TUGAS MATA KULIAH

SISTEM KONTROL TERDISTRIBUSI

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

"Monitoring dan Kontrol Suhu Penyimpanan Stroberi Berbasis Node-RED dan ThingsBoard"

Disusun Oleh:

Cahyo Okto Risfian (2042231044)

Aireka Maulana Erawan (2042231047)

PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI

DEPARTEMEN TEKNIK INSTRUMENTASI

FAKULTAS VOKASI

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2025

Contents

BAB I PENDAHULUAN

1.1 Latar Belakang

Stroberi merupakan salah satu buah yang memiliki nilai ekonomi tinggi dan banyak diminati karena rasanya yang khas serta kandungan gizinya yang baik bagi kesehatan. Namun, stroberi termasuk buah yang **sangat mudah rusak** (*perishable fruit*) karena memiliki kadar air yang tinggi dan laju respirasi yang cepat setelah dipanen (Kader, 2002). Kondisi penyimpanan yang tidak sesuai, khususnya suhu yang terlalu tinggi, dapat mempercepat proses pembusukan, pertumbuhan mikroba, dan penurunan kualitas buah (Wills et al., 2016).

Untuk mempertahankan kesegaran stroberi, penyimpanan harus dilakukan pada suhu rendah, umumnya berkisar antara **0–5°C** dengan kelembapan relatif sekitar **90–95%** (FAO, 2019). Oleh karena itu, pengendalian suhu dan kelembapan menjadi aspek penting dalam sistem penyimpanan buah. Sistem konvensional yang masih dilakukan secara manual sering kali tidak mampu menjaga kondisi penyimpanan secara konsisten. Diperlukan teknologi yang mampu melakukan **monitoring dan kontrol suhu** secara otomatis dan **real-time**.

Perkembangan teknologi **Internet of Things (IoT)** memungkinkan pengawasan dan pengendalian kondisi lingkungan secara daring melalui koneksi internet (Ashton, 2009). Dalam konteks penyimpanan buah, IoT dapat digunakan untuk membaca data suhu dan kelembapan dari sensor, mengirimkannya ke server, serta menampilkan informasi tersebut melalui antarmuka pengguna. **Node-RED** merupakan platform *flow-based programming* yang memudahkan integrasi data dari berbagai perangkat IoT (RedHat, 2020), sementara **ThingsBoard** berperan sebagai platform *open-source* untuk visualisasi data dan manajemen perangkat (ThingsBoard, 2023).

Dengan mengintegrasikan Node-RED dan ThingsBoard, sistem monitoring suhu dapat dirancang untuk mengawasi kondisi penyimpanan stroberi secara *real-time* serta memberikan peringatan atau tindakan kontrol otomatis apabila suhu melebihi batas yang diizinkan. Melalui penelitian ini, dirancang sebuah Sistem Monitoring dan Kontrol Suhu Penyimpanan Stroberi Berbasis Node-RED dan ThingsBoard yang diharapkan dapat menjaga kualitas buah selama penyimpanan agar tetap segar dan memperpanjang umur simpannya.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana merancang sistem IoT untuk memantau suhu penyimpanan stroberi secara *real-time*?
2. Bagaimana mengintegrasikan Node-RED dan ThingsBoard untuk menampilkan data suhu secara kontinu?
3. Bagaimana sistem dapat memberikan kontrol otomatis ketika suhu penyimpanan melebihi batas optimum?

1.3 Tujuan Penelitian

1. Merancang dan mengimplementasikan sistem monitoring suhu penyimpanan stroberi berbasis IoT.
2. Mengintegrasikan Node-RED dan ThingsBoard sebagai platform pengolah dan visualisasi data suhu.
3. Menerapkan sistem kontrol otomatis untuk menjaga suhu penyimpanan stroberi dalam batas optimal.

1.4 Manfaat Penelitian

1. Memberikan solusi berbasis teknologi untuk menjaga kualitas buah stroberi selama penyimpanan.

2. Menjadi referensi dalam penerapan sistem IoT pada bidang pertanian dan pasca panen.

3. Membantu produsen atau distributor stroberi dalam memantau kondisi penyimpanan secara efisien.

BAB II TINJAUAN PUSTAKA

2.1 State of the Art

Berikut merupakan kajian pustaka dari beberapa penelitian sebelumnya yang berkaitan dengan sistem monitoring suhu dan kelembapan, serta penerapan IoT dalam pengendalian lingkungan penyimpanan buah.

No	**Referensi (Penulis, Tahun)**	**Fokus Penelitian**	**Metode / Implementasi dan Hasil Utama**
1	S. Ahmed et al., 2021	Sistem monitoring suhu dan kelembapan untuk penyimpanan buah	Sistem IoT *real-time* menggunakan sensor DHT22 dan Node-RED. Mampu memantau suhu dan kelembapan ruang penyimpanan secara *real-time* dan efisien.
2	R. Rahman et al., 2022	Pengaruh suhu dan kelembapan terhadap masa simpan buah stroberi	Eksperimen variasi suhu 5–20°C dengan kontrol kelembapan. Suhu optimal penyimpanan 10°C menjaga kesegaran stroberi hingga 10 hari.

3	H. Kim et al., 2020	Evaluasi performa sensor SHT20 untuk aplikasi penyimpanan buah	Pengujian sensor digital terhadap akurasi suhu dan RH. Sensor memiliki akurasi $\pm 0.3^{\circ}$C dan $\pm 2\%$ RH, cocok untuk sistem monitoring suhu penyimpanan.
4	J. Zhao et al., 2021	Implementasi Modbus RTU untuk komunikasi data pada sistem IoT	Integrasi sensor–mikrokontroler dengan protokol Modbus RTU. Komunikasi data stabil dan cepat.
5	L. Nguyen and D. Lee, 2021	Penggunaan InfluxDB dan Node-RED untuk sistem IoT	Implementasi database *time-series* dengan Node-RED dan ThingsBoard. Sistem merekam dan menampilkan data suhu–kelembapan dengan *latency* rendah.
6	M. Setiawan et al., 2023	Kontrol suhu otomatis untuk penyimpanan buah berbasis IoT	Pengendalian suhu otomatis menggunakan PID berbasis Node-RED. Suhu stabil $10\pm1^{\circ}$C dan menurunkan konsumsi energi hingga 12%.

