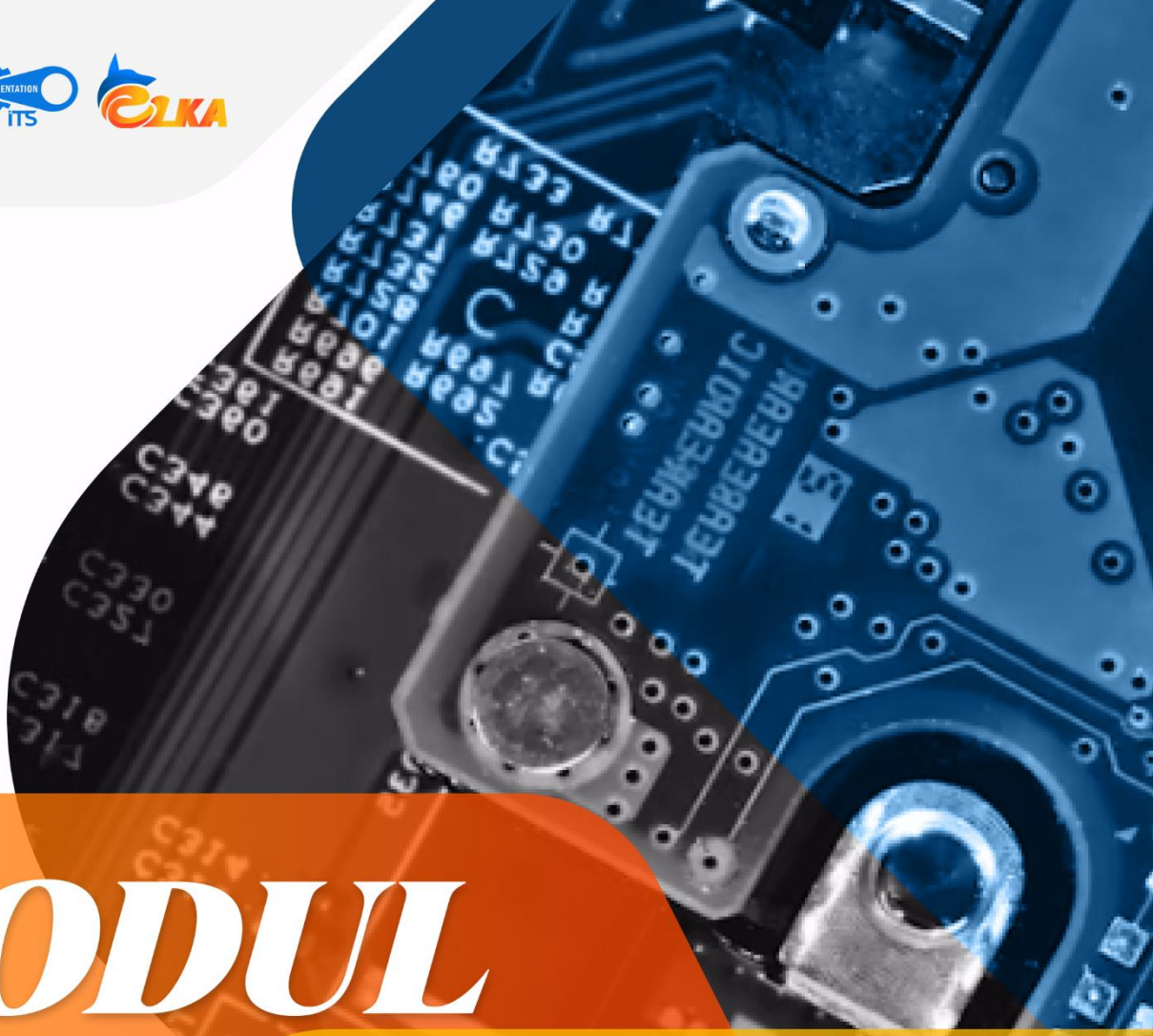


MODUL PRAKTIKUM

VI231418

INTERKONEKSI SISTEM INSTRUMENTASI

Departemen Teknik Instrumentasi
Fakultas Vokasi
Institut Teknologi Sepuluh Nopember
2025

Disusun oleh:
Laboratorium Elektronika
Teknik Instrumentasi
Institut Teknologi Sepuluh Nopember
2025

DAFTAR ISI

TIM PENYUSUN MODUL PRAKTIKUM

Program Studi	: Teknologi Rekayasa Instrumentasi
Departemen	: Teknik Instrumentasi
Fakultas	: Vokasi
Institusi	: Institut Teknologi Sepuluh Nopember
Semester	: 4
Mata Kuliah	: Interkoneksi Sistem Instrumentasi
Kode Matkul	: VI231418
Tahun Ajaran	: 2025/2026

DOSEN PENGAMPU

No	Nama	NIP/NIDN	Peran
1	Ahmad Radhy, S.Si., M.Si	0013118906	Dosen Pengampu
2	Ir. Dwi Oktavianto Wahyu Nugroho, S.T., M.T.	0008108306	Dosen Pengampu
3	Muhammad Roy Ashiddiqi, S.T., M.T.	199301312024061001	Dosen Pengampu

TIM PENYUSUN

No	Nama	NRP	Peran
1	Faradilla Damayanti	2042221004	Penyusun
2	Muhammad Mishbahul Huda	2042221008	Penyusun
3	Reza Akbar Azary	2042221033	Penyusun
4	Mukhamad Da'ul Azimi	2042221071	Penyusun
5	Nabilla Nurul Pratiwi	2042221083	Penyusun
6	Muhammad Ivan Hermawan	2042221096	Penyusun
7	M Dwi Aswangga Azhari	2042221103	Penyusun
8	Akbar Pria Agung Sukarno	2042221105	Penyusun
9	Alif Devintia Pratiwi Hariyanti	2042221139	Penyusun
10	Galuh Pandu Satrio	2042231019	Penyusun
11	Akhmad Maulvin Nazir Zakaria	2042231028	Penyusun
12	Ahmad Malikul Karim Amrullah	2042231041	Penyusun
13	Lusty Hanna Isyajidah	2042231045	Penyusun
14	Wildan Rizki Auzay	2042231061	Penyusun
15	Naufal Faqiih Ashshiddiq	2042231068	Penyusun

TUJUAN UMUM PRAKTIKUM

Mata kuliah Interkonesi Sistem Instrumentasi berada di semester 4 dengan bobot 3 SKS yang terdiri dari 2 SKS TEORI dan 1 SKS PRAKTIKUM, dimana mata kuliah ini mempunyai tujuan pembelajaran (CPL MK):

CAPAIAN PEMBELAJARAN MATA KULIAH (CP-MK) BERDASARKAN KEGIATAN PRAKTIKUM	
1	Mahasiswa mampu memahami prinsip komunikasi data instrumentasi menggunakan Modbus RTU.
2	Mahasiswa mampu melakukan akuisisi data sensor melalui pembacaan register Modbus dan mengonversi data mentah menjadi satuan fisik.
3	Mahasiswa mampu membentuk dan menstandarkan format data menggunakan struktur data dan JSON untuk pertukaran data antarsistem.
4	Mahasiswa mampu mengirim data hasil akuisisi ke sistem lain melalui komunikasi jaringan TCP.
5	Mahasiswa mampu membangun TCP listener untuk menerima data, melakukan parsing serta menangani kesalahan komunikasi/format data.
6	Mahasiswa mampu menyimpan data sensor ke database time-series InfluxDB.
7	Mahasiswa mampu memahami konsep dasar smart contract dan menerapkan pencatatan data melalui event pada jaringan Ethereum lokal.
8	Mahasiswa mampu mengintegrasikan sistem end-to-end dan melakukan validasi hasil melalui antarmuka web/Web3.

Untuk menunjang tercapainya CPL MK interkoneksi sistem instrumentasi, maka MODUL praktikum Interkoneksi Sistem Instrumentasi, disusun untuk dapat memberikan pemahaman kepada mahasiswa tentang cara kerja **Web3, Smartcontract,** serta **Blockchain**. Pelaksanaan praktikum ini umumnya memerlukan beberapa komponen utama seperti Laptop, Sensor, dan module RS485.

