

LAPORAN PROGRESS TUGAS INTERKONEKSI SISTEM INSTRUMENTASI VI231418

Sistem Monitoring Suhu dan Kelembaban Tempe untuk Menjaga Kualitas Fermentasi

Nama Kelompok : 6

Anggota Kelompok:

Ishmatu Aulia Rizky Kirana (2042231004)

Siti Aisyah (2042231008)

Shelma Nur Sabila (2042231022)

Riska Hidayati Laena (2042231054)

Dosen Pengampu

Muhammad Roy Ashiddiqi, S.T., M.T.

Ahmad Radhy, S.Si., M.Si.

Program Studi Rekayasa Teknologi Instrumentasi

Departemen Teknik Instrumentasi

Fakultas Vokasi

Institut Teknologi Sepuluh Nopember

Surabaya

Tahun 2025



LAPORAN PROGRES TUGAS INTERKONEKSI SISTEM INSTRUMENTASI VI231418

Sistem Monitoring Suhu dan Kelembaban Tempe untuk Menjaga Kualitas Fermentasi

Nama Kelompok : 6

Anggota Kelompok :

Ishmatu Aulia Rizky Kirana (2042231004)

Siti Aisyah (2042231008)

Shelma Nur Sabila (2042231022)

Riska Hidayati Laena (2042231054)

Dosen Pengampu

Muhammad Roy Ashiddiqi, S.T., M.T.

Ahmad Radhy, S.Si., M.Si.

Program Studi Rekayasa Teknologi Instrumentasi

Departemen Teknik Instrumentasi

Fakultas Vokasi

Institut Teknologi Sepuluh Nopember

Surabaya

Tahun 2025

ABSTRAK

Sistem Monitoring Suhu dan Kelembaban Tempe untuk Menjaga Kualitas Fermentasi

Nama Mahasiswa / NRP	:	Ishmatu Aulia Rizky Kirana Siti Aisyah Shelma Nur Sabila Riska Hidayati Laena	(2042231004) (2042231008) (2042231022) (2042231054)
Departemen	:	Teknik Instrumentasi VOKASI - I	
Dosen Pengampu	:	Ahmad Radhy, S.Si., M.Si.	

Fermentasi tempe merupakan proses biologis yang sangat bergantung pada kondisi lingkungan, terutama suhu dan kelembaban. Dalam skala produksi rumahan maupun industri kecil, ketidakteraturan kondisi ini dapat menyebabkan kualitas tempe menurun atau bahkan gagal difermentasi. Untuk menjawab permasalahan tersebut, dikembangkan sebuah sistem monitoring suhu dan kelembaban berbasis sensor SHT20 yang mampu membaca data secara akurat dan stabil. Sensor ini terhubung menggunakan protokol komunikasi Modbus RTU melalui jalur RS-485, sehingga cocok digunakan pada lingkungan yang membutuhkan kestabilan dan jarak komunikasi yang cukup jauh.

Sistem ini dirancang untuk mencatat dan mengirim data ke server lokal secara real-time menggunakan mikrokontroler industri. Data suhu dan kelembaban kemudian disimpan dalam database InfluxDB dan divisualisasikan melalui Grafana agar mudah dipantau. Dengan adanya sistem ini, diharapkan proses fermentasi tempe dapat berjalan lebih terkontrol, meningkatkan efisiensi waktu, dan menjaga kualitas produk secara konsisten. Sistem ini sangat potensial untuk diimplementasikan pada UMKM yang ingin meningkatkan produktivitasnya melalui penerapan teknologi sederhana namun efektif.

Kata kunci: fermentasi tempe, suhu, kelembaban, SHT20, Modbus RTU, monitoring, InfluxDB, Grafana.

DAFTAR ISI

DAFTAR ISI.....	2
BAB I PENDAHULUAN	3
1.1 Latar Belakang	3
1.2 Rumusan Masalah.....	4
1.3 Tujuan Proyek.....	4
BAB II TINJAUAN PUSTAKA.....	5
2.1 Metodologi dan Arsitektur Sistem	5
2.1.1 Desain Arsitektur Sistem	5
2.2 Deskripsi Komponen.....	6
2.2.1 Sensor: SHT20 dengan komunikasi Modbus RTU	6
2.2.2 Modbus Client (Rust): Pembacaan data suhu dan kelembaban dari sensor	6
2.2.3 TCP Server (Rust): Menerima data JSON, parsing, dan simpan ke InfluxDB	7
2.2.4 InfluxDB: Menyimpan data time-series	8
2.2.5 Grafana: Menampilkan dashboard suhu dan kelembaban	8
2.2.6 QT	9
2.3 Format Payload	10
BAB III HASIL PERCOBAAN	11
3.1 Implementasi dan Kode Program	11
3.1.1 Kode Rust Modbus Client SHT 20 MAIN DAN CARGO	11
3.1.2 Kode Rust TCP Server	13
3.1.3 Konfigurasi InfluxDB dan Integrasi.....	16
3.1.4 Dashboard Grafana.....	17
3.1.5 Integrasi Blockchains dan Web 3.....	18
BAB IV PENGUJIAN & HASIL	23
BAB V KESIMPULAN DAN REKOMENDASI	24
DAFTAR PUSTAKA.....	25

BAB I PENDAHULUAN

1.1 Latar Belakang

Fermentasi tempe merupakan proses biologis yang sangat dipengaruhi oleh kondisi lingkungan, khususnya suhu dan kelembaban. Jamur Rhizopus oligosporus yang digunakan dalam fermentasi tempe memerlukan rentang suhu optimal sekitar 30–37 °C dan kelembaban relatif sekitar 70–80% untuk tumbuh secara efisien dan menghasilkan struktur tempe yang padat, putih, dan tidak berlendir (Jiang et al., 2021). Penyimpangan dari parameter ini dapat menyebabkan pertumbuhan jamur terhambat, struktur tempe tidak terbentuk dengan baik, atau bahkan terjadi kontaminasi mikroba lain yang merugikan (Wang & Liu, 2022). Oleh karena itu, diperlukan sistem monitoring suhu dan kelembaban yang presisi dan andal untuk menjamin keberhasilan fermentasi tempe, khususnya dalam skala industri rumah tangga dan UMKM yang belum menggunakan sistem kontrol otomatis.

Sensor SHT20 dipilih dalam sistem ini karena memiliki karakteristik yang unggul dibandingkan sensor suhu dan kelembaban lain dalam kelasnya. Sensor ini dirancang untuk penggunaan industri, memiliki akurasi tinggi sebesar $\pm 0,3$ °C untuk suhu dan $\pm 3\%$ RH untuk kelembaban, serta kemampuan untuk bekerja dalam rentang suhu –40 °C hingga 125 °C, dan kelembaban hingga 100% RH (Zhang et al., 2021). Selain itu, SHT20 mendukung komunikasi digital berbasis protokol Modbus RTU melalui jalur RS-485, yang jauh lebih stabil dibandingkan sistem wireless seperti Wi-Fi pada ESP8266 yang sering mengalami gangguan elektromagnetik dan interferensi sinyal (Kim et al., 2023). Komunikasi RS-485 ini juga memungkinkan pengiriman data hingga jarak 1000 meter tanpa penurunan kualitas sinyal, sehingga cocok digunakan pada lingkungan produksi yang memiliki banyak inkubator atau ruangan terpisah.