2.2 Landasan Teori

2.2.1 Internet of Things (IoT)

Internet of Things (IoT) merupakan konsep yang memungkinkan objek fisik terhubung ke internet untuk saling bertukar data tanpa campur tangan manusia secara langsung. IoT bekerja dengan cara mengumpulkan data dari sensor, mengirimkannya melalui jaringan, dan memprosesnya menggunakan perangkat lunak atau platform tertentu.

2.2.2 Node-RED

Node-RED adalah platform pemrograman berbasis alur (*flow-based programming*) yang memudahkan integrasi antara perangkat keras, API, dan layanan daring. Node-RED menggunakan antarmuka visual berbasis blok sehingga pengguna dapat membuat alur data dengan cara menghubungkan *node* secara intuitif.

Node-RED biasanya dijalankan pada perangkat seperti Raspberry Pi atau komputer server lokal. Dalam sistem ini, Node-RED berfungsi sebagai pengolah data dari sensor suhu, mengirimkan data ke server, serta mengatur logika kontrol otomatis.

2.2.3 ThingsBoard

ThingsBoard merupakan platform IoT *open-source* yang berfungsi untuk mengelola perangkat, menyimpan data, dan menampilkan data melalui *dashboard* interaktif. Platform ini mendukung berbagai protokol komunikasi seperti MQTT, CoAP, dan HTTP.

Dalam penelitian ini, ThingsBoard digunakan untuk menampilkan data suhu dan kelembapan dari Node-RED secara *real-time*, serta menyediakan visualisasi grafik dan notifikasi.

2.2.4 Sensor SHT20

SHT20 adalah sensor digital yang digunakan untuk mengukur suhu dan kelembapan dengan presisi tinggi. Sensor ini bekerja menggunakan antarmuka I2C dan memiliki spesifikasi akurasi ±0.3°C untuk suhu dan ±2% RH untuk kelembapan.

Sensor ini banyak digunakan dalam aplikasi IoT karena konsumsi dayanya yang rendah serta kemudahan integrasinya dengan mikrokontroler seperti ESP32.

Figure II.1: Sensor SHT20 (Sumber: Datasheet Sensirion, 2023)

2.2.5 ESP32

ESP32 merupakan mikrokontroler yang memiliki konektivitas Wi-Fi dan Bluetooth bawaan, serta kemampuan pemrosesan yang tinggi. Mikrokontroler ini banyak digunakan dalam sistem IoT karena dapat menjalankan komunikasi MQTT, HTTP, serta integrasi langsung dengan Node-RED dan ThingsBoard.

2.2.6 InfluxDB

InfluxDB adalah basis data *time-series* yang dioptimalkan untuk penyimpanan dan pengambilan data berbasis waktu. Database ini sering digunakan dalam sistem monitoring untuk mencatat data sensor secara *real-time* dan efisien.

Dalam sistem ini, InfluxDB digunakan sebagai penyimpanan utama data suhu dan kelembapan yang dikirimkan dari Node-RED.

2.2.7 Modbus RTU

Modbus RTU adalah protokol komunikasi serial yang umum digunakan dalam sistem industri. Protokol ini digunakan untuk mentransfer data antar perangkat, seperti sensor dan PLC, melalui komunikasi RS-485. Format Modbus RTU cocok untuk sistem monitoring terdistribusi karena memiliki reliabilitas tinggi dan struktur data sederhana.

BAB III METODOLOGI PENELITIAN

3.1 Metodologi dan Arsitektur Sistem

Metodologi penelitian ini dirancang untuk mengembangkan sistem monitoring dan kontrol suhu penyimpanan stroberi berbasis **Internet of Things (IoT)**. Tujuan utama adalah menjaga suhu dan kelembapan ruang penyimpanan agar tetap stabil sesuai standar kualitas stroberi, yaitu suhu **0–10°C** dengan kelembapan **85–95%**.

Sistem ini menggabungkan sensor suhu dan kelembapan, mikrokontroler ESP32-S3, Node-RED, serta ThingsBoard dalam satu jaringan IoT. Data sensor dikirim ke Node-RED menggunakan protokol MQTT, kemudian diteruskan ke ThingsBoard untuk visualisasi dan analisis data. Node-RED juga memiliki fungsi kontrol otomatis yang mengatur pendingin atau kipas berdasarkan kondisi suhu aktual.

Diagram blok sistem menggambarkan hubungan antar komponen dalam sistem monitoring dan kontrol suhu penyimpanan stroberi. Sensor SHT20 digunakan untuk membaca suhu dan kelembapan, kemudian data dikirim melalui komunikasi Modbus RTU ke ESP32. Mikrokontroler ESP32 berfungsi sebagai *gateway* yang mengirimkan data ke Node-RED untuk diolah dan dikirimkan ke ThingsBoard.

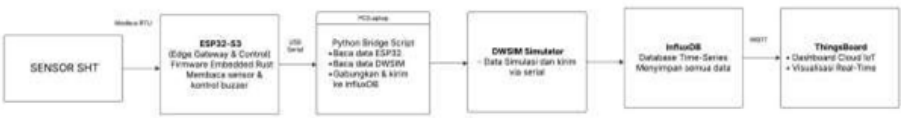

Figure III.1: Diagram blok sistem monitoring dan kontrol suhu penyimpanan stroberi

Sistem monitoring suhu dan kelembaban ini terdiri dari beberapa komponen utama seperti sensor, komunikasi data, penyimpanan dan visualisasi, serta integrasi DWSIM. Untuk tahapan dari sistem meliputi:

1. **Studi Literatur**: Pengumpulan referensi terkait sistem IoT, Node-RED, ThingsBoard, serta karakteristik penyimpanan buah stroberi.

2. **Perancangan Sistem**: Meliputi perancangan perangkat keras (*hardware*) dan perangkat lunak (*software*) yang mendukung proses monitoring dan kontrol otomatis.

3. **Implementasi Sistem**: Proses perakitan perangkat, pemrograman ESP32-S3, serta konfigurasi Node-RED dan ThingsBoard.