Untuk dapat mengukur tingkat pemahaman mahasiswa, pada praktikum ini dilengkapi dengan pertanyaan terkait dengan:

- Dasar teori penunjang praktikum
- Gambar ilustrasi rangkaian
- Tahapan dan metodologi percobaan

Proses evaluasi praktikum dilakukan melalui **laporan praktikum** dalam bentuk *soft file* yang berisi hasil percobaan dan jawaban terhadap pertanyaan dalam modul praktikum.

Praktikum 1

ESP32-S3 ModBus RTU

I. DASAR TEORI

1.1 ModBus Client

Modbus merupakan protokol komunikasi yang banyak digunakan dalam sistem otomasi industri untuk pertukaran data antara perangkat seperti PLC, sensor, dan aktuator. Dalam versi Modbus TCP, istilah *Modbus Client* mengacu pada perangkat atau program yang berperan aktif dalam memulai komunikasi, seperti membaca data dari atau menulis data ke *Modbus Server*.

Dalam hal ini, bahasa pemrograman Rust menjadi pilihan yang menarik untuk mengembangkan Modbus Client karena memiliki keunggulan dalam hal keamanan memori, kecepatan eksekusi, dan efisiensi dalam pengelolaan proses secara bersamaan (asinkron).

Pengembang dapat membangun aplikasi Modbus Client yang andal dan efisien untuk kebutuhan industri, seperti sistem pemantauan jarak jauh, pengiriman data dari perangkat ke cloud, atau integrasi antar perangkat di lingkungan industri. Penggunaan Rust dalam pengembangan Modbus Client memberikan jaminan kestabilan dan kinerja tinggi, yang sangat penting dalam sistem industri yang menuntut kecepatan dan keandalan komunikasi data secara real-time.

1.2 TCP Server

TCP Server dalam pemrograman Rust adalah program yang bertugas menerima dan merespons koneksi dari client menggunakan protokol TCP, yang menjamin data dikirim secara utuh dan berurutan. Rust sangat cocok untuk membuat TCP server karena punya sistem manajemen memori yang aman tanpa garbage collector, sehingga server lebih stabil dan bebas dari bug seperti crash atau data race. Selain itu, Rust terkenal efisien dan cepat, membuatnya mampu menangani banyak koneksi sekaligus tanpa membuat sistem lambat.

Dengan bantuan pustaka seperti *std::net* untuk versi sederhana (sinkron) atau *tokio* untuk versi asinkron (non-blocking), kita bisa membangun server yang bisa melayani ribuan client secara bersamaan. Sebagai contoh, kita bisa membuat TCP server sederhana yang membaca pesan dari client dan membalasnya, atau membangun versi asinkron yang lebih efisien dengan *tokio::spawn*. Semua kelebihan ini membuat TCP server di Rust sangat ideal digunakan dalam aplikasi nyata seperti sistem monitoring, kendali jarak jauh, atau layanan berbasis jaringan lainnya.

1.3 Blockchain

Blockchain adalah buku besar digital yang terdistribusi (distributed ledger) dan bekerja secara peer-to-peer. Teknologi ini menyimpan catatan transaksi dalam blok yang saling terhubung dan diberi penanda waktu. Setiap blok terenkripsi dan terhubung dengan blok sebelumnya melalui kriptografi, sehingga menciptakan rantai data yang tidak dapat diubah (immutable) dan transparan tanpa otoritas pusat (Alam, 2023).

1.4 Web3

Web3 atau *Web 3.0* merupakan generasi baru dari layanan internet yang didesain berbasis teknologi blockchain, dengan prinsip utama desentralisasi, kepemilikan data oleh pengguna, serta penghapusan ketergantungan pada pihak ketiga terpercaya. Dalam Web3, pengguna memiliki kendali penuh terhadap identitas digital, aset, dan data mereka, tidak seperti pada Web2 yang dikelola oleh platform terpusat (Wang et al., 2022). Web3 memiliki sejumlah karakteristik penting:

1. Terbuka: Data disimpan di jaringan publik dan dapat diakses siapa saja.
2. Trustless: Interaksi antar pengguna tidak membutuhkan kepercayaan atau perantara.
3. Permissionless: Tidak diperlukan izin untuk mengakses atau menggunakan layanan.
4. Anonim: Identitas pengguna dapat disamarkan melalui pseudonim.
5. Ketersediaan tinggi: Layanan tetap berjalan meskipun terjadi gangguan pada beberapa node.
6. Interoperabilitas: Dapat digunakan di berbagai platform blockchain seperti Ethereum, Solana, atau Binance Smart Chain.

1.5 InfluxDB

InfluxDB merupakan salah satu implementasi dari database time-series yang dikembangkan secara khusus untuk memenuhi kebutuhan penyimpanan dan pengolahan data berdasarkan waktu. InfluxDB didesain dengan arsitektur yang mampu menangani data dalam jumlah besar secara efisien dan memberikan performa tinggi, baik dalam hal

pencatatan data (write performance) maupun pengambilan data (query performance), terutama pada skala waktu yang sangat detail (misalnya per detik atau bahkan milidetik).

Berbeda dengan sistem basis data relasional seperti MySQL atau PostgreSQL yang bersifat umum, InfluxDB dioptimalkan untuk skenario penggunaan yang bersifat time-series. Hal ini menjadikannya unggul dalam beberapa aspek penting, seperti efisiensi penyimpanan data yang tinggi, kemampuan melakukan query berbasis waktu, agregasi data berdasarkan rentang waktu tertentu, pengaturan masa simpan data (data retention), serta pengolahan dan visualisasi data secara real-time. Keunggulan ini menjadikan InfluxDB sebagai salah satu pilihan utama dalam membangun sistem monitoring dan analisis data waktu secara modern.

1.6 Grafana

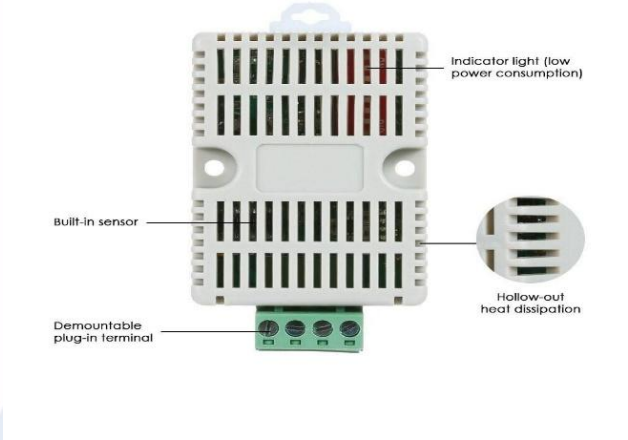
Grafana merupakan sebuah platfrom open-source yang digunakan untuk visualisasi data, pemantauan sistem, dan analisis data secara real-time. Platform ini memungkinkan pengguna untuk membuat dashboard interaktif yang menampilkan berbagai jenis visualisasi seperti grafik, tabel, dan gauge, sehingga memudahkan pemantauan performa sistem atau aplikasi secara langsung.

Grafana mendukung koneksi ke berbagai sumber data (data source) seperti InfluxDB, Prometheus, MySQL, PostgreSQL, dan Elasticsearch, sehingga fleksibel digunakan pada berbagai jenis aplikasi dan bidang. Sistem kerja Grafana dimulai dengan menghubungkan platform ini ke data source, kemudian mengambil data melalui query yang ditentukan pengguna. Data yang diperoleh selanjutnya divisualisasikan dalam bentuk panel-panel pada dashboard, yang dapat dikustomisasi sesuai kebutuhan.

Selain visualisasi, Grafana juga menyediakan fitur notifikasi dan alerting untuk memberikan peringatan otomatis jika terjadi kondisi abnormal sesuai aturan yang telah diatur pengguna. Kelebihan Grafana terletak pada sifatnya yang open-source, antarmuka yang user-friendly berbasis web, serta dukungan luas terhadap plugin dan data source,

sehingga menjadikannya pilihan populer dalam pemantauan infrastruktur TI, Internet of Things (IoT), sistem manufaktur, dan berbagai aplikasi lainnya. Penggunaan Grafana telah terbukti efektif dalam memonitor performa server, menganalisis data sensor IoT, mengawasi sistem industri, serta melakukan analisis data time-series secara efisien dan real-time.

