scheme.lower() == "mqtts":
            use_tls = True
        elif u.scheme.lower() == "mqtt":
            # hanya ubah jika belum dipaksa TLS via env
            if TB_MQTT_TLS == 0:
                use_tls = False
    # Heuristik: jika port 8883 dan TLS belum true, aktifkan TLS
    if port == 8883 and not use_tls:
        use_tls = True
    return host, port, use_tls

def main():
    _ensure_cfg()
    host, port, use_tls = _resolve_mqtt_host_port_tls()
    print(f"[START] Influx -> TB MQTT | bucket='{INFLUX_BUCKET}' meas
    ='{MEASUREMENT}' field='{FIELD}' key='{TB_KEY}' interval={
    POLL_INTERVAL_S}s")
    print(f"[MQTT] host={host}:{port} client_id={TB_CLIENT_ID} user={
    TB_USERNAME} tls={'on' if use_tls else 'off'}")
    state = _load_state()
    start_ts = state.get("last_ts_rfc3339") or f"-{QUERY_LOOKBACK}"
    print(f"[INIT] Mulai query dari: {start_ts}")
    mqttc = TBMqtt(host, port, use_tls)
    mqttc.connect()
    while True:
        try:
            rows = query_influx_since(start_ts)
            if rows:
                sent = 0
                max_ts: Optional[int] = None
                for ts_ms, val in rows:
                    mqttc.publish(ts_ms, TB_KEY, val)
                    sent += 1
                    if max_ts is None or ts_ms > max_ts:
                        max_ts = ts_ms
                print(f"[OK] MQTT kirim {sent} titik. last_ts={max_ts}"
    )
                if max_ts is not None:
                    dt = datetime.fromtimestamp(max_ts / 1000, tz=
    timezone.utc)
                    start_ts = dt.isoformat()
                    state["last_ts_rfc3339"] = start_ts
                    _save_state(state)
            else:
```

```
                print("[INFO] Tidak ada data baru.")
        except Exception as e:
            print(f"[ERR] Loop error: {e}")
        time.sleep(POLL_INTERVAL_S)

if __name__ == "__main__":
    main()
```

Listing 5: InfluxDB DWSIM dan Kirim ke Thingsboard (python)

4 Implementasi dan Hasil

4.1 Konfigurasi DWSIM

4.2 Pembahasan

Proyek sistem kontrol terdistribusi ini berfokus pada pemantauan dan pengendalian parameter lingkungan, khususnya temperatur dan kelembaban, dengan mengintegrasikan data sensor fisik dan data simulasi proses. Implementasi sistem ini melibatkan beberapa komponen teknologi kunci yang saling berinteraksi, mulai dari akuisisi data di tingkat edge hingga visualisasi di cloud.

Akuisisi Data Sensor: Komponen sensor yang digunakan adalah SHT20, yang berfungsi mengukur temperatur dan kelembaban relatif. Sensor ini dihubungkan melalui antarmuka RS485 dan berkomunikasi menggunakan protokol Modbus RTU. Penggunaan RS485 dipilih karena kemampuannya dalam transmisi data jarak jauh yang andal, cocok untuk lingkungan industri. Mikrokontroler ESP32-S3 berperan sebagai edge gateway, bertugas membaca data dari sensor SHT20 via Modbus RTU. Laporan pertama (Kelompok 13) secara spesifik menyebutkan penggunaan bahasa pemrograman Rust pada ESP32-S3, yang menawarkan keunggulan dalam hal keamanan memori dan efisiensi. Proses pembacaan data Modbus, seperti terlihat pada hasil pengujian terminal (Gambar 5.1 dan 5.2), menunjukkan keberhasilan akuisisi data temperatur dan kelembaban secara periodik.