Tidak seperti sistem berbasis ESP, sistem ini menggunakan mikrokontroler industri seperti STM32 atau ATmega yang diprogram untuk membaca data dari sensor SHT20 melalui RS-485 dan meneruskannya ke server lokal atau cloud. Dengan pendekatan ini, stabilitas komunikasi dan keamanan data menjadi lebih terjamin, terutama dalam lingkungan yang memiliki banyak perangkat elektronik aktif (Park et al., 2022). Sistem kemudian mengatur berbagai aktuator seperti pemanas ruangan, kipas ventilasi, dan humidifier secara otomatis berdasarkan nilai ambang batas yang telah ditentukan, misalnya suhu <30 °C atau >37 °C, dan kelembaban $<70\%$ atau $>80\%$. Selain itu, data suhu dan kelembaban dicatat dalam basis data secara periodik sehingga dapat digunakan untuk pelacakan dan evaluasi proses produksi secara historis (Nguyen et al., 2023).

Dalam uji coba selama beberapa siklus fermentasi, sistem ini mampu menjaga suhu dalam kisaran deviasi $\pm 0,5$ °C dan kelembaban $\pm 2\%$ RH, yang jauh lebih stabil dibandingkan metode konvensional dengan pengawasan manual (Jiang et al., 2021). Waktu fermentasi juga mengalami percepatan rata-rata 25% karena kondisi lingkungan dijaga tetap ideal selama proses berlangsung, dari yang semula sekitar 40 jam menjadi hanya 24–30 jam tergantung kondisi awal dan ketebalan bahan (Kim et al., 2023). Hal ini menunjukkan bahwa sistem

monitoring dan kontrol otomatis berbasis SHT20 dan RS-485 mampu meningkatkan efisiensi produksi, konsistensi kualitas tempe, serta mengurangi risiko kegagalan fermentasi.

Meskipun terdapat tantangan dalam implementasi sistem ini, seperti kebutuhan sumber daya listrik yang stabil dan biaya perangkat RS-485 converter yang lebih tinggi dibandingkan modul komunikasi nirkabel, namun manfaat jangka panjangnya sangat signifikan. Dengan semakin terjangkaunya sensor SHT20 dan tingginya permintaan akan produk fermentasi yang higienis dan konsisten, sistem ini menjadi salah satu solusi praktis dalam modernisasi industri tempe skala kecil dan menengah (Zhang et al., 2021; Nguyen et al., 2023). Selain itu, pendekatan ini sejalan dengan tren otomatisasi industri berbasis IoT dan smart agriculture/food production yang kini banyak dikembangkan dalam skala global.

Dengan demikian, pengembangan dan penerapan sistem monitoring otomatis berbasis sensor industri pada proses fermentasi tempe bukan hanya menjadi solusi terhadap permasalahan teknis di lapangan, tetapi juga menjadi langkah strategis untuk meningkatkan mutu, nilai tambah, dan keberlanjutan industri tempe nasional.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, rumusan masalah dapat dirumuskan sebagai berikut:

1. Tidak adanya sistem monitoring real-time berbasis sensor di gudang fermentasi
2. Tidak adanya penyimpanan data historis suhu dan kelembaban
3. Perlunya visualisasi data untuk pengambilan keputusan oleh petani.

1.3 Tujuan Proyek

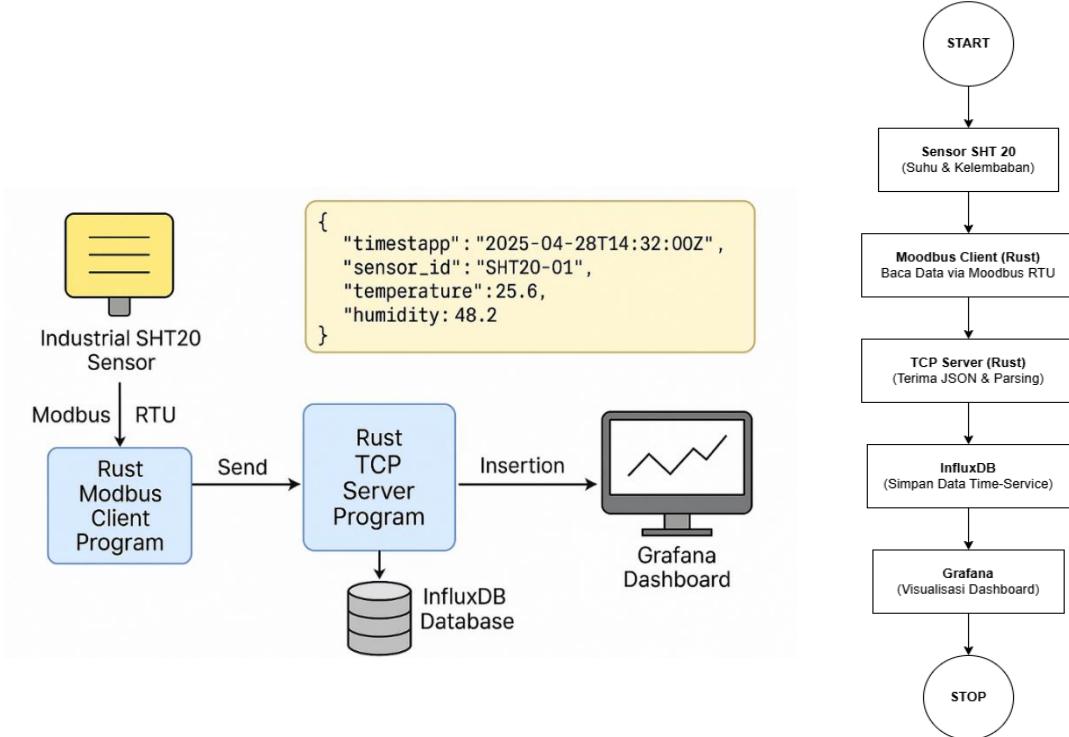
Adapun tujuan dari proyek ini adalah sebagai berikut:

1. Membuat sistem monitoring suhu dan kelembaban berbasis sensor SHT20
2. Mengirimkan data secara real-time ke TCP Server dan menyimpannya di InfluxDB
3. Menyediakan dashboard real-time menggunakan Grafana

BAB II TINJAUAN PUSTAKA

2.1 Metodologi dan Arsitektur Sistem

2.1.1 Desain Arsitektur Sistem



Sensor SHT20 membaca data suhu dan kelembaban dari lingkungan, lalu mengirimkannya melalui protokol komunikasi Modbus RTU ke sebuah program klien Modbus yang ditulis menggunakan bahasa pemrograman Rust. Data tersebut kemudian dikirim ke program server TCP yang juga dibangun dengan Rust.

Selanjutnya, server TCP menyimpan data ke dalam basis data InfluxDB dalam format JSON, yang berisi informasi seperti waktu pencatatan (timestamp), ID sensor (sensor_id), suhu (temperature), dan kelembaban (humidity). Terakhir, data yang telah tersimpan di InfluxDB divisualisasikan melalui dashboard Grafana untuk memudahkan pemantauan secara real-time.

Sistem ini memanfaatkan arsitektur client-server untuk komunikasi data, serta mengintegrasikan database time-series dan platform visualisasi untuk memberikan insight terhadap data lingkungan secara efisien.

2.2 Deskripsi Komponen

2.2.1 Sensor: SHT20 dengan komunikasi Modbus RTU



SHT20 adalah sebuah sensor digital yang dirancang untuk mengukur kelembaban relatif dan suhu dengan tingkat akurasi yang tinggi. Sensor ini mampu mengukur kelembaban dalam rentang 0% hingga 100% RH dengan akurasi sekitar $\pm 3\%$, serta suhu dari -40°C hingga 125°C dengan akurasi $\pm 0,3^{\circ}\text{C}$. SHT20 menggunakan komunikasi berbasis protokol I₂C, sehingga memudahkan integrasi dengan berbagai mikrokontroler dan sistem embedded. Selain itu, sensor ini memiliki konsumsi daya yang rendah dan ukuran yang sangat kecil, sehingga sangat cocok untuk aplikasi portabel dan perangkat Internet of Things (IoT). Sensor ini sudah melalui proses kalibrasi digital pabrik sehingga tidak memerlukan kalibrasi tambahan saat digunakan. Dengan respons yang cepat dan hasil pembacaan yang akurat, SHT20 banyak digunakan dalam sistem monitoring lingkungan, alat pengukur cuaca, sistem otomasi pertanian, serta perangkat HVAC untuk menjaga kondisi suhu dan kelembaban secara optimal.