1.7 SHT20 Temperature & Humidity Sensor



Sensor modbus SHT 20 merupakan sensor temperature dan kelembapan dengan memiliki presisi yang tinggi. Sensor ini menggunakan protocol komunikasi Modbus RTU berbasis RS485. SHT 20 memiliki karakteristik resistif terhadap perubahan kadar air di udara serta terdapat chip yang bisa mengkonversi analog ke digital dengan menggunakan bidirectional (kabel tunggal dua arah).

1.8 RS485 Module

RS485 (TIA/EIA-485) adalah standar komunikasi serial yang dirancang untuk transmisi data jarak jauh dan lingkungan industri yang bising. Berbeda dengan UART biasa yang bersifat single-ended (mengacu ke ground), RS485 menggunakan sinyal diferensial pada dua jalur utama (umumnya diberi label A dan B). Data direpresentasikan oleh selisih tegangan antara A dan B, sehingga lebih tahan terhadap interferensi elektromagnetik (noise) dan drop tegangan pada kabel panjang.

RS485 umumnya digunakan pada topologi bus multipoint, artinya satu jalur komunikasi dapat dipakai oleh banyak perangkat sekaligus (multi-drop). Dalam praktik instrumentasi, RS485 sering dipasangkan dengan protokol seperti Modbus RTU, di mana

satu perangkat bertindak sebagai master dan perangkat lain sebagai slave dengan alamat (ID) masing-masing. Karena satu bus dipakai bersama, komunikasi biasanya bersifat half-duplex (pengiriman dan penerimaan bergantian), sehingga modul RS485 membutuhkan kontrol arah data melalui pin seperti DE (Driver Enable) dan /RE (Receiver Enable). Saat perangkat mengirim data, DE diaktifkan; saat menerima, DE dimatikan dan receiver diaktifkan.

Untuk menjaga kualitas sinyal, jaringan RS485 idealnya memakai kabel twisted pair dan pada ujung-ujung bus dipasang termination resistor 120Ω untuk mengurangi pantulan sinyal. Beberapa modul RS485 juga menyediakan fitur fail-safe biasing agar kondisi idle bus stabil. Dalam implementasi, hal yang paling sering menyebabkan kegagalan komunikasi RS485 adalah pembalikan A/B, ground tidak common (pada sistem tertentu), pengaturan baud/parity yang tidak sama, serta kontrol DE/RE yang salah timing.

1.9 ESP32-S3 Microcontroller

ESP32-S3 adalah mikrokontroler dari Espressif yang ditujukan untuk aplikasi IoT modern karena menggabungkan kemampuan komputasi, konektivitas, dan fitur periferal dalam satu chip. ESP32-S3 memiliki prosesor Xtensa LX7 dual-core dan mendukung konektivitas Wi-Fi 2.4 GHz serta Bluetooth LE, sehingga cocok untuk sistem monitoring instrumentasi berbasis jaringan. Dengan dukungan RAM internal dan opsi flash eksternal pada modul, ESP32-S3 mampu menjalankan aplikasi yang membutuhkan komunikasi, buffering data, dan pemrosesan data sensor secara real-time.

Dalam konteks interkoneksi instrumentasi, ESP32-S3 penting karena menyediakan banyak antarmuka periferal, terutama UART yang digunakan untuk komunikasi serial seperti Modbus RTU melalui transceiver RS485. Selain UART, ESP32-S3 juga mendukung I2C, SPI, ADC, PWM, dan GPIO yang memudahkan integrasi berbagai sensor dan aktuator. ESP32-S3 bekerja pada level logika 3.3V, sehingga saat dihubungkan ke modul RS485 perlu memastikan transceiver kompatibel 3.3V atau menggunakan level shifting bila diperlukan.

Keunggulan ESP32-S3 pada sistem instrumentasi adalah kemampuannya menjadi gateway: membaca data sensor dari lapangan (mis. lewat RS485/Modbus), melakukan pemrosesan dasar (filtering, scaling, validasi), lalu mengirim data ke server menggunakan protokol jaringan seperti TCP/HTTP/MQTT melalui Wi-Fi. Dalam implementasi praktikum, perhatian utama pada ESP32-S3 biasanya mencakup pemilihan pin UART

yang benar, kestabilan catu daya, manajemen timing komunikasi (khususnya saat half-duplex RS485), serta pengemasan data (mis. JSON) agar dapat diproses sistem lain secara konsisten.

II. KEBUTUHAN PRAKTIKUM

Praktikum Interkoneksi Sistem Instrumentasi dilakukan secara offline, sehingga peralatan yang diperlukan adalah:

a. PC/ Laptop untuk melakukan praktikum
b. Module RS485
c. Microcontroller ESP32-S3
d. Sensor SHT20
e. Modul Praktikum Interkoneksi Sistem Instrumentasi

III. PROSEDUR PRAKTIKUM

A. PERCOBAAN

1. Siapkan perangkat: Sensor SHT20 RS485, USB-to-RS485 converter, dan PC/Laptop.
2. Hubungkan kabel RS485: A(+) ke A, B(−) ke B, dan pastikan sensor mendapat supply sesuai modul.
3. Colok USB-to-RS485 ke PC.
4. Buka terminal dan cek port serial terdeteksi:

```
ls /dev/ttyUSB*
```

5. Jika tidak bisa akses port (permission), jalankan:

```
sudo usermod -aG dialout $USER
# logout/login setelah ini
```

6. Buat project Rust baru:

```
cargo new modbus_client_tcp
cd modbus_client_tcp
```

7. Edit Cargo.toml, isi dependency berikut:

```
[package]
name = "modbus_client_tcp"
version = "0.1.0"
edition = "2021"

[dependencies]
anyhow = "1"
tokio = { version = "1", features = ["full"] }
tokio-modbus = "0.13"
tokio-serial = "5"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
chrono = "0.4"
```

8. Buka src/main.rs, lalu tempel kode ini (baca 2 input register mulai alamat 1, konversi /10, kirim JSON ke TCP):

```
use tokio_modbus::{client::rtu, prelude::*};
use tokio_serial::{SerialPortBuilderExt, Parity, StopBits, DataBits};
use tokio::net::TcpStream;
use tokio::io::AsyncWriteExt;
use serde::Serialize;
use chrono::Utc;
use anyhow::{Result, bail};
use tokio::time::{sleep, Duration};

#[derive(Serialize)]
struct SensorData {
```

```
    timestamp: String,
    sensor_id: String,
    location: String,
    process_stage: String,
    temperature_celsius: f32,
    humidity_percent: f32,
}

async fn read_sht20(port_path: &str, slave_id: u8) -> Result<(f32, f32)> {
    let builder = tokio_serial::new(port_path, 9600)
        .parity(Parity::None)
        .stop_bits(StopBits::One)
        .data_bits(DataBits::Eight)
        .timeout(std::time::Duration::from_secs(1));

    let port = builder.open_native_async()?;
    let mut ctx = rtu::connect_slave(port, Slave(slave_id)).await?;

    // Input registers addr=1 qty=2 (sesuai spec kamu)
    let resp = ctx.read_input_registers(1, 2).await?;
    if resp.len() != 2 {
        bail!("Response length != 2, got {:?}", resp);
    }

    let temp = resp[0] as f32 / 10.0;
    let rh   = resp[1] as f32 / 10.0;
    Ok((temp, rh))
}

async fn send_json_line(tcp_addr: &str, json_line: &str) -> Result<()> {
    let mut stream = TcpStream::connect(tcp_addr).await?;
    stream.write_all(json_line.as_bytes()).await?;
    stream.write_all(b"\n").await?;
    Ok(())
}

#[tokio::main]
async fn main() -> Result<()> {
    let serial_port = "/dev/ttyUSB0";   // hasil langkah 4
    let slave_id = 1u8;
    let tcp_addr = "127.0.0.1:9000";    // server di percobaan 3

    loop {
        match read_sht20(serial_port, slave_id).await {
            Ok((temp, rh)) => {
                println!("✅ Temp: {:.1} °C | RH: {:.1} %", temp, rh);
```

```
            let payload = SensorData {
                timestamp: Utc::now().to_rfc3339(),
                sensor_id: "SHT20-PascaPanen-001".into(),
                location: "Kumbung Inkubasi 1".into(),
                process_stage: "Inkubasi".into(),
                temperature_celsius: temp,
                humidity_percent: rh,
            };

            let json = serde_json::to_string(&payload)?;

            match send_json_line(tcp_addr, &json).await {
                Ok(_) => println!("📤 Data dikirim ke TCP server"),
                Err(e) => println!("❌ Gagal kirim ke TCP server: {e}"),
            }
        }
        Err(e) => println!("❌ Gagal baca sensor: {e}"),
    }

    sleep(Duration::from_secs(2)).await;
  }
}
```