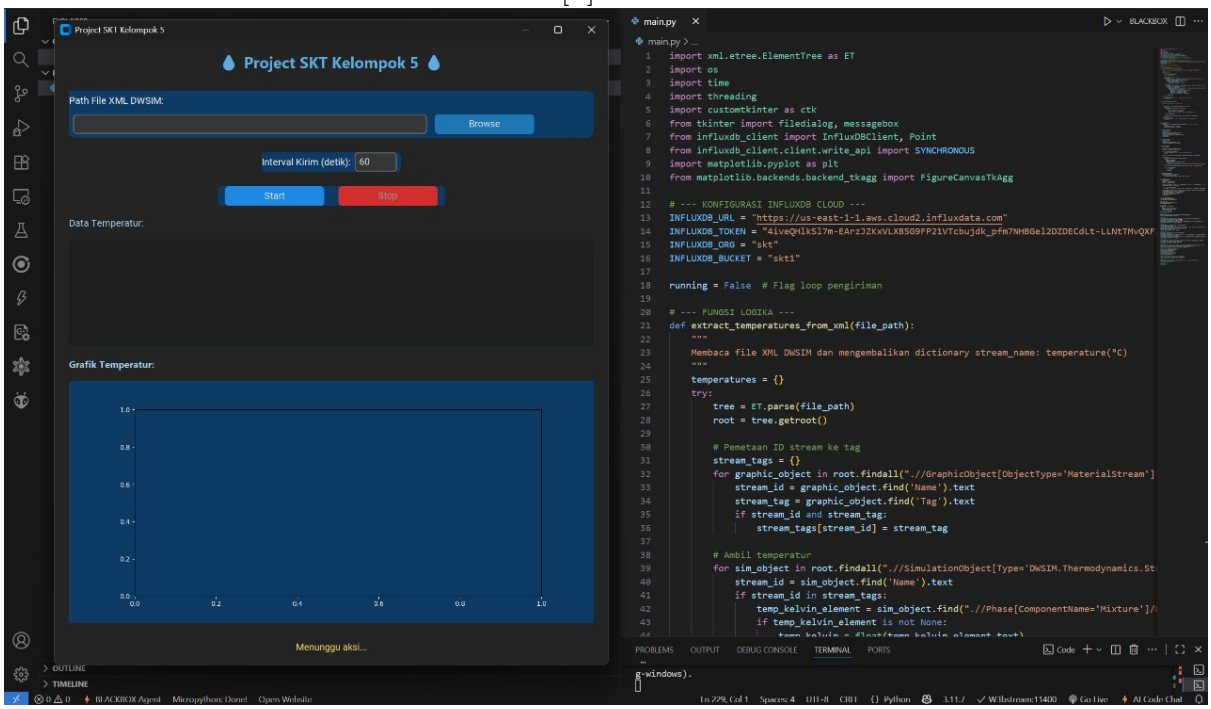
Salah satu tantangan yang dihadapi adalah memastikan komunikasi Modbus yang stabil, yang diatasi dengan probe mapping untuk menentukan function code dan register yang tepat, serta penyesuaian timing komunikasi.

Integrasi Data Simulasi DWSIM: Kedua proyek mengintegrasikan data dari DWSIM, sebuah simulator proses kimia open-source. DWSIM digunakan untuk memodelkan proses yang relevan (misalnya melibatkan aliran air atau pertukaran panas) dan mengekspor hasil simulasi, seperti temperatur aliran, ke dalam format XML. Skrip Python kemudian digunakan untuk mengurai (parse) file XML ini secara periodik dan mengekstrak parameter yang relevan (misalnya, temperatur aliran "water in"). Data hasil ekstraksi ini selanjutnya dikirim ke basis data InfluxDB. Hasil pengiriman data DWSIM ke InfluxDB terdokumentasi dalam tangkapan layar terminal (Gambar 5.3) dan antarmuka skrip Python (Laporan 2, Gambar 4.1). Integrasi ini memungkinkan perbandingan atau kombinasi data simulasi teoretis dengan data sensor empiris dalam satu platform. Tantangan sinkronisasi format dan timestamp antara data DWSIM dan data sensor nyata diatasi dengan pra-pemrosesan data simulasi sebelum disimpan ke InfluxDB.

Penyimpanan Data Time-Series (InfluxDB): InfluxDB dipilih sebagai basis data time-series untuk menyimpan data temperatur dan kelembaban dari sensor SHT, serta data temperatur dari simulasi DWSIM. InfluxDB dioptimalkan untuk menangani volume besar data yang diindeks berdasarkan waktu, memungkinkan penulisan (ingestion) dan kueri data yang cepat dan efisien. Data dari ESP32 (sensor SHT) dikirim ke InfluxDB, baik secara langsung melalui permintaan HTTP dari firmware Rust (Laporan 1) maupun melalui perantara skrip Python yang membaca data dari TCP server ESP32 (Laporan 1). Data dari DWSIM juga dikirim ke InfluxDB oleh skrip Python. Keberadaan data di InfluxDB dikonfirmasi melalui tangkapan layar Data Explorer (Gambar 5.5) dan dasbor InfluxDB (Laporan 2, Gambar 4.2). Penggunaan InfluxDB memfasilitasi penyimpanan historis data untuk analisis tren lebih lanjut.