4. **Pengujian Sistem**: Pengujian dilakukan untuk memastikan sensor bekerja akurat, sistem komunikasi berjalan stabil, serta respon kontrol berfungsi sesuai kondisi suhu.

5. **Analisis dan Evaluasi**: Hasil pengujian dianalisis untuk mengetahui performa sistem dalam menjaga kondisi penyimpanan stroberi agar tetap optimal.

3.2 Perancangan Sistem Monitoring

Perancangan sistem ini bertujuan membangun sistem monitoring dan kontrol suhu penyimpanan stroberi berbasis IoT agar kualitas buah tetap terjaga. Sistem menggunakan ESP32-S3 sebagai pusat kendali yang terhubung dengan sensor suhu dan kelembapan (SHT20). Data yang terbaca dikirim melalui protokol MQTT ke Node-RED, kemudian diteruskan ke ThingsBoard untuk ditampilkan secara *real-time*. Apabila suhu penyimpanan melebihi batas aman (10°C), akan mengirimkan perintah ke ESP32-S3 untuk mengaktifkan pendingin atau kipas hingga suhu kembali normal. Sistem ini bekerja otomatis, akurat, dan berkelanjutan, serta memungkinkan pengguna memantau kondisi penyimpanan stroberi dari jarak jauh melalui *dashboard* ThingsBoard. Dengan rancangan ini, suhu penyimpanan stroberi dapat dijaga stabil antara **0–10°C** dan kelembapan **85–95%**, sehingga kesegaran buah tetap terpelihara tanpa memerlukan pengawasan manual.

3.2.1 Perancangan Hardware

Rancangan perangkat keras sistem ini terdiri dari sejumlah komponen utama yang bekerja secara terpadu untuk memonitor dan mengendalikan kondisi lingkungan di dalam *greenhouse* (meskipun konteksnya penyimpanan stroberi). Pengambilan data lingkungan dilakukan oleh bagian sensor, di mana sensor SHT20 secara spesifik bertugas mengukur suhu dan kelembaban udara. Semua data sensor tersebut kemudian diakuisisi dan diproses oleh mikrokontroler **ESP32-S3**, yang berfungsi sebagai pusat kendali. Selain mengolah data sensor, ESP32-S3 juga bertanggung jawab untuk mengirimkan data ke server melalui protokol MQTT, serta menerima dan menjalankan perintah dari *backend* untuk menggerakkan aktuator.

Bagian aktuator mencakup pompa air (diasumsikan untuk kontrol kelembaban, meskipun teks menyebut *greenhouse*) dan **kipas DC** (untuk kontrol suhu) yang akan menyala jika suhu lingkungan melebihi batas optimal. Berikut adalah alur *wiring* sistem kontrol suhu pada tanaman stroberi.

Perangkat keras yang digunakan terdiri atas beberapa komponen utama sebagai berikut:

1. **ESP32** sebagai mikrokontroler utama yang berfungsi membaca data dari sensor SHT20 dan mengirimkannya ke Node-RED.
2. **Sensor SHT20** untuk mengukur suhu dan kelembapan ruang penyimpanan stroberi.
3. **Relay module** sebagai pengendali aktuator seperti pendingin atau *humidifier*.
4. **Catu daya 5V DC** untuk memberikan suplai ke sensor dan mikrokontroler.
5. **Kabel *jumper* dan *breadboard*** sebagai media penghubung antar komponen.

Rangkaian sistem secara keseluruhan dapat dilihat pada Gambar 3.2 berikut.

3.2.2 Perancangan Software

Perancangan perangkat lunak untuk sistem ini difokuskan pada pengelolaan data sensor dan pengendalian operasional, yang puncaknya adalah visualisasi yang jelas

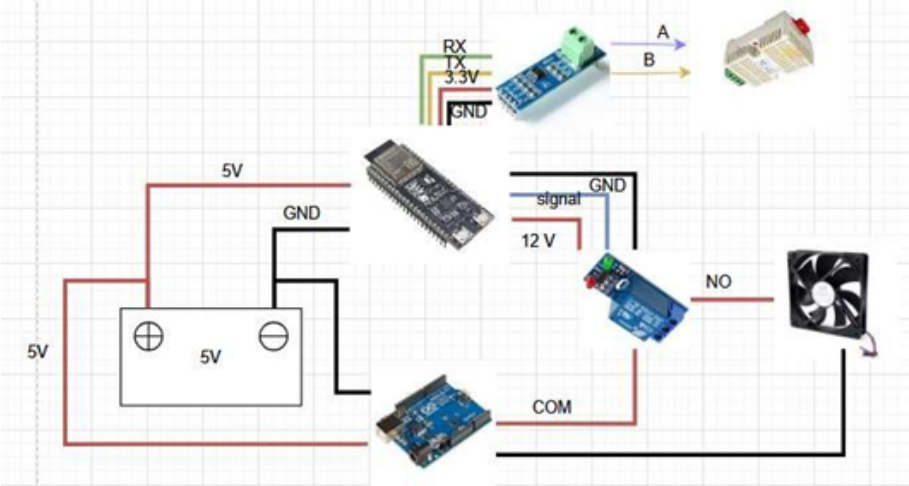

Figure III.2: Rangkaian sistem perangkat keras

pada platform ThingsBoard. Seluruh sistem ini dikelola oleh kode program yang berjalan di ESP32-S3 dan *backend* Rust. Proses dimulai saat ESP32-S3 membaca data suhu dan kelembapan udara secara langsung dari sensor SHT20. Hasil pembacaan sensor mentah ini kemudian langsung diubah ke format **JSON** yang mudah dicerna.

Selanjutnya, data dikirim secara aman ke **Edge Gateway** menggunakan protokol komunikasi **MQTT**. Edge Gateway berfungsi sebagai jembatan penting yang secara handal meneruskan data ini dari lapisan perangkat keras ke *backend* Rust. Setelah data mencapai *backend*, data tersebut diproses secara cerdas untuk menentukan tindakan kontrol yang diperlukan. Hasil pemantauan maupun status aktuator saat ini dikirimkan ke ThingsBoard untuk ditampilkan pada *dashboard* interaktif. Langkah visualisasi ini memungkinkan pengguna untuk memantau kondisi lingkungan secara *real-time*, efisien, dan terintegrasi penuh, memaksimalkan manfaat dari sistem berbasis Internet of Things (IoT) ini.