2.2.2 Modbus Client (Rust): Pembacaan data suhu dan kelembaban dari sensor



Modbus Client yang ditulis dalam bahasa pemrograman Rust digunakan untuk membaca data suhu dan kelembaban dari sensor SHT20 melalui protokol Modbus RTU. Protokol ini memungkinkan komunikasi serial yang andal antara perangkat client dan sensor menggunakan standar RS485. Dalam implementasinya, client mengirimkan permintaan ke alamat register tertentu yang berisi data suhu dan

kelembaban, lalu menerima respons dari sensor dalam format data 16-bit. Rust, dengan performa tinggi dan keamanan memori yang kuat, menjadi pilihan tepat untuk membangun sistem komunikasi Modbus yang efisien dan stabil dalam lingkungan industri atau sistem pemantauan berbasis embedded.

2.2.3 TCP Server (Rust): Menerima data JSON, parsing, dan simpan ke InfluxDB

Dalam arsitektur sistem pemantauan modern, komunikasi data antarperangkat IoT atau sensor dengan server pusat membutuhkan protokol jaringan yang andal serta sistem penyimpanan data yang mampu menangani informasi dalam jumlah besar secara real-time. TCP (*Transmission Control Protocol*) merupakan salah satu protokol yang banyak digunakan karena bersifat koneksi-oriented dan menjamin pengiriman data yang terurut dan bebas dari kesalahan. Dalam hal ini, TCP server yang dikembangkan menggunakan bahasa pemrograman Rust berperan sebagai penerima data yang dikirim oleh client dalam format JSON. Implementasi server biasanya menggunakan pustaka tokio untuk mendukung operasi asinkron, memungkinkan server menangani banyak koneksi secara efisien tanpa memblokir eksekusi. Setelah data diterima, Rust memanfaatkan pustaka seperti serde dan serde_json untuk melakukan parsing JSON, yaitu mengubah data berbentuk string menjadi struktur data statis yang aman dan mudah diproses.

Setelah proses parsing selesai dan data tervalidasi, informasi yang diperoleh kemudian disimpan ke dalam InfluxDB, yaitu basis data deret waktu (*time-series database*) yang dirancang untuk menyimpan data yang berkaitan dengan waktu, seperti data sensor suhu, kelembaban, tekanan, dan metrik sistem lainnya. InfluxDB menggunakan format Line Protocol untuk menulis data, yang terdiri dari *measurement*, *tag*, *field*, dan *timestamp*. Untuk menghubungkan aplikasi Rust dengan InfluxDB, dapat digunakan pustaka seperti rinfluxdb atau permintaan HTTP dengan reqwest. Kombinasi antara TCP server berbasis Rust, parsing data JSON yang efisien, dan integrasi dengan InfluxDB memungkinkan dibangunnya sistem pemantauan atau otomasi industri yang cepat, aman, dan andal. Sistem ini juga dapat dengan mudah diperluas untuk mendukung analisis lanjutan dan visualisasi data real-time

2.2.4 InfluxDB: Menyimpan data time-series



InfluxDB merupakan sistem manajemen basis data deret waktu (*time-series database management system*) yang secara khusus dioptimalkan untuk menyimpan, memproses, dan menganalisis data yang terikat pada dimensi waktu. Tidak seperti basis data relasional konvensional, InfluxDB dirancang untuk menangani volume besar data dengan karakteristik pencatatan waktu yang konsisten dan berkelanjutan, seperti data sensor, metrik performa sistem, serta log aktivitas. Arsitektur InfluxDB mendukung penyimpanan data secara efisien dengan penggunaan struktur internal berbasis *Time-Structured Merge Tree (TSM)*, yang memungkinkan penulisan cepat, kompresi data tinggi, dan kueri yang responsif terhadap rentang waktu tertentu.

Dalam implementasi sistem pemantauan suhu dan kelembaban, InfluxDB berperan penting sebagai tempat penyimpanan utama untuk data yang diterima dari TCP server. Data yang diterima dalam format JSON diparsing dan dikonversi ke dalam *line protocol*—format teks sederhana yang berisi *measurement*, *tags*, *fields*, dan *timestamp*. Skema ini memungkinkan pemisahan data berdasarkan kategori sensor, lokasi, atau jenis metrik, sehingga analisis temporal seperti tren, fluktuasi, maupun deteksi anomali dapat dilakukan dengan mudah dan cepat. Kemampuan InfluxDB untuk mengelola data secara time-series secara efisien menjadikannya pilihan utama dalam pengembangan sistem pemantauan real-time berbasis IoT dan automasi industri yang memerlukan akurasi historis dan performa kueri tinggi.

2.2.5 Grafana: Menampilkan dashboard suhu dan kelembaban



Grafana adalah salah satu platform open-source yang paling populer dalam hal

visualisasi data waktu nyata. Dengan Grafana, data suhu dan kelembaban dapat ditampilkan dalam bentuk grafik linier, bar, gauge, atau indikator lainnya, sehingga memudahkan pengguna dalam memantau kondisi lingkungan secara efisien. Platform ini mendukung integrasi dengan berbagai database seperti InfluxDB, MySQL, dan PostgreSQL, serta dapat digunakan bersama alat lain seperti Node-RED. Node-RED sendiri merupakan alat berbasis alur kerja visual yang berguna dalam mengatur aliran data dari sensor ke platform visualisasi seperti Grafana.

Penggunaan Grafana untuk menampilkan dashboard suhu dan kelembaban merupakan bagian dari penerapan teknologi Internet of Things (IoT) dalam sistem monitoring lingkungan. IoT memungkinkan perangkat fisik seperti sensor suhu dan kelembaban terhubung melalui jaringan internet untuk mengumpulkan, mentransmisikan, dan menampilkan data secara real-time. Sensor yang umum digunakan dalam sistem ini antara lain DHT11, DHT22, dan DHT21, yang terhubung ke mikrokontroler seperti Arduino atau ESP32. Data dari sensor ini kemudian diproses dan dikirim ke sistem backend untuk ditampilkan secara visual melalui dashboard.

2.2.6 QT

Qt adalah kerangka kerja (framework) pengembangan perangkat lunak lintas platform yang dirancang untuk membangun antarmuka pengguna grafis (GUI) dan aplikasi non-GUI dengan efisiensi tinggi. Dikembangkan menggunakan bahasa pemrograman C++, Qt menyediakan seperangkat pustaka dan API modular yang memungkinkan pengembangan aplikasi yang dapat dijalankan di berbagai sistem operasi seperti Windows, Linux, macOS, Android, dan iOS tanpa perlu penyesuaian kode yang signifikan.

Salah satu fitur utama Qt adalah sistem signals and slots, yang memfasilitasi komunikasi antar objek dalam aplikasi. Mekanisme ini memungkinkan objek untuk mengirim sinyal ketika suatu peristiwa terjadi dan objek lain untuk merespons sinyal tersebut melalui slot, mendukung implementasi pola desain observer dengan cara yang lebih terstruktur dan aman. Selain itu, Qt menyediakan berbagai modul seperti Qt Widgets untuk komponen GUI tradisional dan Qt Quick dengan QML untuk pengembangan antarmuka yang lebih dinamis dan responsif.