9. Uji TCP cepat tanpa server percobaan 3 (opsional tapi bagus untuk pembuktian): buka terminal baru:

```
nc -lk 9000
```

10. Jalankan client:

```
cargo run
```

IV. PERTANYAAN PRAKTIKUM

1. Apa arti read_input_registers(1, 2) (alamat mulai dan jumlah register) dan kenapa hasilnya harus **2 nilai**?
2. Kenapa suhu dan kelembapan dibagi **10.0**? Jelaskan efeknya kalau tidak dibagi.
3. Sebutkan 3 penyebab paling umum **timeout/gagal baca** Modbus RTU dari /dev/ttyUSB0.
4. Kenapa data TCP dikirim dengan pemisah newline \n saat server membaca dengan lines()?

V. ANALISA & KESIMPULAN

Dari hasil percobaan terstruktur dan pertanyaan praktikum yang telah dikerjakan, buatlah analisa dan kesimpulan dari percobaan tersebut.

Note:

Laporan praktikum terdiri dari:

1. Dasar teori
2. Prosedur praktikum
3. Hasil percobaan praktikum
4. Hasil pertanyaan praktikum
5. Analisa
6. Kesimpulan

Hasil praktikum dibuktikan oleh laporan praktikum dan percobaan praktikum yang **wajib** dikerjakan secara mandiri oleh mahasiswa, segala bentuk kecurangan dan *plagiarisme* akan dikenai pengurangan nilai. Laporan praktikum dikumpulkan pada folder yang telah disediakan oleh asisten Laboratorium Elektronika.

Praktikum 2

Smart Contract

I. DASAR TEORI

1.1 ModBus Client

Modbus merupakan protokol komunikasi yang banyak digunakan dalam sistem otomasi industri untuk pertukaran data antara perangkat seperti PLC, sensor, dan aktuator. Dalam versi Modbus TCP, istilah *Modbus Client* mengacu pada perangkat atau program yang berperan aktif dalam memulai komunikasi, seperti membaca data dari atau menulis data ke *Modbus Server*.

Dalam hal ini, bahasa pemrograman Rust menjadi pilihan yang menarik untuk mengembangkan Modbus Client karena memiliki keunggulan dalam hal keamanan memori, kecepatan eksekusi, dan efisiensi dalam pengelolaan proses secara bersamaan (asinkron).

Pengembang dapat membangun aplikasi Modbus Client yang andal dan efisien untuk kebutuhan industri, seperti sistem pemantauan jarak jauh, pengiriman data dari perangkat ke cloud, atau integrasi antar perangkat di lingkungan industri. Penggunaan Rust dalam pengembangan Modbus Client memberikan jaminan kestabilan dan kinerja tinggi, yang sangat penting dalam sistem industri yang menuntut kecepatan dan keandalan komunikasi data secara real-time.

1.2 TCP Server

TCP Server dalam pemrograman Rust adalah program yang bertugas menerima dan merespons koneksi dari client menggunakan protokol TCP, yang menjamin data dikirim secara utuh dan berurutan. Rust sangat cocok untuk membuat TCP server karena punya sistem manajemen memori yang aman tanpa garbage collector, sehingga server lebih stabil dan bebas dari bug seperti crash atau data race. Selain itu, Rust terkenal efisien dan cepat, membuatnya mampu menangani banyak koneksi sekaligus tanpa membuat sistem lambat.

Dengan bantuan pustaka seperti *std::net* untuk versi sederhana (sinkron) atau *tokio* untuk versi asinkron (non-blocking), kita bisa membangun server yang bisa melayani ribuan client secara bersamaan. Sebagai contoh, kita bisa membuat TCP server sederhana yang membaca pesan dari client dan membalasnya, atau membangun versi asinkron yang lebih efisien dengan *tokio::spawn*. Semua kelebihan ini membuat TCP server di Rust sangat ideal digunakan dalam aplikasi nyata seperti sistem monitoring, kendali jarak jauh, atau layanan berbasis jaringan lainnya.

1.3 Blockchain

Blockchain adalah buku besar digital yang terdistribusi (distributed ledger) dan bekerja secara peer-to-peer. Teknologi ini menyimpan catatan transaksi dalam blok yang saling terhubung dan diberi penanda waktu. Setiap blok terenkripsi dan terhubung dengan blok sebelumnya melalui kriptografi, sehingga menciptakan rantai data yang tidak dapat diubah (immutable) dan transparan tanpa otoritas pusat (Alam, 2023).

1.4 Web3

Web3 atau *Web 3.0* merupakan generasi baru dari layanan internet yang didesain berbasis teknologi blockchain, dengan prinsip utama desentralisasi, kepemilikan data oleh pengguna, serta penghapusan ketergantungan pada pihak ketiga terpercaya. Dalam Web3, pengguna memiliki kendali penuh terhadap identitas digital, aset, dan data mereka, tidak seperti pada Web2 yang dikelola oleh platform terpusat (Wang et al., 2022). Web3 memiliki sejumlah karakteristik penting:

7. Terbuka: Data disimpan di jaringan publik dan dapat diakses siapa saja.
8. Trustless: Interaksi antar pengguna tidak membutuhkan kepercayaan atau perantara.
9. Permissionless: Tidak diperlukan izin untuk mengakses atau menggunakan layanan.
10. Anonim: Identitas pengguna dapat disamarkan melalui pseudonim.
11. Ketersediaan tinggi: Layanan tetap berjalan meskipun terjadi gangguan pada beberapa node.
12. Interoperabilitas: Dapat digunakan di berbagai platform blockchain seperti Ethereum, Solana, atau Binance Smart Chain.

1.5 InfluxDB

InfluxDB merupakan salah satu implementasi dari database time-series yang dikembangkan secara khusus untuk memenuhi kebutuhan penyimpanan dan pengolahan data berdasarkan waktu. InfluxDB didesain dengan arsitektur yang mampu menangani data dalam jumlah besar secara efisien dan memberikan performa tinggi, baik dalam hal

pencatatan data (write performance) maupun pengambilan data (query performance), terutama pada skala waktu yang sangat detail (misalnya per detik atau bahkan milidetik).

Berbeda dengan sistem basis data relasional seperti MySQL atau PostgreSQL yang bersifat umum, InfluxDB dioptimalkan untuk skenario penggunaan yang bersifat time-series. Hal ini menjadikannya unggul dalam beberapa aspek penting, seperti efisiensi penyimpanan data yang tinggi, kemampuan melakukan query berbasis waktu, agregasi data berdasarkan rentang waktu tertentu, pengaturan masa simpan data (data retention), serta pengolahan dan visualisasi data secara real-time. Keunggulan ini menjadikan InfluxDB sebagai salah satu pilihan utama dalam membangun sistem monitoring dan analisis data waktu secara modern.

1.6 Grafana

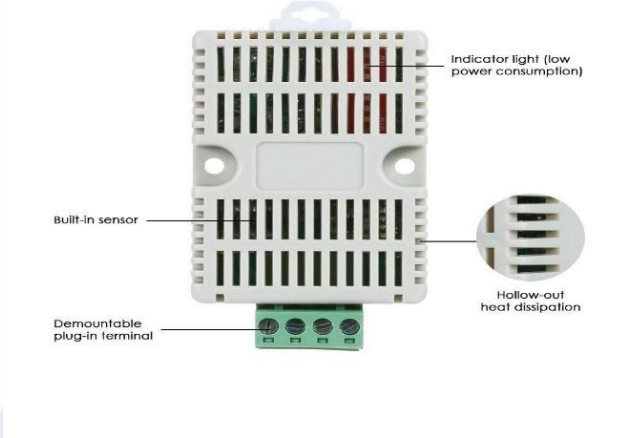
Grafana merupakan sebuah platfrom open-source yang digunakan untuk visualisasi data, pemantauan sistem, dan analisis data secara real-time. Platform ini memungkinkan pengguna untuk membuat dashboard interaktif yang menampilkan berbagai jenis visualisasi seperti grafik, tabel, dan gauge, sehingga memudahkan pemantauan performa sistem atau aplikasi secara langsung.