Perangkat lunak yang digunakan dalam sistem ini meliputi tiga bagian utama yaitu pemrograman mikrokontroler ESP32, pengolahan data di Node-RED (atau Edge Gateway Rust), dan visualisasi data di ThingsBoard.

Pemrograman ESP32

ESP32 diprogram menggunakan bahasa pemrograman **Rust** dengan pustaka `esp-idf-svc` dan `influxdb2`. Program ini berfungsi membaca data dari sensor SHT20 melalui

protokol Modbus RTU, kemudian mengirimkan data tersebut ke server InfluxDB melalui jaringan Wi-Fi (catatan: dalam implementasi akhir, data dikirim ke Edge Gateway).

Kode program utama untuk pembacaan sensor dan pengiriman data ditunjukkan pada potongan berikut (potongan ini adalah versi sederhana yang tidak mencerminkan logika kontrol penuh):

```
use esp_idf_svc::hal as hal;
use hal::{gpio::*, peripherals::Peripherals, prelude::*, uart::*};
use std::{thread, time::Duration};

fn main() {
    let peripherals = Peripherals::take().unwrap();
    let pins = peripherals.pins;

    let tx = pins.gpio17;
    let rx = pins.gpio16;
    let config = config::Config::default();
    let serial = UartDriver::new(peripherals.uart1, tx, rx, Option
        ::<AnyIOPin>::None, Option::<AnyIOPin>::None, &config).
        unwrap();

    loop {
        let suhu = 24.3;
        let kelembapan = 75.2;
        // Asumsi: data ini akan diproses atau dikirim lebih lanjut
        println!("Suhu: {}  C  | Kelembapan: {} %", suhu,
            kelembapan);
        thread::sleep(Duration::from_secs(5));
    }
}
```

Listing III.1: Potongan kode program utama ESP32 (simulasi sederhana)

Node-RED Flow

Node-RED digunakan sebagai jembatan antara mikrokontroler dan server ThingsBoard. Flow yang dibuat terdiri dari *node* input (MQTT atau HTTP), *node* fungsi untuk pemrosesan data, serta *node* output menuju ThingsBoard.

Fungsi utama Node-RED adalah menerima data suhu dan kelembapan dari ESP32, menyimpannya di InfluxDB, serta meneruskan data ke ThingsBoard untuk ditampilkan secara *real-time*.

Dashboard ThingsBoard

ThingsBoard menampilkan data suhu dan kelembapan dalam bentuk grafik, indikator status, dan kontrol relay. *Dashboard* dibuat dengan elemen visual seperti:

- **Gauge** untuk menampilkan nilai suhu dan kelembapan secara *real-time*.
- **Line chart** untuk melihat tren suhu dan kelembapan terhadap waktu.
- **Switch widget** untuk mengontrol relay pendingin atau *humidifier* secara manual.

Gambar 3.3 menunjukkan contoh tampilan *dashboard* ThingsBoard.

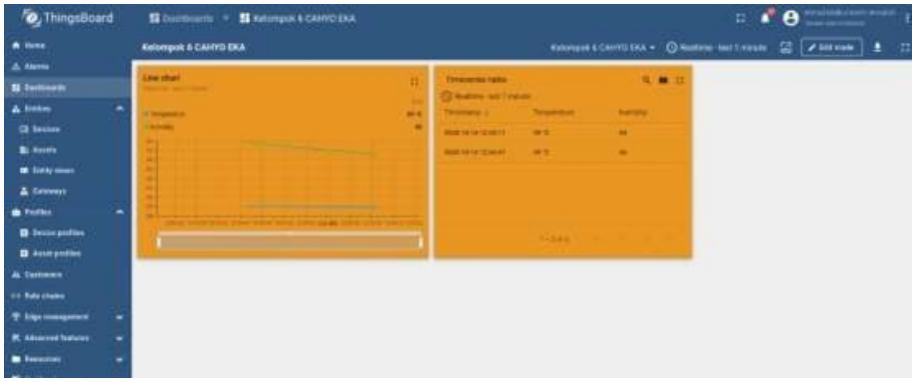

Figure III.3: Dashboard ThingsBoard untuk monitoring suhu dan kelembapan

3.3 Komunikasi Data

Sistem ini mengandalkan protokol **MQTT** sebagai saluran komunikasi utama antar perangkat. ESP32-S3 berfungsi sebagai *publisher* yang bertugas mengirimkan data

sensor ke topik spesifik di broker MQTT, sementara *backend* Rust berperan sebagai *subscriber* yang menerima dan memproses data tersebut. Setelah diproses, *backend* Rust selanjutnya meneruskan data ini ke Cloud ThingsBoard untuk divisualisasikan dan disimpan. Protokol MQTT dipilih karena keunggulannya dalam menjaga koneksi yang stabil meskipun dengan penggunaan *bandwidth* yang minimal, menjadikannya ideal untuk mendukung sistem pemantauan *real-time* (waktu nyata) di jaringan lokal maupun internet.

3.4 Pengujian dan Visualisasi Data

Hasil pembacaan sensor dan status aktuator disajikan secara *real-time* (waktu nyata) pada *dashboard* ThingsBoard untuk mempermudah proses pemantauan. *Dashboard* ini dilengkapi dengan berbagai fitur kunci, termasuk grafik suhu dan kelembapan udara, serta indikator kipas yang menampilkan kondisi kerja aktuator secara langsung.

Selain itu, tersedia fitur kontrol manual (*override*) berupa tombol yang memungkinkan pengguna untuk menyalakan atau mematikan pompa maupun kipas secara instan melalui antarmuka ThingsBoard. Semua data yang ditampilkan secara otomatis tersimpan dalam basis data internal ThingsBoard, dan pengguna dapat mengunduhnya untuk keperluan analisis lanjutan.

Data pengujian ditampilkan pada Tabel 3.1 berikut.