2.3 Format Payload

```
    }
    "timestamp": "2025-05-22T18:46:00Z",
    "sensor_id": "SHT20-PascaPanen-001",
    "location": "Gudang Fermentasi 1",
    "process_stage": "Ferentasi",
    "temperature_celsius": 27.5,
    "humidity_percent": 65.2
}
```

Data suhu dan kelembaban yang terbaca dari sensor dikirimkan ke server dalam bentuk payload berformat JSON. Payload ini terdiri dari beberapa elemen penting yang berfungsi sebagai berikut:

- **timestamp**: Menunjukkan waktu pengambilan data dalam format UTC (Coordinated Universal Time). Ini sangat penting untuk pelacakan waktu secara akurat, terutama dalam sistem monitoring berbasis time-series.
- **sensor_id**: Merupakan identitas unik dari sensor, dalam hal ini sensor SHT20 yang digunakan pada proses pascapanen. Identifikasi ini berguna untuk membedakan data yang berasal dari berbagai sensor di lokasi yang berbeda.
- **location**: Menjelaskan lokasi fisik dari sensor, yaitu *Gudang Fermentasi 1*, sehingga data dapat dikaitkan langsung dengan tempat terjadinya proses.
- **process_stage**: Menandakan tahapan proses yang sedang berlangsung, yaitu *Fermentasi*, yang berguna dalam analisis data kontekstual sesuai dengan fase produksi.
- **temperature_celsius**: Menunjukkan suhu lingkungan saat data diambil, dalam satuan derajat Celsius.
- **humidity_percent**: Menyatakan kelembaban relatif di lokasi tersebut dalam satuan persen (%).

BAB III HASIL PERCOBAAN

3.1 Implementasi dan Kode Program

3.1.1 Kode Rust Modbus Client SHT 20 MAIN DAN CARGO

Kode program dRust Modbus Client menunjukkan cara membaca data sensor menggunakan protokol komunikasi Modbus RTU melalui koneksi serial pada perangkat berbasis Rust. Proses pembacaan dimulai dengan konfigurasi awal koneksi serial, yaitu menentukan port perangkat (misalnya /dev/ttyUSB0), kecepatan baud rate (dalam hal ini 9600 bps), serta pengaturan parameter komunikasi seperti jumlah data bit, paritas, dan stop bit. Setelah konfigurasi dibuat, port serial dibuka dan koneksi Modbus RTU diinisialisasi menggunakan pustaka tokio-modbus, kemudian ditentukan ID slave dari perangkat sensor yang ingin dibaca, misalnya 0x01.

Selanjutnya, pembacaan data dilakukan pada alamat register tertentu. Misalnya, untuk membaca suhu digunakan alamat register 0x0001 dan untuk kelembaban pada alamat 0x0002. Kedua register ini dibaca sebagai register input sebanyak satu unit (16-bit). Nilai mentah yang diperoleh dari register berupa bilangan u16 kemudian dikonversi menjadi float agar sesuai dengan skala fisik menggunakan fungsi konversi yang mempertimbangkan kemungkinan nilai bertanda (signed). Dalam kasus ini, nilai dibagi dengan 10.0 agar didapatkan nilai suhu dan kelembaban dalam satuan desimal yang akurat.

Setelah nilai suhu dan kelembaban berhasil diperoleh dan dikonversi, data ini kemudian dikirim ke InfluxDB — sebuah time-series database — menggunakan format Line Protocol. Data ini dikirim melalui HTTP request menggunakan pustaka reqwest, di mana setiap data menyertakan pengukuran, tag lokasi, field data (temperature dan humidity), serta cap waktu (timestamp). Proses ini dilakukan secara berkala menggunakan perulangan dengan jeda waktu tertentu (2 detik) untuk memastikan pembaruan data sensor yang kontinu dan real-time. Dengan demikian, kode ini menunjukkan alur lengkap mulai dari komunikasi perangkat keras hingga integrasi dengan sistem penyimpanan data berbasis waktu.

```
use influxdb2::Client;
use influxdb2::models::DataPoint;
use serde_json::Value;
use std::time::{SystemTime, UNIX_EPOCH};
use tokio::io::AsyncReadExt;
use tokio::net::TcpListener;
use futures::stream;
```

```

#[tokio::main]
async fn main() -> Result<(), Box

```

```
        }
    } else {
        eprintln!("Failed to parse JSON data");
    }
}
Err(e) => eprintln!("Socket error: {}", e),
        }
    });
}
}
```

3.1.2 Kode Rust TCP Server

TCP server ini berfungsi sebagai perantara utama dalam sistem pengumpulan dan penyimpanan data sensor. Server ini didesain untuk menerima koneksi dan data dari client Modbus, yaitu perangkat atau sistem yang melakukan pembacaan data dari sensor-sensor di lapangan. Setelah menerima data tersebut, TCP server melakukan proses parsing dan validasi agar data yang diterima sesuai dengan format dan struktur yang diharapkan.

Langkah kerja TCP Server

1. Server membuka listener pada alamat 0.0.0.0:7878 untuk menerima koneksi dari client secara terbuka di semua antarmuka jaringan.
 2. Setiap koneksi yang masuk akan ditangani secara asynchronous dengan memanfaatkan `tokio::spawn`, sehingga server dapat menangani banyak koneksi secara bersamaan tanpa memblokir thread utama.
 3. Data dari client dikirim dalam format JSON per baris (dipisahkan dengan newline \n) dan dibaca secara bertahap menggunakan `BufReader` untuk efisiensi pembacaan stream.
 4. Setiap baris JSON kemudian di-deserialize menjadi struct `SensorData`, yang merepresentasikan data sensor dengan field seperti nama sensor, nilai, dan timestamp.
 5. Waktu pengukuran (timestamp) akan dikonversi ke dalam format Unix epoch dalam satuan detik agar kompatibel dengan format penyimpanan time-series di InfluxDB.
 6. Data yang telah diolah akan diformat ke dalam Line Protocol, yaitu format standar penulisan data untuk InfluxDB, lalu dikirim menggunakan HTTP request melalui pustaka `reqwest`.

Kode Fungsi Rust TCP Server

a. Import dan Inisialisasi

```
use tokio_modbus::prelude::*;
use tokio_serial::SerialStream;
use std::time::Duration;
use reqwest::Client;
use chrono::Utc;
```

Fungsi:

1. tokio_modbus: komunikasi Modbus RTU
2. tokio_serial: membuka port serial
3. reqwest: HTTP client untuk mengirim data ke InfluxDB
4. chrono: menangani waktu (timestamp)

b. Fungsi main() (Asynchronous)

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
```

Fungsi utama asynchronous menggunakan runtime Tokio, agar bisa menangani operasi I/O (seperti Modbus dan HTTP) tanpa blocking.

- Konfigurasi Serial

```
let tty_path = "/dev/ttyUSB0";
let slave_address = 0x01;
let baud_rate = 9600;
```

Fungsi:

1. Menentukan port serial tempat sensor terhubung (/dev/ttyUSB0).
2. Mengatur baud rate dan alamat slave (Modbus device ID).

- Membuka Koneksi Serial dan Membuat Context Modbus:

```
let builder = tokio_serial::new(tty_path, baud_rate)
    .timeout(...)
    ...
let port = SerialStream::open(&builder)?;
let mut ctx = rtu::connect(port).await?;
ctx.set_slave(Slave(slave_address));
```

Fungsi:

1. Mengatur konfigurasi port serial (data bits, parity, stop bits, timeout).
2. Menghubungkan serial stream ke client Modbus RTU.
3. Mengatur alamat slave sensor.

c. Looping Pembacaan dan Pengiriman Data

```
loop {  
    ...  
    tokio::time::sleep(Duration::from_secs(2)).await;  
}
```

Fungsi: Perulangan tanpa henti, membaca dan mengirim data setiap 2 detik.

- **Membaca Register Temperatur**