Grafana mendukung koneksi ke berbagai sumber data (data source) seperti InfluxDB, Prometheus, MySQL, PostgreSQL, dan Elasticsearch, sehingga fleksibel digunakan pada berbagai jenis aplikasi dan bidang. Sistem kerja Grafana dimulai dengan menghubungkan platform ini ke data source, kemudian mengambil data melalui query yang ditentukan pengguna. Data yang diperoleh selanjutnya divisualisasikan dalam bentuk panel-panel pada dashboard, yang dapat dikustomisasi sesuai kebutuhan.

Selain visualisasi, Grafana juga menyediakan fitur notifikasi dan alerting untuk memberikan peringatan otomatis jika terjadi kondisi abnormal sesuai aturan yang telah diatur pengguna. Kelebihan Grafana terletak pada sifatnya yang open-source, antarmuka yang user-friendly berbasis web, serta dukungan luas terhadap plugin dan data source,

sehingga menjadikannya pilihan populer dalam pemantauan infrastruktur TI, Internet of Things (IoT), sistem manufaktur, dan berbagai aplikasi lainnya. Penggunaan Grafana telah terbukti efektif dalam memonitor performa server, menganalisis data sensor IoT, mengawasi sistem industri, serta melakukan analisis data time-series secara efisien dan real-time.

1.7 SHT20 Temperature & Humidity Sensor



Sensor modbus SHT 20 merupakan sensor temperature dan kelembapan dengan memiliki presisi yang tinggi. Sensor ini menggunakan protocol komunikasi Modbus RTU berbasis RS485. SHT 20 memiliki karakteristik resistif terhadap perubahan kadar air di udara serta terdapat chip yang bisa mengkonversi analog ke digital dengan menggunakan bidirectional (kabel tunggal dua arah).

1.8 RS485 Module

RS485 (TIA/EIA-485) adalah standar komunikasi serial yang dirancang untuk transmisi data jarak jauh dan lingkungan industri yang bising. Berbeda dengan UART biasa yang bersifat single-ended (mengacu ke ground), RS485 menggunakan sinyal diferensial pada dua jalur utama (umumnya diberi label A dan B). Data direpresentasikan oleh selisih tegangan antara A dan B, sehingga lebih tahan terhadap interferensi elektromagnetik (noise) dan drop tegangan pada kabel panjang.

RS485 umumnya digunakan pada topologi bus multipoint, artinya satu jalur komunikasi dapat dipakai oleh banyak perangkat sekaligus (multi-drop). Dalam praktik instrumentasi, RS485 sering dipasangkan dengan protokol seperti Modbus RTU, di mana

satu perangkat bertindak sebagai master dan perangkat lain sebagai slave dengan alamat (ID) masing-masing. Karena satu bus dipakai bersama, komunikasi biasanya bersifat half-duplex (pengiriman dan penerimaan bergantian), sehingga modul RS485 membutuhkan kontrol arah data melalui pin seperti DE (Driver Enable) dan /RE (Receiver Enable). Saat perangkat mengirim data, DE diaktifkan; saat menerima, DE dimatikan dan receiver diaktifkan.

Untuk menjaga kualitas sinyal, jaringan RS485 idealnya memakai kabel twisted pair dan pada ujung-ujung bus dipasang termination resistor 120Ω untuk mengurangi pantulan sinyal. Beberapa modul RS485 juga menyediakan fitur fail-safe biasing agar kondisi idle bus stabil. Dalam implementasi, hal yang paling sering menyebabkan kegagalan komunikasi RS485 adalah pembalikan A/B, ground tidak common (pada sistem tertentu), pengaturan baud/parity yang tidak sama, serta kontrol DE/RE yang salah timing.

1.9 ESP32-S3 Microcontroller

ESP32-S3 adalah mikrokontroler dari Espressif yang ditujukan untuk aplikasi IoT modern karena menggabungkan kemampuan komputasi, konektivitas, dan fitur periferal dalam satu chip. ESP32-S3 memiliki prosesor Xtensa LX7 dual-core dan mendukung konektivitas Wi-Fi 2.4 GHz serta Bluetooth LE, sehingga cocok untuk sistem monitoring instrumentasi berbasis jaringan. Dengan dukungan RAM internal dan opsi flash eksternal pada modul, ESP32-S3 mampu menjalankan aplikasi yang membutuhkan komunikasi, buffering data, dan pemrosesan data sensor secara real-time.

Dalam konteks interkoneksi instrumentasi, ESP32-S3 penting karena menyediakan banyak antarmuka periferal, terutama UART yang digunakan untuk komunikasi serial seperti Modbus RTU melalui transceiver RS485. Selain UART, ESP32-S3 juga mendukung I2C, SPI, ADC, PWM, dan GPIO yang memudahkan integrasi berbagai sensor dan aktuator. ESP32-S3 bekerja pada level logika 3.3V, sehingga saat dihubungkan ke modul RS485 perlu memastikan transceiver kompatibel 3.3V atau menggunakan level shifting bila diperlukan.

Keunggulan ESP32-S3 pada sistem instrumentasi adalah kemampuannya menjadi gateway: membaca data sensor dari lapangan (mis. lewat RS485/Modbus), melakukan pemrosesan dasar (filtering, scaling, validasi), lalu mengirim data ke server menggunakan protokol jaringan seperti TCP/HTTP/MQTT melalui Wi-Fi. Dalam implementasi praktikum, perhatian utama pada ESP32-S3 biasanya mencakup pemilihan pin UART

yang benar, kestabilan catu daya, manajemen timing komunikasi (khususnya saat half-duplex RS485), serta pengemasan data (mis. JSON) agar dapat diproses sistem lain secara konsisten.

II. KEBUTUHAN PRAKTIKUM

Praktikum Interkoneksi Sistem Instrumentasi dilakukan secara offline, sehingga peralatan yang diperlukan adalah:

a. PC/ Laptop untuk melakukan praktikum
b. Module RS485
c. Microcontroller ESP32-S3
d. Sensor SHT20
e. Modul Praktikum Interkoneksi Sistem Instrumentasi

III. PROSEDUR PRAKTIKUM

1. Buat folder project:

```
mkdir percobaan2-smartcontract
cd percobaan2-smartcontract
npm init -y
npm install --save-dev hardhat
npx hardhat
```

2. Pilih Create a JavaScript project.

3. Jalankan node lokal:

```
npx hardhat node
```

4. Buat file contracts/SensorStorage.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SensorStorage {
  event DataStored(
    uint256 timestamp,
    string sensorId,
    string location,
    string stage,
    int256 temperature,
    int256 humidity
  );

  function storeData(
    uint256 timestamp,
    string memory sensorId,
    string memory location,
    string memory stage,
    int256 temperature,
    int256 humidity
  ) public {
    emit DataStored(timestamp, sensorId, location, stage, temperature, humidity);
  }
}
```

5. Compile kontrak:

```
npx hardhat compile
```

6. Buat file scripts/deploy.js:

```
const hre = require("hardhat");

async function main() {
 const SensorStorage = await hre.ethers.getContractFactory("SensorStorage");
```

```
  const contract = await SensorStorage.deploy();
  await contract.waitForDeployment();
  console.log("✅ SensorStorage deployed to:", await contract.getAddress());
}

main().catch((e) => {
  console.error(e);
  process.exitCode = 1;
});
```

7. Deploy ke localhost (terminal baru):

```
npx hardhat run scripts/deploy.js --network localhost
```

8. Buat file scripts/test_store.js:

```
const hre = require("hardhat");

async function main() {
  const [signer] = await hre.ethers.getSigners();

  const contractAddress = "0xYOUR_CONTRACT_ADDRESS"; // ganti dari hasil deploy

  const abi = [
    "event DataStored(uint256 timestamp,string sensorId,string location,string stage,int256 temperature,int256 humidity)",
    "function storeData(uint256,string,string,string,int256,int256)"
  ];

  const c = new hre.ethers.Contract(contractAddress, abi, signer);

  const ts = Math.floor(Date.now() / 1000);
  const tx = await c.storeData(ts, "SHT20-001", "Kumbung 1", "Inkubasi", 2730, 7820); // *100
  await tx.wait();
  console.log("✅ storeData ok");

  const ev = await c.queryFilter(c.filters.DataStored(), 0, "latest");
  console.log("Total events:", ev.length);
  console.log("Last args:", ev[ev.length - 1].args);
}

main().catch(console.error);
```

9. Jalankan uji:

```
npx hardhat run scripts/test_store.js --network localhost
```

IV. PERTANYAAN PRAKTIKUM

1. Kenapa data dicatat lewat event bukan disimpan sebagai variabel state di contract?
2. Kenapa temperatur & humidity disimpan sebagai integer skala ×100?
3. Apa bedanya transaction dan call, dan `storeData()` termasuk yang mana?
4. Kenapa `contractAddress` harus benar, dan apa yang terjadi kalau deploy ulang?

V. ANALISA DAN KESIMPULAN

Dari hasil percobaan terstruktur dan pertanyaan praktikum yang telah dikerjakan, buatlah analisa dan kesimpulan dari percobaan tersebut.

Note:

Laporan praktikum terdiri dari:

1. Dasar teori
2. Prosedur praktikum
3. Hasil percobaan praktikum
4. Hasil pertanyaan praktikum
5. Analisa
6. Kesimpulan

Hasil praktikum dibuktikan oleh laporan praktikum dan percobaan praktikum yang **wajib** dikerjakan secara mandiri oleh mahasiswa, segala bentuk kecurangan dan *plagiarisme* akan dikenai pengurangan nilai. Laporan praktikum dikumpulkan pada folder yang telah disediakan oleh asisten Laboratorium Elektronika.

Praktikum 3

Integrasi Blockhain

I. DASAR TEORI

1.1 ModBus Client

Modbus merupakan protokol komunikasi yang banyak digunakan dalam sistem otomasi industri untuk pertukaran data antara perangkat seperti PLC, sensor, dan aktuator. Dalam versi Modbus TCP, istilah *Modbus Client* mengacu pada perangkat atau program yang berperan aktif dalam memulai komunikasi, seperti membaca data dari atau menulis data ke *Modbus Server*.

Dalam hal ini, bahasa pemrograman Rust menjadi pilihan yang menarik untuk mengembangkan Modbus Client karena memiliki keunggulan dalam hal keamanan memori, kecepatan eksekusi, dan efisiensi dalam pengelolaan proses secara bersamaan (asinkron).

Pengembang dapat membangun aplikasi Modbus Client yang andal dan efisien untuk kebutuhan industri, seperti sistem pemantauan jarak jauh, pengiriman data dari perangkat ke cloud, atau integrasi antar perangkat di lingkungan industri. Penggunaan Rust dalam pengembangan Modbus Client memberikan jaminan kestabilan dan kinerja tinggi, yang sangat penting dalam sistem industri yang menuntut kecepatan dan keandalan komunikasi data secara real-time.

1.2 TCP Server

TCP Server dalam pemrograman Rust adalah program yang bertugas menerima dan merespons koneksi dari client menggunakan protokol TCP, yang menjamin data dikirim secara utuh dan berurutan. Rust sangat cocok untuk membuat TCP server karena punya sistem manajemen memori yang aman tanpa garbage collector, sehingga server lebih stabil dan bebas dari bug seperti crash atau data race. Selain itu, Rust terkenal efisien dan cepat, membuatnya mampu menangani banyak koneksi sekaligus tanpa membuat sistem lambat.

Dengan bantuan pustaka seperti *std::net* untuk versi sederhana (sinkron) atau *tokio* untuk versi asinkron (non-blocking), kita bisa membangun server yang bisa melayani ribuan client secara bersamaan. Sebagai contoh, kita bisa membuat TCP server sederhana yang membaca pesan dari client dan membalasnya, atau membangun versi asinkron yang lebih efisien dengan *tokio::spawn*. Semua kelebihan ini membuat TCP server di Rust sangat ideal digunakan dalam aplikasi nyata seperti sistem monitoring, kendali jarak jauh, atau layanan berbasis jaringan lainnya.

1.3 Blockchain

Blockchain adalah buku besar digital yang terdistribusi (distributed ledger) dan bekerja secara peer-to-peer. Teknologi ini menyimpan catatan transaksi dalam blok yang saling terhubung dan diberi penanda waktu. Setiap blok terenkripsi dan terhubung dengan blok sebelumnya melalui kriptografi, sehingga menciptakan rantai data yang tidak dapat diubah (immutable) dan transparan tanpa otoritas pusat (Alam, 2023).

1.4 Web3

Web3 atau *Web 3.0* merupakan generasi baru dari layanan internet yang didesain berbasis teknologi blockchain, dengan prinsip utama desentralisasi, kepemilikan data oleh pengguna, serta penghapusan ketergantungan pada pihak ketiga terpercaya. Dalam Web3, pengguna memiliki kendali penuh terhadap identitas digital, aset, dan data mereka, tidak seperti pada Web2 yang dikelola oleh platform terpusat (Wang et al., 2022). Web3 memiliki sejumlah karakteristik penting:

13. Terbuka: Data disimpan di jaringan publik dan dapat diakses siapa saja.
14. Trustless: Interaksi antar pengguna tidak membutuhkan kepercayaan atau perantara.
15. Permissionless: Tidak diperlukan izin untuk mengakses atau menggunakan layanan.
16. Anonim: Identitas pengguna dapat disamarkan melalui pseudonim.
17. Ketersediaan tinggi: Layanan tetap berjalan meskipun terjadi gangguan pada beberapa node.
18. Interoperabilitas: Dapat digunakan di berbagai platform blockchain seperti Ethereum, Solana, atau Binance Smart Chain.

1.5 InfluxDB

InfluxDB merupakan salah satu implementasi dari database time-series yang dikembangkan secara khusus untuk memenuhi kebutuhan penyimpanan dan pengolahan data berdasarkan waktu. InfluxDB didesain dengan arsitektur yang mampu menangani data dalam jumlah besar secara efisien dan memberikan performa tinggi, baik dalam hal

pencatatan data (write performance) maupun pengambilan data (query performance), terutama pada skala waktu yang sangat detail (misalnya per detik atau bahkan milidetik).

Berbeda dengan sistem basis data relasional seperti MySQL atau PostgreSQL yang bersifat umum, InfluxDB dioptimalkan untuk skenario penggunaan yang bersifat time-series. Hal ini menjadikannya unggul dalam beberapa aspek penting, seperti efisiensi penyimpanan data yang tinggi, kemampuan melakukan query berbasis waktu, agregasi data berdasarkan rentang waktu tertentu, pengaturan masa simpan data (data retention), serta pengolahan dan visualisasi data secara real-time. Keunggulan ini menjadikan InfluxDB sebagai salah satu pilihan utama dalam membangun sistem monitoring dan analisis data waktu secara modern.

1.6 Grafana

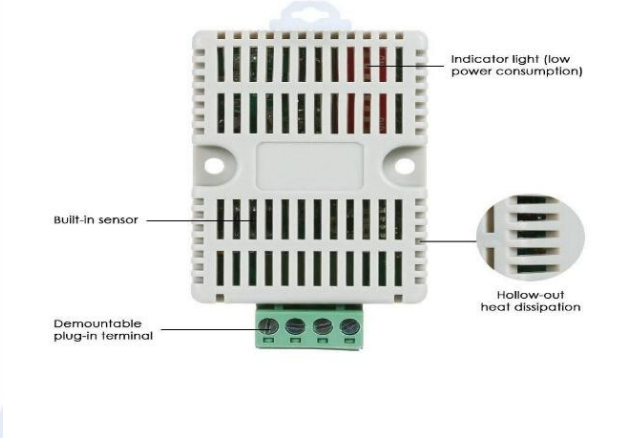
Grafana merupakan sebuah platfrom open-source yang digunakan untuk visualisasi data, pemantauan sistem, dan analisis data secara real-time. Platform ini memungkinkan pengguna untuk membuat dashboard interaktif yang menampilkan berbagai jenis visualisasi seperti grafik, tabel, dan gauge, sehingga memudahkan pemantauan performa sistem atau aplikasi secara langsung.

Grafana mendukung koneksi ke berbagai sumber data (data source) seperti InfluxDB, Prometheus, MySQL, PostgreSQL, dan Elasticsearch, sehingga fleksibel digunakan pada berbagai jenis aplikasi dan bidang. Sistem kerja Grafana dimulai dengan menghubungkan platform ini ke data source, kemudian mengambil data melalui query yang ditentukan pengguna. Data yang diperoleh selanjutnya divisualisasikan dalam bentuk panel-panel pada dashboard, yang dapat dikustomisasi sesuai kebutuhan.

Selain visualisasi, Grafana juga menyediakan fitur notifikasi dan alerting untuk memberikan peringatan otomatis jika terjadi kondisi abnormal sesuai aturan yang telah diatur pengguna. Kelebihan Grafana terletak pada sifatnya yang open-source, antarmuka yang user-friendly berbasis web, serta dukungan luas terhadap plugin dan data source,

sehingga menjadikannya pilihan populer dalam pemantauan infrastruktur TI, Internet of Things (IoT), sistem manufaktur, dan berbagai aplikasi lainnya. Penggunaan Grafana telah terbukti efektif dalam memonitor performa server, menganalisis data sensor IoT, mengawasi sistem industri, serta melakukan analisis data time-series secara efisien dan real-time.

1.7 SHT20 Temperature & Humidity Sensor



Sensor modbus SHT 20 merupakan sensor temperature dan kelembapan dengan memiliki presisi yang tinggi. Sensor ini menggunakan protocol komunikasi Modbus RTU berbasis RS485. SHT 20 memiliki karakteristik resistif terhadap perubahan kadar air di udara serta terdapat chip yang bisa mengkonversi analog ke digital dengan menggunakan bidirectional (kabel tunggal dua arah).

1.8 RS485 Module

RS485 (TIA/EIA-485) adalah standar komunikasi serial yang dirancang untuk transmisi data jarak jauh dan lingkungan industri yang bising. Berbeda dengan UART biasa yang bersifat single-ended (mengacu ke ground), RS485 menggunakan sinyal diferensial pada dua jalur utama (umumnya diberi label A dan B). Data direpresentasikan oleh selisih tegangan antara A dan B, sehingga lebih tahan terhadap interferensi elektromagnetik (noise) dan drop tegangan pada kabel panjang.

RS485 umumnya digunakan pada topologi bus multipoint, artinya satu jalur komunikasi dapat dipakai oleh banyak perangkat sekaligus (multi-drop). Dalam praktik instrumentasi, RS485 sering dipasangkan dengan protokol seperti Modbus RTU, di mana

satu perangkat bertindak sebagai master dan perangkat lain sebagai slave dengan alamat (ID) masing-masing. Karena satu bus dipakai bersama, komunikasi biasanya bersifat half-duplex (pengiriman dan penerimaan bergantian), sehingga modul RS485 membutuhkan kontrol arah data melalui pin seperti DE (Driver Enable) dan /RE (Receiver Enable). Saat perangkat mengirim data, DE diaktifkan; saat menerima, DE dimatikan dan receiver diaktifkan.

Untuk menjaga kualitas sinyal, jaringan RS485 idealnya memakai kabel twisted pair dan pada ujung-ujung bus dipasang termination resistor 120Ω untuk mengurangi pantulan sinyal. Beberapa modul RS485 juga menyediakan fitur fail-safe biasing agar kondisi idle bus stabil. Dalam implementasi, hal yang paling sering menyebabkan kegagalan komunikasi RS485 adalah pembalikan A/B, ground tidak common (pada sistem tertentu), pengaturan baud/parity yang tidak sama, serta kontrol DE/RE yang salah timing.

1.9 ESP32-S3 Microcontroller

ESP32-S3 adalah mikrokontroler dari Espressif yang ditujukan untuk aplikasi IoT modern karena menggabungkan kemampuan komputasi, konektivitas, dan fitur periferal dalam satu chip. ESP32-S3 memiliki prosesor Xtensa LX7 dual-core dan mendukung konektivitas Wi-Fi 2.4 GHz serta Bluetooth LE, sehingga cocok untuk sistem monitoring instrumentasi berbasis jaringan. Dengan dukungan RAM internal dan opsi flash eksternal pada modul, ESP32-S3 mampu menjalankan aplikasi yang membutuhkan komunikasi, buffering data, dan pemrosesan data sensor secara real-time.

Dalam konteks interkoneksi instrumentasi, ESP32-S3 penting karena menyediakan banyak antarmuka periferal, terutama UART yang digunakan untuk komunikasi serial seperti Modbus RTU melalui transceiver RS485. Selain UART, ESP32-S3 juga mendukung I2C, SPI, ADC, PWM, dan GPIO yang memudahkan integrasi berbagai sensor dan aktuator. ESP32-S3 bekerja pada level logika 3.3V, sehingga saat dihubungkan ke modul RS485 perlu memastikan transceiver kompatibel 3.3V atau menggunakan level shifting bila diperlukan.

Keunggulan ESP32-S3 pada sistem instrumentasi adalah kemampuannya menjadi gateway: membaca data sensor dari lapangan (mis. lewat RS485/Modbus), melakukan pemrosesan dasar (filtering, scaling, validasi), lalu mengirim data ke server menggunakan protokol jaringan seperti TCP/HTTP/MQTT melalui Wi-Fi. Dalam implementasi praktikum, perhatian utama pada ESP32-S3 biasanya mencakup pemilihan pin UART

yang benar, kestabilan catu daya, manajemen timing komunikasi (khususnya saat half-duplex RS485), serta pengemasan data (mis. JSON) agar dapat diproses sistem lain secara konsisten.

II. KEBUTUHAN PRAKTIKUM

Praktikum Interkoneksi Sistem Instrumentasi dilakukan secara offline, sehingga peralatan yang diperlukan adalah:

a. PC/ Laptop untuk melakukan praktikum
b. Module RS485
c. Microcontroller ESP32-S3
d. Sensor SHT20
e. Modul Praktikum Interkoneksi Sistem Instrumentasi

III. PROSEDUR PRAKTIKUM

B. PERCOBAAN

1. Pastikan InfluxDB v2 berjalan.
2. Buat ORG, BUCKET, dan TOKEN (write access).
3. Dari project Hardhat (Praktikum ke-2), ambil ABI & bytecode dari artifacts.
4. Buat folder build/ dan buat file ABI + BIN
5. Buat project:

```
cargo new tcp_listener_influx_chain
cd tcp_listener_influx_chain
```

6. Edit Cargo.toml:

```
[package]
name = "tcp_listener_influx_chain"
version = "0.1.0"
edition = "2021"

[dependencies]
anyhow = "1"
tokio = { version = "1", features = ["full"] }
serde = { version = "1", features = ["derive"] }
serde_json = "1"
reqwest = { version = "0.12", features = ["rustls-tls"] }
chrono = "0.4"
ethers = "2"
```

7. Buat file src/main.rs

```
use tokio::net::TcpListener;
use tokio::io::{AsyncBufReadExt, BufReader};
use serde::Deserialize;
use reqwest::Client;

use ethers::prelude::*;
use ethers::abi::Abi;
use std::{fs, sync::Arc};
use chrono::{DateTime, Utc};

#[derive(Deserialize, Debug)]
struct SensorData {
  timestamp: String,
  sensor_id: String,
  location: String,
  process_stage: String,
  temperature_celsius: f32,
  humidity_percent: f32,
```

```
}

#[tokio::main]
async fn main() -> anyhow::Result<()> {
  // ===== InfluxDB =====
  let influx_org = "rival team";
  let influx_bucket = "sensor_data";
  let influx_token = "PASTE_INFLUX_TOKEN";
  let influx_url = format!(
    "http://localhost:8086/api/v2/write?org={}&bucket={}&precision=s",
    urlencoding::encode(influx_org),
    urlencoding::encode(influx_bucket)
  );
  let http = Client::new();