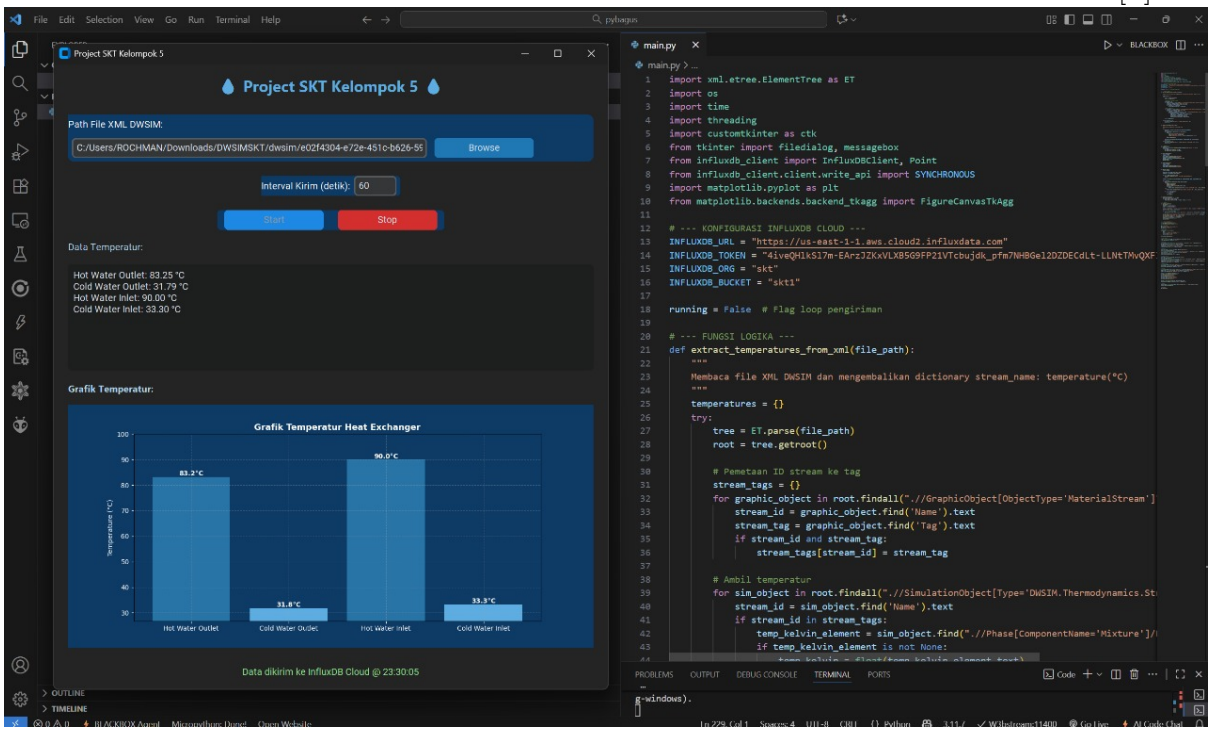
Integrasi Cloud dan Visualisasi (ThingsBoard): Platform IoT ThingsBoard digunak-

[b]0.48



Gambar 7: Konfigurasi DWSIM

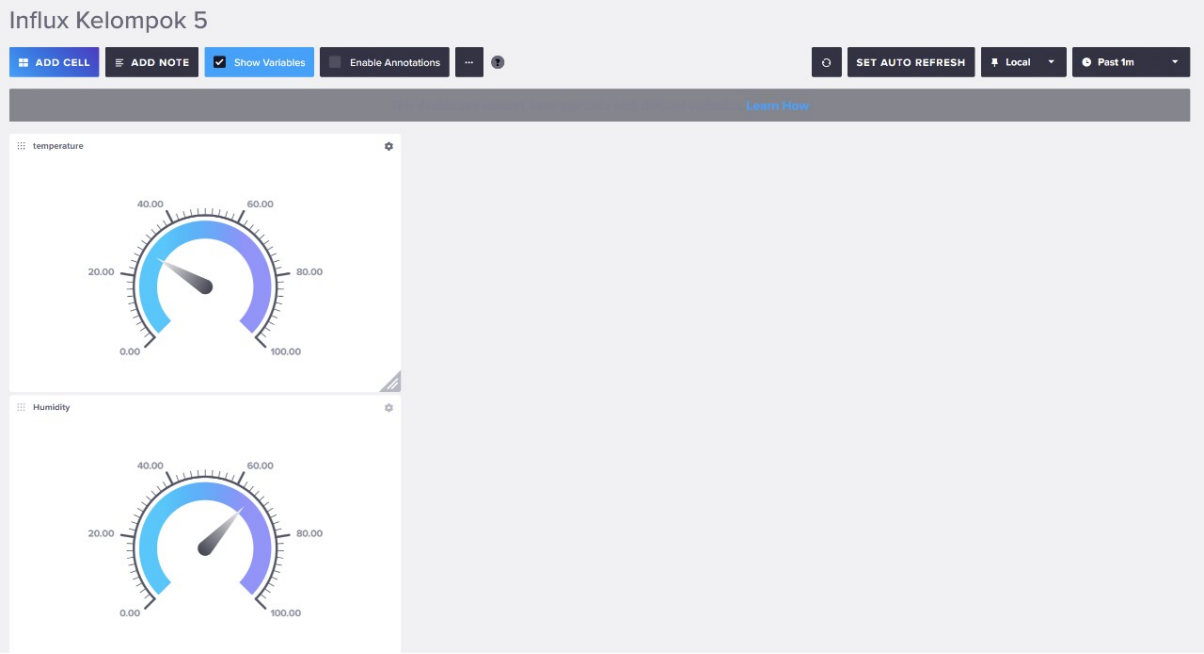
[b]0.48



Gambar 8: Eksekusi DWSIM

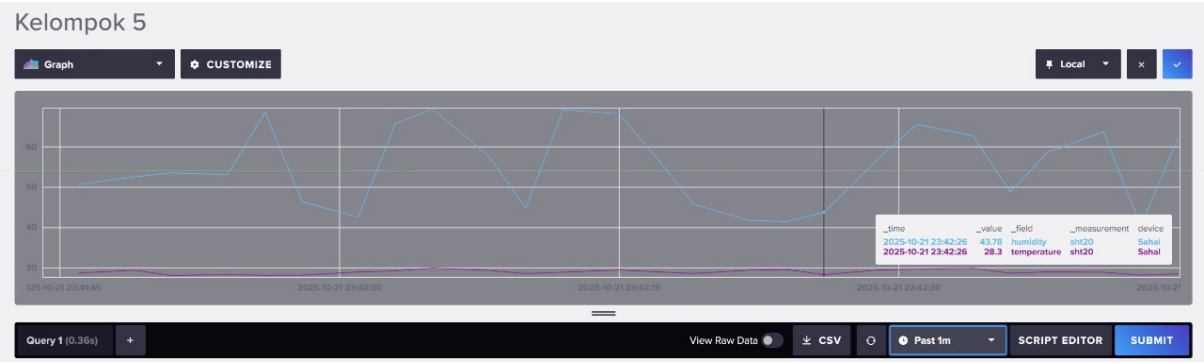
Gambar 9: Konfigurasi dan Eksekusi DWSIM

[b]0.48



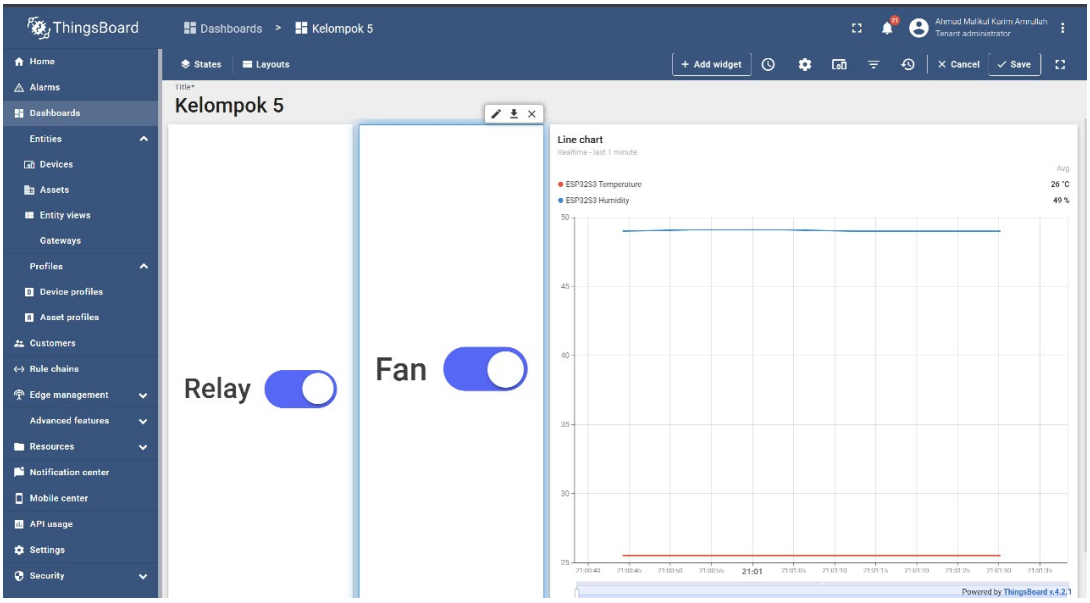
Gambar 10: Dashboard Awal InfluxDB

[b]0.48



Gambar 11: Dashboard Akhir InfluxDB

Gambar 12: Dashboard InfluxDB Kelompok 5



Gambar 13: Dashboard ThingsBoard Kelompok 5

an sebagai antarmuka visualisasi data berbasis cloud. Data dari sensor SHT dan/atau DWSIM (setelah disimpan di InfluxDB) dikirim ke ThingsBoard menggunakan protokol MQTT. MQTT, dengan model publish-subscribe-nya, cocok untuk transmisi data telemetri dari perangkat edge ke cloud secara efisien. Laporan 1 menunjukkan pengiriman data SHT dari ESP32 langsung ke ThingsBoard via MQTT dan juga skrip Python (Thingsboard.py) yang membaca data DWSIM dari InfluxDB lalu meneruskannya ke ThingsBoard via MQTT. Hasilnya ditampilkan pada dasbor ThingsBoard yang interaktif, menampilkan data temperatur dan kelembaban secara