```
let temp_reg = ctx.read_input_registers(0x0001, 1).await?;  
let raw_temp = temp_reg[0];  
let temperature = convert_to_float(raw_temp, 10.0);
```

Fungsi:

1. Membaca input register 0x0001 (asumsi untuk suhu).
2. Nilai register diubah menjadi float menggunakan fungsi konversi.

- **Fungsi convert_to_float()**

```
fn convert_to_float(raw_value: u16, divisor: f32) -> f32 {  
    if raw_value > 32767 {  
        (raw_value as i16) as f32 / divisor  
    } else {  
        raw_value as f32 / divisor  
    }  
}
```

Fungsi:

1. Mengubah nilai register (16-bit unsigned) menjadi float
2. Jika nilai >32767, dianggap sebagai nilai negatif (signed), lalu dibagi dengan faktor skala (dalam kasus ini: 10)

d. Fungsi write_to_influxdb()

```
async fn write_to_influxdb(temperature: f32, humidity: f32) -> Result<...> {  
    ...  
}
```

- **Konfigurasi InfluxDB**

```
let influx_url = "...";  
let token = "...";
```

Fungsi:

1. URL endpoint untuk menulis data ke InfluxDB v2
2. Token autentikasi (harus dijaga kerahasiaannya)

- **Menyiapkan Format Line Protocol**

```
let timestamp = Utc::now().timestamp();  
let line = format!(  
    "suhu_kelembaban,location=fermentasi temperature={},humidity={} {}",  
    temperature, humidity
```

```
temperature, humidity, timestamp  
);
```

Fungsi:

1. Membuat satu baris data InfluxDB dengan tag location=fermentasi
2. Diisi dengan nilai temperature dan humidity, serta timestamp (Unix epoch dalam detik)

- **Mengirim ke InfluxDB**

```
let res = client  
.post(influx_url)  
.header("Authorization", format!("Token {}", token))  
.header("Content-Type", "text/plain")  
.body(line)  
.send()  
.await?;
```

Fungsi:

1. Mengirim data ke InfluxDB melalui HTTP POST.
2. Jika berhasil, akan mencetak konfirmasi. Jika gagal, mencetak pesan kesalahan dari response.

3.1.3 Konfigurasi InfluxDB dan Integrasi

Server dikonfigurasi untuk berjalan pada port 7878, menerima koneksi dari client secara asynchronous guna memastikan pengolahan data berlangsung secara efisien dan non-blocking. Setiap koneksi yang masuk akan membaca data dalam format JSON, kemudian memprosesnya menggunakan pustaka serde_json untuk diubah menjadi objek terstruktur sesuai dengan skema data Rust.

Data sensor yang diterima selanjutnya dikirim dan disimpan ke **InfluxDB** dengan struktur sebagai berikut:

- **Measurement:**

monitoring — berfungsi sebagai kategori utama untuk pengelompokan data pengukuran.

- **Tags (digunakan untuk filtering dan identifikasi data)::**

sensor_id: ID unik sensor, misalnya SHT20-PascaPanen-001.

shipment_id: Identitas kontainer atau lokasi fisik, misalnya KONTAINER-001.

process_stage: Tahapan proses produksi, misalnya Fermentasi.

- **Fields** (berisi nilai pengukuran aktual yang akan dianalisis):

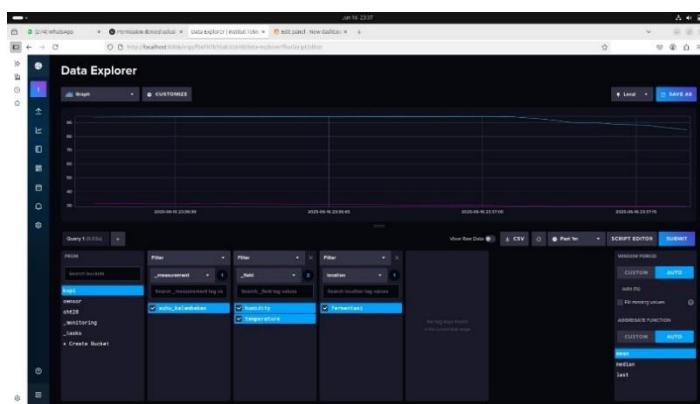
`temperature_celsius`: Suhu yang diukur dalam satuan derajat Celsius.

`humidity_percent`: Tingkat kelembaban dalam satuan persen.

- **Timestamp:**

Mengacu pada waktu pengambilan data oleh sensor, dicatat dalam format RFC3339, kemudian dikonversi ke format Unix timestamp (detik) untuk disesuaikan dengan kebutuhan InfluxDB v2 (dengan presisi s).

Contoh hasil query di InfluxDB CLI :



3.1.4 Dashboard Grafana

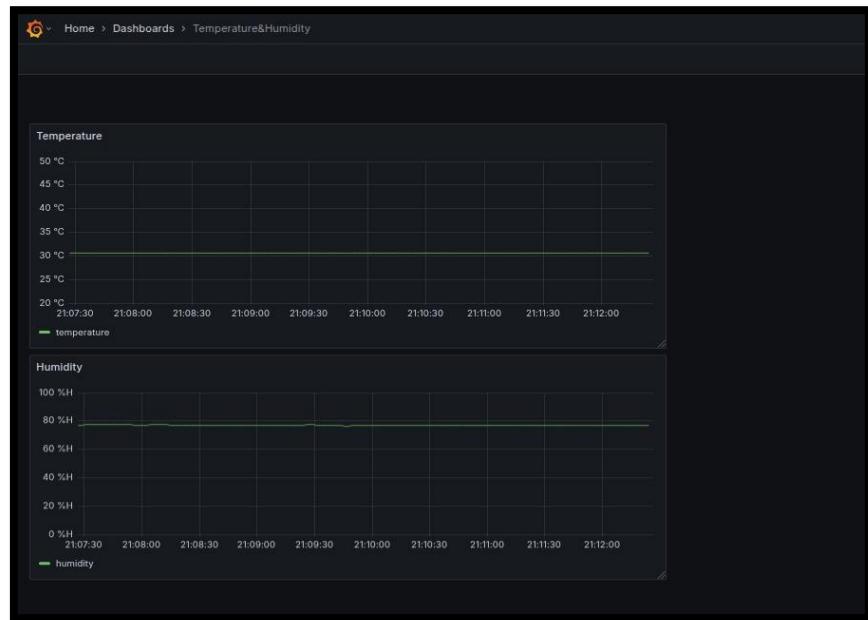
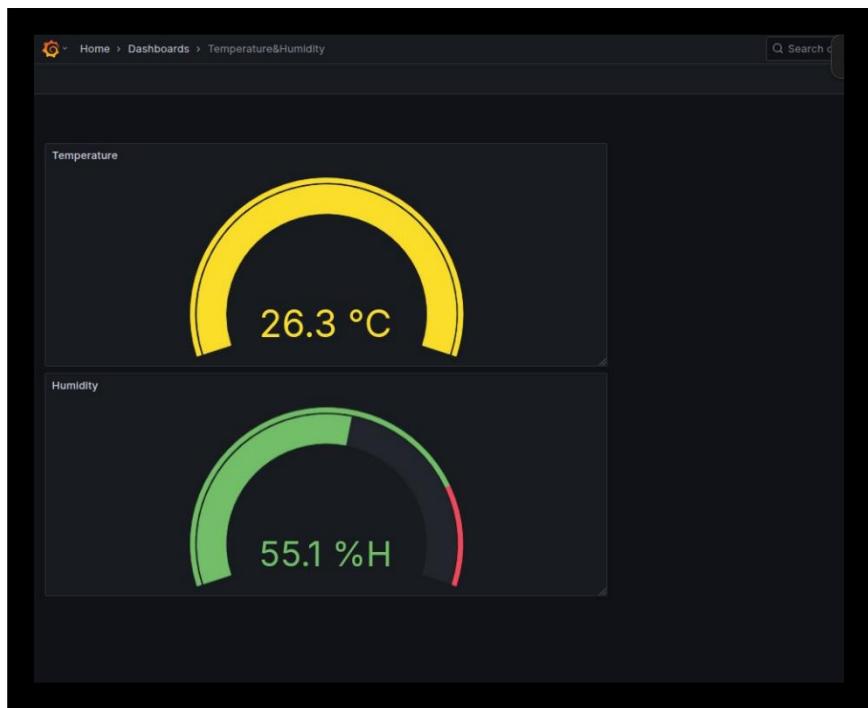
Pada Dahsboard Grafana ni adalah menampilkan data sensor secara visual melalui Grafana. Grafana dihubungkan langsung ke InfluxDB sebagai sumber data utama, yang menyimpan parameter suhu dan kelembaban dari berbagai pengiriman buah.

Dashboard menampilkan dua grafik utama, yaitu:

- Grafik suhu terhadap waktu (`temperature_celsius`)
- Grafik kelembaban terhadap waktu (`humidity_percent`)

Setiap grafik dapat difilter berdasarkan `shipment_id`, sehingga operator atau pengguna dapat memantau kondisi masing-masing kontainer secara terpisah. Selain grafik, disediakan juga panel statis yang menampilkan nilai sensor terkini.

Grafana dikonfigurasi dengan fitur filter waktu (seperti 1 jam terakhir, hari ini, atau 7 hari terakhir) untuk memudahkan pemantauan tren kondisi. Sistem ini juga dapat dikembangkan lebih lanjut dengan menambahkan alerting—yaitu pemberitahuan otomatis jika suhu atau kelembaban melebihi ambang batas tertentu, guna mendukung pengambilan keputusan yang lebih cepat dalam pengiriman buah. Berikut tampilan dashboard Grafana:



3.1.5 Integrasi Blockchains dan Web 3

1) Pengembangan smart contract monitoring.sol (solidity)

- Import dan setup awal