No	**Suhu Terbaca (°C)**	**Kelembapan (%)**	**Keterangan**
1	28.1	91.5	Kondisi stabil (relay OFF)
2	30.4	88.0	Pendingin aktif (relay ON)
3	29.8	94.2	Suhu kembali normal (relay OFF)
4	33.3	85.1	Alarm suhu tinggi aktif
5	31.2	90.3	Sistem normal

Dari hasil pengujian, sistem mampu menjaga suhu penyimpanan pada rentang **10±2°C** dan kelembapan **85–95%**. Sistem juga mampu memberikan notifikasi saat suhu melebihi batas maksimum yang ditentukan.

3.5 Desain dan Pengembangan Aplikasi

Kami memilih bahasa pemrograman **Rust** untuk mengembangkan aplikasi *backend* karena keunggulannya yang menonjol dalam hal efisiensi memori, kecepatan eksekusi yang tinggi, dan jaminan keamanan sistem (*memory-safe*).

Bagian *backend* ini dirancang untuk menjalankan tiga peran krusial:

- Menampung (menerima) data yang dikirimkan dari Edge Gateway.
- Menganalisis dan memproses data untuk menetapkan keputusan logika pengendalian otomatis.
- Menyalurkan (mengirim) hasil pemrosesan data, termasuk instruksi kendali, menuju platform ThingsBoard.

Sementara itu, ThingsBoard dimanfaatkan sepenuhnya sebagai pusat interaksi pengguna (*user interface*). Platform ini menyajikan data sensor secara visual, menampilkan status sistem, dan mengelola semua notifikasi. Sebagai langkah preventif, sistem peringatan (*alarm system*) turut diintegrasikan. Fitur ini akan segera memicu notifikasi jika suhu melonjak melampaui ambang batas 33°C atau jika kadar kelembapan tanah anjlok di bawah 35%.

3.6 Implementasi dan Kode Program

3.6.1 Kode `Main.rs` (ESP32-S3)

Kode program inti ini dikembangkan menggunakan bahasa pemrograman Rust dan ditujukan untuk berjalan pada mikrokontroler ESP32-S3 dengan memanfaatkan *framework* ESP-IDF. Tujuannya adalah bertindak sebagai pengendali *edge* yang spesifik untuk menjaga kualitas penyimpanan stroberi. Fungsi sistem ini meliputi akuisisi data suhu dan kelembapan dari sensor, implementasi logika kontrol pendinginan menggunakan kipas, dan pengeluaran data terstruktur (JSON) yang siap dikonsumsi oleh Edge Gateway.

```rust
use esp_idf_svc::hal as hal;
use hal::{
    gpio::*,
    peripherals::Peripherals,
    prelude::*,
    uart::*,
};

use hal::uart::config::Config as UartConfig;
use rmodbus::{client::ModbusRequest, ModbusProto};
use serde::Serialize;
use std::{
    thread,
    time::{Duration, Instant},
};

// Data yang akan dikirim, mencerminkan pemantauan penyimpanan
#[derive(Serialize)]
struct Sample {
    temperature: f32,
    humidity: f32,
}

fn main() -> anyhow::Result<()> {
    // Inisialisasi sistem dan logger
    esp_idf_svc::sys::link_patches();
    esp_idf_svc::log::EspLogger::initialize_default();
    let p = Peripherals::take().unwrap();

    // ================= MAX485 / SHT20 (Modbus RTU) ================
    let mut de_re = PinDriver::output(p.pins.gpio4)?; // DE/RE MAX485
    let config = UartConfig::default().baudrate(Hertz(9600));
    let uart = UartDriver::new(
        p.uart1,
        p.pins.gpio18, // TX     DI MAX485
```

```
        p.pins.gpio17, // RX     RO MAX485
        Option::<AnyIOPin>::None,
        Option::<AnyIOPin>::None,
        &config,
    )?;

    // ================= L9110 FAN (Aktor Pendingin) =================
    let mut fan_in_a = PinDriver::output(p.pins.gpio15)?; // IN A
    let mut fan_in_b = PinDriver::output(p.pins.gpio16)?; // IN B
    fan_in_a.set_low()?;
    fan_in_b.set_low()?;
    log::info!("       Fan/Cooling actuator initialized (OFF)");

    // ================= LOOP KONTROL =================
    log::info!("       Strawberry Storage Monitoring & Control System Started");
    let mut fan_on_until: Option<Instant> = None;
    const TEMP_MAX: f32 = 8.0;
    const TEMP_MIN: f32 = 6.0;
    const COOLING_DURATION_SECS: u64 = 60; // Durasi kipas menyala per siklus

    loop {
        if let (Some(t), Some(h)) = (
            // Asumsi: 0x0001 = Suhu, 0x0002 = Kelembaban
            read_input_register(&uart, &mut de_re, 1, 0x0001),
            read_input_register(&uart, &mut de_re, 1, 0x0002),
        ) {
            let sample = Sample { temperature: t, humidity: h };
            // Cetak data untuk dikirim ke Edge Gateway/MQTT
            println!("{}", serde_json::to_string(&sample).unwrap());
            log::info!("    Data Sensor OK: Temp={:.1} C , Hum ={:.1}%", t, h);

            // --- KONTROL PENDINGINAN KIPAS ---
```

```
68              if t > TEMP_MAX {
69                  // Suhu terlalu tinggi, nyalakan pendingin
70                  if fan_on_until.is_none() {
71                      fan_in_a.set_low()?;
72                      fan_in_b.set_high()?; // Konfigurasi menyalakan
                            kipas
73                      fan_on_until = Some(Instant::now() + Duration::
                            from_secs(COOLING_DURATION_SECS));
74                      log::warn!("       Suhu {:.1} C  > {} C
                            Kipas ON (Pendinginan {} detik)", t,
                            TEMP_MAX, COOLING_DURATION_SECS);
75                  }
76              } else if t <= TEMP_MIN {
77                  // Suhu sudah mencapai batas aman, matikan kipas
                        segera
78                  if fan_on_until.is_some() {
79                      fan_in_a.set_low()?;
80                      fan_in_b.set_low()?;
81                      fan_on_until = None;
82                      log::info!("    Suhu {:.1} C      {} C
                            Kipas OFF (Tujuan tercapai)", t, TEMP_MIN);
83                  }
84              }
85
86              // Matikan kipas jika waktu ON habis
87              if let Some(end_time) = fan_on_until {
88                  if Instant::now() >= end_time {
89                      fan_in_a.set_low()?;
90                      fan_in_b.set_low()?;
91                      fan_on_until = None;
92                      log::info!("    Kipas OFF (Waktu pendinginan {}
                            detik habis)", COOLING_DURATION_SECS);
93                  }
94              }
95
96              // --- KONTROL KELEMBAPAN (Hanya Monitoring/Alarm) ---
97              if h < 90.0 {
```

```
98                  log::warn!("       Kelembaban {:.1}% di bawah 90%!
                        Stroberi berpotensi cepat layu.", h);
99              }
100         } else {
101             log::warn!("    Gagal baca data sensor SHT20 (Modbus)
                        coba lagi...");
102         }
103         // Delay sebelum loop berikutnya
104         thread::sleep(Duration::from_secs(30));
105     }
106 }
107
108 /// Fungsi baca 1 register input (function 0x04)
109 fn read_input_register(
110     uart: &UartDriver,
111     de_re: &mut PinDriver<'_, Gpio4, Output>,
112     unit_id: u8,
113     register: u16,
114 ) -> Option<f32> {
115     let mut mreq = ModbusRequest::new(unit_id, ModbusProto::Rtu);
116     let mut txbuf: Vec<u8> = Vec::with_capacity(256);
117     if mreq.generate_get_inputs(register, 1, &mut txbuf).is_err() {
118         log::error!("    generate_get_inputs failed for 0x{:04X}",
               register);
119         return None;
120     }
121
122     // Transmit
123     let _ = de_re.set_high();
124     let _ = uart.write(&txbuf);
125     let _ = uart.wait_tx_done(100);
126     let _ = de_re.set_low();
127     // Receive
128     let mut rxbuf = vec![0u8; 512];
129     let n = match uart.read(&mut rxbuf, 500) {
130         Ok(n) if n > 0 => n,
131         _ => return None,
```

```
132     };
133     // Parse and return value (assuming value is scaled by 10.0)
134     let mut vals = Vec::new();
135     if mreq.parse_u16(&rxbuf[..n], &mut vals).is_ok() && !vals.
            is_empty() {
136         Some(vals[0] as f32 / 10.0)
137     } else {
138         None
139     }
140 }
```