real-time dalam bentuk grafik dan gauge (Gambar 5.6 dan Laporan 2, Gambar 4.3). ThingsBoard memungkinkan pembuatan antarmuka HMI (Human-Machine Interface) yang dapat disesuaikan untuk pemantauan dan pengambilan keputusan. Salah satu kendala yang dihadapi adalah memastikan konfigurasi device token dan topik MQTT yang benar agar data muncul di dasbor.

Logika Kontrol (Laporan 1): Selain pemantauan, sistem pada Laporan 1 juga mengimplementasikan logika kontrol sederhana. Sebuah relay diaktifkan berdasarkan ambang batas kelembaban (RH $> 59\%$) untuk mengendalikan aktuator eksternal (misalnya, kipas). Selain itu, motor servo SG90 dikendalikan untuk bergerak ke posisi tertentu (0, 90, atau 180 derajat) berdasarkan ambang batas temperatur. Implementasi ini menunjukkan kemampuan sistem untuk tidak hanya memantau tetapi juga merespons kondisi lingkungan secara otomatis.

Secara keseluruhan, arsitektur sistem yang dibangun menunjukkan integrasi yang berhasil antara perangkat keras edge (ESP32-S3, sensor Modbus, aktuator), perangkat lunak simulasi (DWSIM), basis data time-series.

5 Kesimpulan dan Saran

5.1 Kesimpulan