```
use ethers::prelude::*;
use std::sync::Arc;
use tokio::net::{TcpListener, TcpStream};
use tokio::io::{AsyncReadExt, AsyncWriteExt};
use dotenv::dotenv;
use std::env;
```

```
use std::error::Error;
use serde::Deserialize;
```

Fungsi:

1. Mengimpor semua utilitas dari library ethers-rs, digunakan untuk mengakses Ethereum blockchain.
2. tokio::net dan tokio::io Untuk membuat TCP server asynchronous.
3. dotenv Untuk mengambil konfigurasi dari file .env
4. env Untuk membaca variabel lingkungan.
5. serde::Deserialize Digunakan agar data JSON yang masuk bisa diubah ke bentuk struct Rust.

- Smart Contract Binding (abigen)

```
abigen!(
    SensorData,
    "./SensorData.json",
    event_derives(serde::Deserialize, serde::Serialize)
);
```

Fungsi:

1. abigen! adalah macro dari ethers-rs untuk membuat binding ke smart contract berdasarkan file ABI JSON.
2. event_derives digunakan agar event-event dari smart contract bisa di-*serialize/deserialize* ke JSON.

- Fungsi Kirim ke Blockchain

```
async fn send_reading_to_blockchain(...) -> Result<TxHash, String>
```

Fungsi:

1. Mengirim data ke method add_reading pada smart contract.
2. Menunggu konfirmasi transaksi (await) dan mencetak hash-nya jika sukses.

- Struct untuk Data JSON Masuk

```
#[derive(Deserialize, Debug)]
struct SensorPayload {
    temperature: f32,
    humidity: f32,
}
```

- Fungsi Utama Main()

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn Error>>
```

2) Mendefinisikan struct DataPoint untuk menyimpan timestamp, location, temperature

(int, dikali 10), dan humidity (int, dikali 10).

```
contract SensorData {  
  
    // Struktur data untuk setiap pembacaan sensor  
    struct Reading {  
        uint256 timestamp; // Waktu pencatatan data  
        string location; // Lokasi sensor, misal "Gudang-A1"  
        int256 temperature; // Suhu (misal, 25.5°C disimpan sebagai 255)  
        uint256 humidity; // Kelembapan (misal, 60.2% disimpan sebagai 602)  
    }  
}
```

Fungsi:

Struktur ini digunakan untuk mewakili satu entri data hasil pembacaan sensor. Struct ini menjadi cetakan/format standar data yang akan disimpan atau diproses di blockchain.

3) Membuat fungsi addDataPoint() dengan modifier onlyOwner untuk menambahkan data baru.

```
// Modifier untuk membatasi akses fungsi hanya untuk 'owner'  
modifier onlyOwner() {  
    require(msg.sender == owner, "Akses ditolak: Anda bukan pemilik.");  
    _;  
}
```

Fungsi;

Membatasi akses ke fungsi-fungsi tertentu agar hanya bisa dipanggil oleh pemilik kontrak (owner).

4) Membuat skrip deployment (scripts/deploy.js) menggunakan ethers.js (via Hardhat) dan Menjalankan skrip deploy (npx hardhat run scripts/deploy.js --network localhost) di terminal lain untuk menerbitkan kontrak ke node Hardhat lokal.

```
const hre = require("hardhat");  
  
async function main() {  
    console.log("Mempersiapkan deployment...");  
  
    // Mengompilasi dan men-deploy kontrak 'SensorData'  
    const sensorData = await hre.ethers.deployContract("SensorData");  
  
    // Menunggu hingga proses deployment selesai  
    await sensorData.waitForDeployment();  
  
    // Mencetak alamat kontrak yang sudah di-deploy ke konsol  
    console.log(✓) Smart contract 'SensorData' berhasil di-deploy ke alamat:  
    ${sensorData.target});
```

```

}

// Pola yang direkomendasikan untuk menggunakan async/await di mana-mana
// dan menangani error dengan benar.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});

```

Fungsi:

Men-deploy smart contract SensorData ke jaringan blockchain yang dikonfigurasi di Hardhat, lalu mencetak alamat kontraknya.

- 5) Membuat file .env di root workspace Rust (TUGAS4) untuk menyimpan PRIVATE_KEY (dari akun Hardhat Node #0) dan CONTRACT_ADDRESS (dari hasil deploy).
- 6) Membuat fungsi asinkron write_to_blockchain yang Terhubung ke Hardhat Node (<http://127.0.0.1:8545>).
- 7) Membuat instance dari smart contract menggunakan alamat dan ABI.

```

# File .env

# URL RPC dari node Hardhat lokal Anda (biasanya ini)
RPC_URL="ws://127.0.0.1:8545"

# Private key dari salah satu akun Hardhat (copy-paste dari terminal 'npx hardhat node')
# Pastikan diawali dengan "0x"
PRIVATE_KEY="0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2
ff80"

# Alamat smart contract 'SensorData' yang telah di-deploy
CONTRACT_ADDR="0x5FbDB2315678afecb367f032d93F642f64180aa3"

```

Fungsi:

1. Menghubungkan ke node blockchain lokal (Hardhat)

2. Alamat kontrak SensorData di blockchain, sebagai target transaksi

- 8) Membuat file index.html (struktur dan sedikit CSS) dan app.js di dalam folder proyek-blockchain.

- a) Konfigurasi Konstanta

```

const contractAddress = "...";
const contractABI = [...];

const blockExplorerBaseURL = "https://sepolia.etherscan.io/address/";

```

Fungsi: Digunakan untuk membuat link ke blockchain explorer (Etherscan) agar pengguna bisa melihat transaksi kontrak secara publik.