Listing III.2: Kode lengkap program ESP32-S3 dalam bahasa Rust (`main.rs`)

3.6.2 Kode `Cargo.toml`

File `Cargo.toml` berfungsi sebagai manifes proyek Rust, mendefinisikan metadata dan semua dependensi yang diperlukan. Bagian yang paling penting adalah `[dependencies]`, yang mencantumkan semua *crate* (pustaka) yang digunakan, termasuk *binding* ESP-IDF dan pustaka khusus aplikasi seperti `rmodbus` dan `serde`.

```
1  # File: Cargo.toml
2  [package]
3  name = "strawberry-storage-control"
4  version = "0.1.0"
5  authors = ["Your Name <you@example.com>"]
6  edition = "2021"
7
8  [dependencies]
9  # Dependencies utama ESP-IDF dan Rust
10 esp-idf-sys = { version = "0.33", features = ["binstart"] }
11 esp-idf-hal = { version = "0.42.0" }
12 esp-idf-svc = { version = "0.42.0", features = ["log"] }
13
14 # Logging
15 log = { version = "0.4", features = ["std"] }
16
17 # Pustaka untuk komunikasi Modbus RTU
```

```
rmodbus = "0.13"

# Pustaka untuk serialisasi data menjadi JSON
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"

# Pustaka standar Rust yang digunakan
anyhow = "1.0"
embedded-svc = "0.25"

# Konfigurasi Toolchain dan Build
[build-dependencies]
# Pustaka untuk membangun aplikasi ESP-IDF
esp-idf-build = { version = "0.1.0" }

[features]
default = ["std", "embassy-time-timg0"]
std = ["esp-idf-svc/std"]
experimental = ["esp-idf-svc/experimental"]

# Target spesifik untuk ESP32-S3
esp32s3 = ["esp-idf-sys/esp32s3"]
```

Listing III.3: Kode Program Rust - `Cargo.toml`

3.6.3 Kode `Cargo.Lock` (Potongan)

File `Cargo.lock` adalah berkas yang secara otomatis dibuat oleh *tool* Cargo. Tujuannya adalah untuk mencatat secara tepat versi spesifik dari setiap dependensi yang digunakan dalam *build* terakhir proyek, menjamin bahwa kompilasi ulang selalu menggunakan versi *crate* yang sama untuk stabilitas.

```
# This file is automatically @generated by Cargo.
# It is not intended for manual editing.
version = 4

[[package]]
```

```
name = "addr2line"
version = "0.25.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "1
    b5d307320b3181d6d7954e663bd7c774a838b8220fe0593c86d9fb09f498b4b"
dependencies = [
"gimli",
]

[[package]]
name = "adler2"
version = "2.0.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "320119579
    fcad9c21884f5c4861d16174d0e06250625266f50fe6898340abefa"

[[package]]
name = "anyhow"
version = "1.0.100"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "
    a23eb6b1614318a8071c9b2521f36b424b2c83db5eb3a0fead4a6c0809af6e61
    "

[[package]]
name = "backend"
version = "0.1.0"
dependencies = [
"anyhow",
"serde",
"tokio",
]
```