1. Proyek ini berhasil mendemonstrasikan kelayakan integrasi berbagai komponen teknologi, meliputi akuisisi data sensor fisik (SHT20) melalui protokol Modbus RTU/RS485, pengumpulan data dari perangkat lunak simulasi proses (DWSIM), penyimpanan data secara persisten dalam basis data time-series (InfluxDB), dan visualisasi data secara real-time pada platform IoT cloud (ThingsBoard) menggunakan protokol MQTT.

2. Mikrokontroler ESP32-S3 terbukti efektif berfungsi sebagai edge gateway, mampu menangani komunikasi Modbus RTU, konektivitas Wi-Fi, pemrosesan data awal, dan komunikasi dengan cloud. Penggunaan bahasa Rust (pada Laporan 1) menunjukkan potensi untuk pengembangan firmware yang aman dan efisien, meskipun memerlukan kurva pembelajaran.

3. Integrasi data sensor empiris dengan data simulasi DWSIM memberikan nilai tambah, memungkinkan validasi model simulasi atau pengayaan data monitoring dengan parameter yang tidak terukur secara langsung.

4. Penggunaan InfluxDB sebagai data historian dan ThingsBoard untuk visualisasi dasbor interaktif, yang dihubungkan melalui MQTT, merupakan kombinasi yang solid dan umum digunakan dalam arsitektur IoT modern untuk pemantauan dan analisis data time-series.