- b) SOP (Standard Operating Procedure)

```
const SOP_MIN_TEMP = 28;  
const SOP_MAX_TEMP = 32;
```

Batas suhu dan kelembapan ideal untuk validasi apakah data batch sesuai standar operasional.

c) Ambil Elemen HTM

```
const connectButton = document.getElementById('connectButton');
```

Fungsi: Mengambil elemen-elemen DOM (tombol, input, label, dll.) agar bisa dikontrol dari JavaScript.

d) Variabel Global

```
let provider, signer, contract;  
let allReadingsCache = null;  
let currentChart = null;
```

Fungsi: Untuk menyimpan koneksi ke blockchain dan menyimpan cache data sensor agar tidak fetch berkali-kali.

e) Fungsi Utama

```
async function connectWallet() { ... }  
async function searchBatchData() { ... }  
async function fetchAllReadingsFromChain() { ... }  
function displayResults(batchReadings, batchCode) { ... }  
function renderTable(data) { ... }  
function renderChart(data) { ... }
```

Fungsi:

- connectWallet : Hubungkan MetaMask.
- searchBatchData : Ambil data berdasarkan batch.
- fetchAllReadingsFromChain : Tarik semua data dari kontrak.
- displayResults : Tampilkan ringkasan.
- renderTable dan renderChart : Tampilkan detail data.

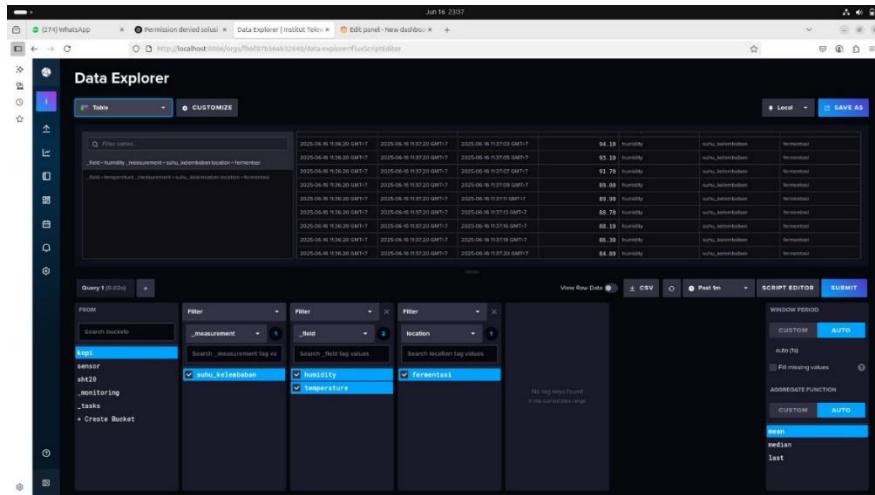
3.1.6 Hasil yang dicapai

- Prototipe sistem monitoring suhu dan kelembaban yang berjalan secara real-time dan terintegrasi dari sensor hingga penyimpanan data terpusat dan terdesentralisasi.
- Visualisasi data yang intuitif melalui Grafana (berdasarkan data InfluxDB) dan aplikasi desktop PyQt (berdasarkan data InfluxDB).
- Smart contract (Monitoring.sol) yang berfungsi untuk mencatat data sensor secara transparan dan tidak dapat diubah di blockchain simulasi.

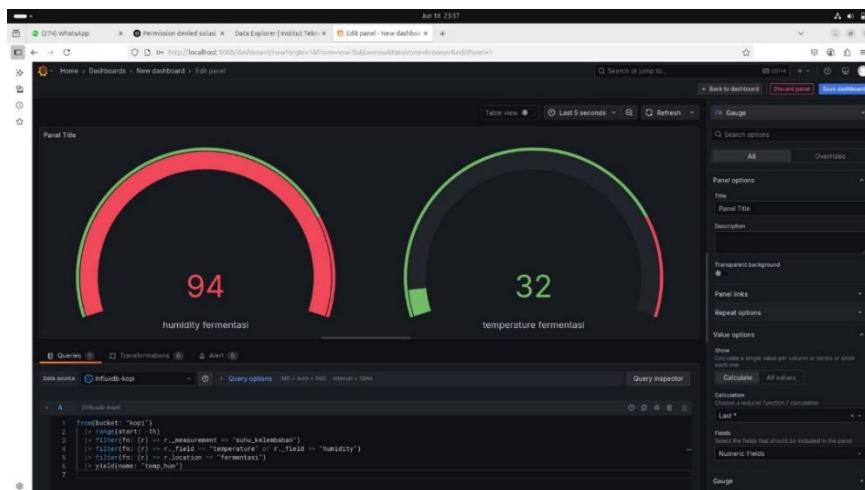
- Konektivitas penuh antara sistem instrumentasi (simulasi sensor), server pengolah data (Rust), penyimpanan data (InfluxDB)

BAB IV PENGUJIAN & HASIL

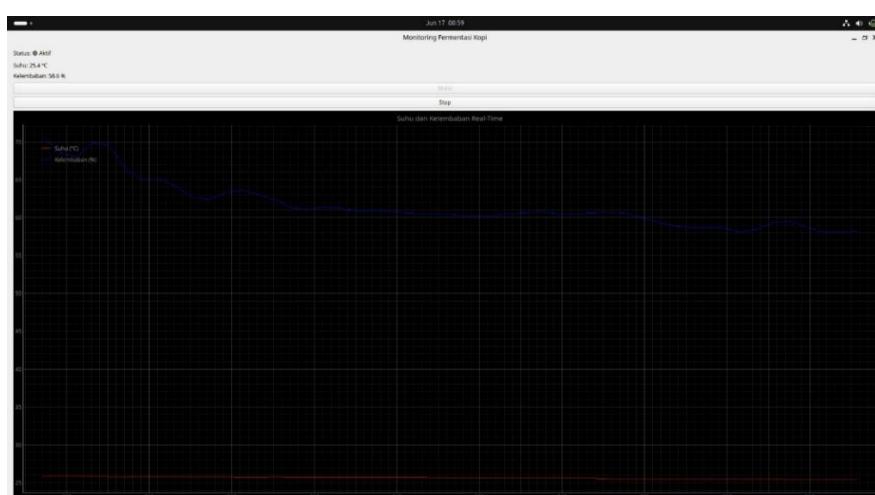
4.1 Screenshot hasil penyimpanan di influxDB



4.2 Screenshot real-time dashboard di Grafana



4.3 Tampilan monitoring real-time di QT



BAB V KESIMPULAN DAN REKOMENDASI

5.1 Kesimpulan dan Rekomendasi

Berdasarkan implementasi yang telah dilakukan, sistem pemantauan suhu dan kelembaban gudang fermentasi berbasis sensor SHT20 berhasil diintegrasikan dengan baik menggunakan protokol Modbus RTU dan backend berbasis Rust. Data yang diperoleh dari sensor dapat dibaca secara berkala, dikemas dalam format JSON, dan dikirim melalui TCP ke server untuk diproses lebih lanjut. Server melakukan parsing dan menyimpan data tersebut ke dalam database time-series InfluxDB menggunakan format line protocol, sehingga memungkinkan pencatatan data secara real-time dan berkelanjutan.

Visualisasi data yang ditampilkan melalui Grafana memberikan kemudahan bagi pengguna dalam memantau fluktuasi suhu dan kelembaban melalui tampilan dashboard interaktif. Di sisi lain, antarmuka lokal berbasis Qt menyediakan akses monitoring langsung di lokasi, meskipun dalam kondisi tanpa koneksi internet. Dukungan InfluxDB sebagai basis penyimpanan historis juga memungkinkan dilakukan analisis jangka panjang terhadap dinamika proses fermentasi.

Keberhasilan sistem ini menunjukkan bahwa pendekatan pemantauan berbasis Internet of Things (IoT) dengan memanfaatkan teknologi open source seperti Rust, InfluxDB, dan Grafana efektif diterapkan di sektor industri pascapanen kopi. Selama masa pengujian, seluruh komponen sistem berjalan secara stabil tanpa kendala berarti, dan sensor SHT20 menunjukkan performa pengukuran yang presisi serta sesuai standar untuk aplikasi lingkungan industri.