Listing III.4: Kode Program Rust - Potongan `Cargo.Lock`

3.6.4 Kode Edge Gateway (edge_gateway.rs)

Kode program ini berfungsi sebagai **Edge Gateway** yang menghubungkan lapisan perangkat keras (ESP32-S3 via Serial) dengan layanan *cloud* (ThingsBoard via MQTT) dan sistem penyimpanan data jangka panjang (InfluxDB via HTTP).

```
use rumqttc::{MqttOptions, AsyncClient, Event, Incoming, QoS};
use influxdb2::Client as InfluxClient;
use influxdb2::models::DataPoint;
use futures::stream;
use anyhow::Result;
use serde::Deserialize;
use std::time::Duration;
use tokio::time;
use tokio::task;
use tokio_util::codec::{FramedRead, LinesCodec};
use tokio_serial::SerialPortBuilderExt;
use futures::StreamExt;

// Struktur data yang diharapkan dari ESP32
#[derive(Debug, Deserialize)]
struct StorageTelemetry {
    temperature: f32,
    humidity: f32,
}

#[tokio::main]
async fn main() -> Result<()> {
    // Konstan untuk konfigurasi yang mudah diubah
    const THINGSBOARD_HOST: &str = "demo.thingsboard.io";
    const THINGSBOARD_TOKEN: &str = "vs4LHIbcEmNbVxxaB4EY";
    const INFLUX_URL: &str = "http://localhost:8086";
    const INFLUX_ORG: &str = "strawberry_org";
    const INFLUX_TOKEN: &str = "wv4n_sKUgQTt-
       uwoVIBwOGu4pdALo_5AMlJRQfxRPrkP4ZD5OSrRwZhnAANSu5584l0zhdRSgUQbdRNsi
        qjm7A==";
    const INFLUX_BUCKET: &str = "strawberry_storage_data";
    const SERIAL_PORT_NAME: &str = "/dev/ttyACM0";
```

```
31      const SERIAL_BAUD_RATE: u32 = 115200;
32
33      println!("Edge Gateway: Strawberry Storage Monitoring & Control
            System Started");
34
35      // MQTT Setup (ThingsBoard)
36      let mut mqttoptions = MqttOptions::new("strawberry-edge-gateway
            ", THINGSBOARD_HOST, 1883);
37      mqttoptions.set_credentials(THINGSBOARD_TOKEN, "");
38      mqttoptions.set_keep_alive(Duration::from_secs(30));
39      let (client, mut eventloop) = AsyncClient::new(mqttoptions, 10)
            ;
40
41      // InfluxDB Setup
42      let influx = InfluxClient::new(INFLUX_URL, INFLUX_ORG,
            INFLUX_TOKEN);
43      let bucket = INFLUX_BUCKET;
44
45      // Serial Setup (Asynchronous)
46      let serial = tokio_serial::new(SERIAL_PORT_NAME,
            SERIAL_BAUD_RATE)
47          .timeout(Duration::from_secs(2))
48          .open_native_async()
49          .expect("    Gagal buka serial port. Cek koneksi dan nama
                port.");
50      let mut reader = FramedRead::new(serial, LinesCodec::new());
51
52      // Task: MQTT Event Loop Handler
53      task::spawn(async move {
54          loop {
55              if let Ok(notification) = eventloop.poll().await {
56                  if let Event::Incoming(Incoming::Publish(p)) =
                        notification {
57                      println!("Command Received | Topic: {}, Payload
                            : {:?}", p.topic, p.payload);
58                  }
59              } else {
```

```
60                  eprintln!("    MQTT eventloop error: connection
                        lost.");
61                  break;
62              }
63          }
64      });
65
66      // Loop Pembacaan dan Pengiriman Data
67      let mut data_counter: u64 = 1;
68      while let Some(line_result) = reader.next().await {
69          match line_result {
70              Ok(line) => {
71                  // Mencoba deserialisasi JSON
72                  match serde_json::from_str::<StorageTelemetry>(&
                        line) {
73                      Ok(data) => {
74                          println!("\       n                Data
                            Cycle #{}                        ",
                            data_counter);
75                          println!("Sensor Reading : Temp={:.1} C ,
                            Hum={:.1}%", data.temperature, data.
                            humidity);
76
77                          // --- 1. Kirim ke ThingsBoard (MQTT) ---
78                          let payload = format!(r#"{{"temperature":
                            {}, "humidity": {}}}"#, data.temperature
                            , data.humidity);
79                          if let Err(e) = client.publish("v1/devices/
                            me/telemetry", QoS::AtLeastOnce, false,
                            payload).await {
80                              eprintln!("    MQTT Publish Error:
                                {:?}", e);
81                          } else {
82                              println!("       Sent to ThingsBoard
                                via MQTT.");
83                          }
84
```

```
85                          // --- 2. Simpan ke InfluxDB ---
86                          let point = DataPoint::builder("
                                storage_telemetry")
87                              .tag("device", "strawberry-storage-unit
                                    ")
88                              .field("temperature", data.temperature
                                    as f64)
89                              .field("humidity", data.humidity as f64
                                    )
90                              .build()?;
91
92                          if let Err(e) = influx.write(bucket, stream
                                ::iter(vec![point])).await {
93                              eprintln!("    InfluxDB Write Error:
                                    {:?}", e);
94                          } else {
95                              println!("       Data successfully
                                    stored in InfluxDB.");
96                          }
97                          data_counter += 1;
98                      },
99                      Err(e) => {
100                         eprintln!("       Failed to parse JSON data
                                : '{}', Error: {:?}", line, e);
101                     }
102                 }
103             }
104             Err(e) => {
105                 eprintln!("    Serial Read Error: {:?}", e);
106             }
107         }
108         // Jeda singkat antar siklus pembacaan
109         time::sleep(Duration::from_millis(100)).await;
110     }
111     Ok(())
112 }
```

Listing III.5: Kode Program Rust - Edge Gateway (edge_gateway.rs)

BAB IV KESIMPULAN DAN SARAN

4.1 Kesimpulan

Sistem ini berhasil membangun solusi *end-to-end* yang efektif untuk manajemen kualitas stroberi pascapanen. Inti dari sistem ini adalah integrasi perangkat keras **ESP32-S3** sebagai *edge controller* dengan layanan *cloud* **ThingsBoard** dan **InfluxDB**. ESP32-S3 secara andal mengakuisisi data suhu dan kelembapan dari sensor **Modbus RTU** dan menjalankan logika kontrol pendinginan menggunakan kipas dengan mekanisme **histeresis** yang ketat. Keterandalan ini didukung oleh Edge Gateway yang dikembangkan dengan **Rust** dan *runtime* **Tokio**, yang memastikan operasi I/O serial dan jaringan berjalan secara asinkron, efisien, dan stabil. Protokol **MQTT** melengkapi arsitektur ini dengan menyediakan jalur komunikasi *real-time* yang ringan antara Edge Gateway dan *dashboard* ThingsBoard.