5. Sistem yang dikembangkan ini menjadi prototipe yang valid untuk aplikasi pemantauan lingkungan industri, pertanian presisi (hidroponik), atau sistem otomasi gedung, di mana pemantauan suhu, kelembaban, dan parameter proses lainnya secara terdistribusi dan real-time sangat diperlukan. Implementasi logika kontrol aktuator (relay dan servo pada Laporan 1) semakin memperkuat potensi aplikasinya dalam sistem otomasi.

5.2 Saran

Penguatan kapabilitas di tingkat edge dapat dicapai dengan mengimplementasikan algoritma machine learning atau logika analisis yang lebih kompleks langsung pada mikrokontroler ESP32-S3. Hal ini memungkinkan deteksi anomali atau pengambilan keputusan kontrol secara lokal (edge intelligence), mengurangi latensi dan ketergantungan pada konektivitas cloud.

Sejalan dengan peningkatan kecerdasan di edge, aspek reliabilitas juga krusial ditingkatkan. Penambahan mekanisme data buffering pada ESP32-S3 akan memastikan kontinuitas data saat terjadi gangguan koneksi ke cloud, di mana data sensor disimpan sementara secara lokal dan dikirimkan kembali saat konektivitas pulih. Peningkatan ketahanan koneksi Wi-Fi dan MQTT melalui implementasi mekanisme watchdog dan reconnect otomatis yang lebih robust juga sangat disarankan.

Dari sisi implementasi praktis dan interaksi pengguna, optimalisasi konsumsi daya menjadi penting, terutama untuk penempatan di lokasi terpencil atau bertenaga baterai. Analisis mendalam mengenai pola konsumsi daya dan implementasi mode sleep (misalnya, deep sleep atau light sleep) pada ESP32-S3 dapat secara signifikan meningkatkan

efisiensi energi. Selain itu, pengembangan antarmuka pengguna lokal berbasis web server yang berjalan di ESP32-S3 dapat memberikan alternatif pemantauan dan konfigurasi yang tidak bergantung pada koneksi internet eksternal, meningkatkan aksesibilitas sistem dalam berbagai kondisi jaringan.

Terakhir, integrasi dengan DWSIM dapat diperdalam lebih lanjut. Perlu dieksplorasi kemungkinan interaksi dua arah, di mana data sensor real-time dari sistem fisik dapat diumpankan kembali ke DWSIM untuk memperbarui parameter simulasi secara dinamis. Sebaliknya, hasil keluaran dari simulasi DWSIM yang lebih kompleks dapat dimanfaatkan untuk menentukan atau menyesuaikan setpoint kontrol pada aktuator di sistem fisik secara otomatis. Dengan mengimplementasikan saran-saran ini, sistem yang telah dibangun dapat berevolusi menjadi solusi pemantauan dan kontrol terdistribusi yang lebih andal, fleksibel, efisien, dan cerdas.

Pustaka

[1] R. Roman et al. Comparative analysis of time series databases in the context of edge computing for low power sensor networks. *Sensors*, 20(13):3762, 2020. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC7302557/.

[2] A. Khelifati et al. Performance evaluation of modern time-series database technologies for the atlas operational monitoring data archiving service. *IEEE Transactions on Nuclear Science*, 2023. URL https://www.researchgate.net/publication/368729433_Performance_Evaluation_of_Modern_Time-series_Database_Technologies_for_the_ATLAS_Operational_Monitoring_Data_Archiving_Service.

[3] A. Khelifati et al. Tsm-bench: Benchmarking time series database systems for the atlas operational monitoring data archiving service. *In Proceedings of the VLDB Endowment*, volume 16, pages 3363-3376, 2023. URL https://www.vldb.org/pvldb/vol16/p3363-khelifati.pdf.

[4] X. Li, J. Zhang, and H. Chen. Scits: A benchmark for time-series databases in scientific experiments and industrial internet of things. *TimeStored Technical Report*, 2022. URL https://www.timestored.com/data/files/2022-06-08-scits-scientific-time-series-benchmark.pdf

[5] Bonil Shah, P. M. Jat, and Kalyan Sasidhar. Performance study of time series databases. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2022. URL https://www.researchgate.net/publication/363128579_Performance_Study_of_Time_Series_Databases.

[6] Titin Nurfadhila Sudirman. Perancangan dashboard dan query. *Jurnal Teknologi Informasi*, 2019. Institut Teknologi Nasional.