  // ===== Ethereum local =====
  // Hardhat node default: http://localhost:8545, chain_id: 31337
  let provider = Provider::<Http>::try_from("http://localhost:8545")?;
  let wallet: LocalWallet = "0xPASTE_PRIVATE_KEY"
    .parse::<LocalWallet>()?
    .with_chain_id(31337u64);
  let signer = Arc::new(SignerMiddleware::new(provider, wallet));

  // ===== Deploy contract from ABI + BIN =====
  let abi_str = fs::read_to_string("build/SensorStorage.abi")?;
  let bin_str = fs::read_to_string("build/SensorStorage.bin")?;
  let abi: Abi = serde_json::from_str(&abi_str)?;
  let bytecode = bin_str.trim().parse::<Bytes>()?;
  let factory = ContractFactory::new(abi, bytecode, signer.clone());

  let contract = factory.deploy(())?.send().await?;
  println!("✅ Contract deployed at: {:?}", contract.address());

  // ===== TCP server =====
  let listener = TcpListener::bind("0.0.0.0:9000").await?;
  println!("✅ TCP listening on 9000...");

  loop {
    let (socket, addr) = listener.accept().await?;
    println!("🔌 Client connected: {}", addr);

    let influx_url = influx_url.clone();
    let influx_token = influx_token.to_string();
    let http = http.clone();
    let contract = contract.clone();
```

```
    tokio::spawn(async move {
        let reader = BufReader::new(socket);
        let mut lines = reader.lines();

        while let Ok(Some(line)) = lines.next_line().await {
            let parsed = serde_json::from_str::<SensorData>(&line);
            if parsed.is_err() {
                println!("❌ Invalid JSON: {}", line);
                continue;
            }
            let data = parsed.unwrap();
            println!("📩 Received: {:?}", data);

            // RFC3339 -> unix seconds
            let ts = DateTime::parse_from_rfc3339(&data.timestamp)
                .map(|dt| dt.timestamp())
                .unwrap_or_else(|_| Utc::now().timestamp());

            // ===== Influx write (Line Protocol) =====
            let lp = format!(
                "monitoring,sensor_id={},location={},stage={}
temperature={},humidity={} {}",
                data.sensor_id.replace(" ", "\\ "),
                data.location.replace(" ", "\\ "),
                data.process_stage.replace(" ", "\\ "),
                data.temperature_celsius,
                data.humidity_percent,
                ts
            );

            let resp = http
                .post(&influx_url)
                .header("Authorization", format!("Token {}", influx_token))
                .header("Content-Type", "text/plain")
                .body(lp)
                .send()
                .await;

            match resp {
                Ok(r) if r.status().is_success() => println!("✅ InfluxDB: written"),
                Ok(r) => println!("❌ InfluxDB status: {}", r.status()),
                Err(e) => println!("❌ InfluxDB error: {}", e),
            }
```

```
            // ===== Emit event on-chain =====
            let temp_i = (data.temperature_celsius * 100.0) as i64;
            let hum_i  = (data.humidity_percent * 100.0) as i64;

            let call = contract.method::<_, H256>("storeData", (
              ts as u64,
              data.sensor_id.clone(),
              data.location.clone(),
              data.process_stage.clone(),
              temp_i,
              hum_i,
            ));

            match call {
              Ok(m) => match m.send().await {
                Ok(p) => println!("✅ Ethereum tx sent: {:?}", p),
                Err(e) => println!("❌ Ethereum tx error: {:?}", e),
              },
              Err(e) => println!("❌ Contract call build error: {:?}", e),
            }
          }
        });
    }
}
```

8. Jalankan listener:

```
cargo run
```

9. Jalankan Modbus client
10. Buat folder frontend/ dan file index.html:

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <title>Sensor Event Monitor</title>
 <script
src="https://cdn.jsdelivr.net/npm/ethers@6/dist/ethers.umd.min.js"></script
>
 <script
src="https://cdn.jsdelivr.net/npm/chart.js@4/dist/chart.umd.min.js"></script
>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <h1>Monitoring Sensor (Blockchain Events)</h1>
 <button onclick="loadSensorData()">Load Data</button>
```

```
 <canvas id="chart" height="90"></canvas>

 <table id="sensorTable">
  <thead>
   <tr>
    <th>Time</th><th>Sensor
ID</th><th>Location</th><th>Stage</th><th>Temp</th><th>RH</th>
   </tr>
  </thead>
  <tbody></tbody>
 </table>

 <script src="script.js"></script>
</body>
</html>
```

11. Buat style.css:

```
body { font-family: sans-serif; margin: 20px; background: #f9f9f9; }
table { width: 100%; border-collapse: collapse; margin-top: 14px; background:
#fff; }
th, td { border: 1px solid #ddd; padding: 8px; text-align: center; }
button { padding: 10px 14px; cursor: pointer; }
```

12. Buat script.js (ganti contractAddress sesuai alamat kontrak yang tercetak dari listener Rust):

```
const contractAddress = "0xYOUR_CONTRACT_ADDRESS";
const abi = [
 "event DataStored(uint256 timestamp,string sensorId,string location,string
stage,int256 temperature,int256 humidity)"
];

let chart;

async function loadSensorData() {
 const provider = new ethers.BrowserProvider(window.ethereum);
 await provider.send("eth_requestAccounts", []);
 const signer = await provider.getSigner();
 const contract = new ethers.Contract(contractAddress, abi, signer);

 const events = await contract.queryFilter(contract.filters.DataStored(), 0,
"latest");

 const tbody = document.querySelector("#sensorTable tbody");
 tbody.innerHTML = "";
```

```
  const labels = [];
  const temps = [];
  const hums = [];

  for (const e of events) {
   const d = e.args;
   const timeStr = new Date(Number(d.timestamp) * 1000).toLocaleString();
   const temp = Number(d.temperature) / 100;
   const hum = Number(d.humidity) / 100;

   tbody.innerHTML += `
    <tr>
     <td>${timeStr}</td>
     <td>${d.sensorId}</td>
     <td>${d.location}</td>
     <td>${d.stage}</td>
     <td>${temp.toFixed(2)}</td>
     <td>${hum.toFixed(2)}</td>
    </tr>
   `;

   labels.push(timeStr);
   temps.push(temp);
   hums.push(hum);
  }

  renderChart(labels, temps, hums);
}

function renderChart(labels, temps, hums) {
  const ctx = document.getElementById("chart").getContext("2d");
  if (chart) chart.destroy();

  chart = new Chart(ctx, {
   type: "line",
   data: {
    labels,
    datasets: [
     { label: "Temperature (°C)", data: temps },
     { label: "Humidity (%)", data: hums }
    ]
   },
   options: { responsive: true }
  });
}
```

13. Jalankan web dengan Live Server / npx serve, lalu buka browser dan klik Load Data.

IV. PERTANYAAN PRAKTIKUM

1. Jelaskan alur data end-to-end dari pengiriman JSON sampai tampil di web (urut 4–6 langkah).
2. Dalam InfluxDB Line Protocol, mana yang jadi tag dan mana yang jadi field untuk `sensor_id`, `location`, `temperature`, `humidity`?
3. Kalau InfluxDB sukses tapi transaksi blockchain gagal, apa dampaknya ke data dan gimana cara meminimalkan masalah itu?
4. Kenapa `queryFilter(0, "latest")` bisa jadi masalah kalau data makin banyak, dan apa solusi sederhananya?

V. ANALISA & KESIMPULAN

Dari hasil percobaan terstruktur dan pertanyaan praktikum yang telah dikerjakan, buatlah analisa dan kesimpulan dari percobaan tersebut.

Note:

Laporan praktikum terdiri dari:

7. Dasar teori
8. Prosedur praktikum
9. Hasil percobaan praktikum
10. Hasil pertanyaan praktikum
11. Analisa
12. Kesimpulan

Hasil praktikum dibuktikan oleh laporan praktikum dan percobaan praktikum yang **wajib** dikerjakan secara mandiri oleh mahasiswa, segala bentuk kecurangan dan *plagiarisme* akan dikenai pengurangan nilai. Laporan praktikum dikumpulkan pada folder yang telah disediakan oleh asisten Laboratorium Elektronika.