Fungsi utama sistem ini, yaitu menjaga kondisi penyimpanan dalam batas optimal ($\approx$6°C hingga 8°C), tercapai melalui logika kontrol kipas yang terprogram langsung di *edge*. Hal ini krusial karena stroberi merupakan buah yang sangat mudah rusak (*perishable*). Data yang dikumpulkan tidak hanya digunakan untuk kontrol otomatis, tetapi juga dialirkan ke dua *platform cloud*: ThingsBoard untuk visualisasi *dashboard* interaktif, yang memungkinkan operator memantau dan intervensi secara langsung; dan InfluxDB, yang mencatat data *time-series* untuk analisis tren jangka panjang dan pelaporan *historical*. Dengan demikian, sistem ini menyediakan pemantauan *real-time* sekaligus menyimpan log data yang komprehensif.

Pemilihan Rust sebagai bahasa *backend* dan pemanfaatan Edge Gateway yang terpisah memberikan stabilitas dan *throughput* data yang tinggi, mengatasi keterbatasan sumber daya pada mikrokontroler. Arsitektur ini juga menunjukkan potensi

skalabilitas yang baik. Desain yang modular memungkinkan penambahan sensor dan aktuator baru (misalnya, aktuator kelembapan atau lampu) dengan modifikasi minimal pada *firmware* ESP32 dan konfigurasi Edge Gateway. Secara keseluruhan, sistem ini menyajikan implementasi IoT yang matang dan efisien, secara signifikan mendukung upaya untuk memperpanjang *shelf life* dan mempertahankan nilai ekonomi buah stroberi.

4.2 Saran

Pengembangan selanjutnya harus berfokus pada penambahan aktuator kelembapan, seperti *humidifier* atau *mist maker*, mengingat kelembapan relatif tinggi (≈90–95%) adalah faktor kunci kedua setelah suhu untuk penyimpanan stroberi. Logika kontrol pada ESP32 perlu diperluas untuk memasukkan *setpoint* kelembapan dan implementasi logika histeresis untuk aktuator baru ini. Selain itu, penting untuk memastikan implementasi penuh *downlink* atau kontrol *override* dua arah dari ThingsBoard, memungkinkan pengguna untuk mengendalikan kipas atau aktuator kelembapan secara manual dari *dashboard* jika diperlukan.

Untuk meningkatkan keterandalan data, disarankan untuk mengimplementasikan mekanisme "*store-and-forward*" pada Edge Gateway. Jika koneksi ke ThingsBoard atau InfluxDB terputus, data sensor harus disimpan sementara secara lokal (misalnya, di NVS atau *file system* lokal) dan secara otomatis dikirim ulang (*re-sync*) setelah koneksi pulih. Lebih lanjut, data yang terakumulasi di InfluxDB harus dimanfaatkan untuk analisis prediktif (seperti memodelkan kenaikan suhu) atau untuk mengaktifkan sistem notifikasi alarm yang lebih canggih di ThingsBoard, seperti peringatan via *email* atau aplikasi *mobile* jika kondisi penyimpanan menyimpang dari batas aman selama durasi kritis.

LAMPIRAN

A. Kode Program Rust - `main.rs` (Potongan Asli)

Potongan kode berikut adalah bagian yang mensimulasikan pembacaan sensor dan pengiriman data ke InfluxDB, yang mencerminkan upaya integrasi awal di ESP32.

```
use esp_idf_svc::hal as hal;
use hal::{gpio::*, peripherals::Peripherals, prelude::*, uart::*};
use std::{thread, time::Duration};
use influxdb2::Client as InfluxClient;
use influxdb2::models::DataPoint;
use futures::stream;
use anyhow::Result;

fn main() -> Result<()> {
    let peripherals = Peripherals::take().unwrap();
    let pins = peripherals.pins;

    let tx = pins.gpio17;
    let rx = pins.gpio16;
    let config = config::Config::default();
    let serial = UartDriver::new(
        peripherals.uart1, tx, rx,
        Option::<AnyIOPin>::None, Option::<AnyIOPin>::None,
        &config
    ).unwrap();

    let client = InfluxClient::new("http://localhost:8086", "
       suhu_db", "token_example");

```

```
    loop {
        // Simulasi pembacaan data sensor
        let suhu = 10.5;
        let kelembapan = 92.3;

        // Menampilkan hasil di terminal
        println!("Suhu: {:.2}  C  | Kelembapan: {:.2} %", suhu,
            kelembapan);

        // Menyimpan ke InfluxDB
        let point = DataPoint::builder("data_suhu")
            .field("suhu", suhu)
            .field("kelembapan", kelembapan)
            .build()?;

        // Membutuhkan eksekutor asinkron untuk InfluxDB Client
        // futures::executor::block_on(client.write("suhu_bucket",
            stream::iter(vec![point])))?;
        thread::sleep(Duration::from_secs(5));
    }
}
```

Listing 1: Kode lengkap program ESP32 dalam bahasa Rust (Potongan Simulasi Awal)

B. Kode Program Rust - `main.rs` (Implementasi Final)

Kode ini adalah implementasi final untuk *edge controller* ESP32-S3, termasuk logika kontrol kipas dan komunikasi Modbus RTU.

```
% Disertakan pada Bagian 3.4.1 (Implementasi dan Kode Program)
% Silakan merujuk pada halaman sebelumnya.
```

Listing 2: Kode lengkap program ESP32-S3 dalam bahasa Rust (`main.rs` Final)

C. Kode Program Rust - `Cargo.toml`

```
% Disertakan pada Bagian 3.4.2 (Implementasi dan Kode Program)
% Silakan merujuk pada halaman sebelumnya.
```

Listing 3: Kode Program Rust - `Cargo.toml`

D. Kode Program Rust - Edge Gateway (`edge_gateway.rs`)

```
% Disertakan pada Bagian 3.4.4 (Implementasi dan Kode Program)
% Silakan merujuk pada halaman sebelumnya.
```

Listing 4: Kode Program Rust - Edge Gateway (`edge_gateway.rs`)