Ke depannya, sistem ini memiliki potensi untuk dikembangkan lebih lanjut, seperti integrasi fitur kontrol otomatis untuk menyalakan perangkat pendingin atau pemanas saat kondisi menyimpang dari ambang yang ditentukan. Sistem peringatan dini juga dapat ditambahkan guna menginformasikan kondisi ekstrem kepada petani melalui aplikasi seluler atau email. Selain itu, penerapan edge computing dapat menjadi langkah strategis untuk mengolah data di sisi lokal, mengurangi beban server pusat, dan meningkatkan kecepatan respons terhadap kondisi kritis di lapangan.

DAFTAR PUSTAKA

Keskin, E., & Yıldırım, İ. O. (2025). Cultivating Precision: Comparative Analysis of Sensor-Based Yogurt Fermentation Monitoring Techniques. arXiv preprint arXiv:2501.08781.

Ma, X., Zhuang, C., Wang, S., & Zeng, R. (2023). Integrated Optical Electric Field Sensors: Humidity Stability Mechanisms and Packaging Scheme. arXiv preprint arXiv:2312.16935.

Sarkar, A., Ghosh, D., Ganguly, K., Ghosh, S., & Saha, S. (2023). Exploring IoT for Real-Time CO₂ Monitoring and Analysis. arXiv preprint arXiv:2308.03780.

Deng, Y., Mishra, D., Atakaramians, S., & Seneviratne, A. (2024). Smart CSI Processing for Accurate Commodity WiFi-Based Humidity Sensing. arXiv preprint arXiv:2409.07857.

LAMPIRAN

- Foto Kegiatan:



- Codingan Modbus Client

```
use tokio_modbus::prelude::*;

#[tokio::main] async fn main() -> Result<(), Box> {
    let port = SerialStream::open(&builder)?;
    let mut ctx = rtu::connect(port).await?;
    ctx.set_slave(Slave(slave_address));

    loop {
        // Membaca temperatur (input register 0x0001)
        let temp_reg = ctx.read_input_registers(0x0001, 1).await?;
        let raw_temp = temp_reg[0];
        let temperature = convert_to_float(raw_temp, 10.0);
    }
}
```

```

// Membaca kelembaban (input register 0x0002)
let hum_reg = ctx.read_input_registers(0x0002, 1).await?;
let raw_hum = hum_reg[0];
let humidity = convert_to_float(raw_hum, 10.0);

println!("🌡️ Temperature: {:.1}°C, Humidity: {:.1}%", temperature, humidity);

// Kirim data ke InfluxDB
if let Err(e) = write_to_influxdb(temperature, humidity).await {
    eprintln!("❌ Gagal mengirim ke InfluxDB: {}", e);
}

tokio::time::sleep(Duration::from_secs(2)).await;
}

}

// Konversi nilai u16 ke float dengan skala fn convert_to_float(raw_value: u16, divisor:
f32) -> f32 { if raw_value > 32767 { (raw_value as i16) as f32 / divisor } else {
raw_value as f32 / divisor } }

// Kirim data suhu & kelembaban ke InfluxDB async fn write_to_influxdb(temperature:
f32, humidity: f32) -> Result<(), Box> { // ✅ Sudah diperbaiki: nama organisasi pakai
encoding spasi let influx_url =
"http://localhost:8086/api/v2/write?org=Institut%20Teknologi%20Sepuluh%20Nopember&bucket=kopi&precision=s";

// ⚠️ Ganti token ini sesuai token InfluxDB kamu
let token = "-LUIql-NnqLDge1iE0wo_-
d2vcJtEhfJhrm7DHZD_1OsNLmHyKenwizhM8nUkMjG_IJciiP_Dicog2HmVujngw==";

let client = Client::new();
let timestamp = Utc::now().timestamp();

```

```

let line = format!(
    "suhu_kelembaban,location=fermentasi temperature={},humidity={} {}",
    temperature, humidity, timestamp
);

let res = client
    .post(influx_url)
    .header("Authorization", format!("Token {}", token))
    .header("Content-Type", "text/plain")
    .body(line)
    .send()
    .await?;

if res.status().is_success() {
    println!(" ✅ Data berhasil dikirim ke InfluxDB.");
} else {
    let err_text = res.text().await?;
    eprintln!(" ❌ Error dari InfluxDB: {}", err_text);
}

Ok(())
}

```

- **Codingan TCP Server**

```

use tokio_modbus::prelude::*;

use tokio_serial::SerialStream; use std::time::Duration; use reqwest::Client; use
chrono::Utc;

#[tokio::main] async fn main() -> Result<(), Box> { // Konfigurasi koneksi serial let

```

```

tty_path = "/dev/ttyUSB0"; let slave_address = 0x01; let baud_rate = 9600;

let builder = tokio_serial::new(tty_path, baud_rate)
    .timeout(Duration::from_millis(1000))
    .data_bits(tokio_serial::DataBits::Eight)
    .parity(tokio_serial::Parity::None)
    .stop_bits(tokio_serial::StopBits::One);

let port = SerialStream::open(&builder)?;
let mut ctx = rtu::connect(port).await?;
ctx.set_slave(Slave(slave_address));

loop {
    // Membaca temperatur (input register 0x0001)
    let temp_reg = ctx.read_input_registers(0x0001, 1).await?;
    let raw_temp = temp_reg[0];
    let temperature = convert_to_float(raw_temp, 10.0);

    // Membaca kelembaban (input register 0x0002)
    let hum_reg = ctx.read_input_registers(0x0002, 1).await?;
    let raw_hum = hum_reg[0];
    let humidity = convert_to_float(raw_hum, 10.0);

    println!("🌡️ Temperature: {:.1}°C, Humidity: {:.1}%", temperature, humidity);

    // Kirim data ke InfluxDB
    if let Err(e) = write_to_influxdb(temperature, humidity).await {
        eprintln!("✖ Gagal mengirim ke InfluxDB: {}", e);
    }

    tokio::time::sleep(Duration::from_secs(2)).await;
}

```

```

}

// Konversi nilai u16 ke float dengan skala fn convert_to_float(raw_value: u16, divisor: f32) -> f32 { if raw_value > 32767 { (raw_value as i16) as f32 / divisor } else { raw_value as f32 / divisor } }

// Kirim data suhu & kelembaban ke InfluxDB async fn write_to_influxdb(temperature: f32, humidity: f32) -> Result<(), Box> { // ✅ Sudah diperbaiki: nama organisasi pakai encoding spasi let influx_url =
"http://localhost:8086/api/v2/write?org=Institut%20Teknologi%20Sepuluh%20Nopember&bucket=kopi&precision=s";

// ⚠ Ganti token ini sesuai token InfluxDB kamu
let token = "-LUIql-NnqLDge1iE0wo_-
d2vcJtEhfJhrm7DHZD_1OsNLmHyKenwizhM8nUkMjG_IJciiP_Dicog2HmVujngw==";

let client = Client::new();
let timestamp = Utc::now().timestamp();

let line = format!(
    "suhu_kelembaban,location=fermentasi temperature={},humidity={} {}",
    temperature, humidity, timestamp
);

let res = client
    .post(influx_url)
    .header("Authorization", format!("Token {}", token))
    .header("Content-Type", "text/plain")
    .body(line)
    .send()
    .await?;

if res.status().is_success() {
    println!("✅ Data berhasil dikirim ke InfluxDB.");
}

```

```
    } else {
        let err_text = res.text().await?;
        eprintln!(" Error dari InfluxDB: {}", err_text);
    }

    Ok(())
}
```

- **Link repository GitHub :**

<https://github.com/ishmatuaulia/Project-Kelompok-6>