**LAPORAN MATA KULIAH**

**INTERKONEKSI SISTEM INSTRUMENTASI**

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

**“Pengembangan Sistem Monitoring Suhu dan Kelembaban Berbasis IoT dan Blockchain untuk Meningkatkan Keandalan Pemantauan *Control Room* pada Industri *Oil and Gas*”**

**A yellow flower in a blue gear

AI-generated content may be incorrect.**

**Disusun Oleh :**

Ahmad Malikul Karim Amrullah 2042231041

Adam Fareliansah Malandi 2042231049

Kanaya Revania 2042231071

**PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI**

**DEPARTEMEN TEKNIK INSTRUMENTASI**

**FAKULTAS VOKASI**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**2025**

# Latar Belakang

Industri minyak dan gas mengandalkan ruang kontrol (control room) yang aman dan andal untuk pengawasan kondisi suhu dan kelembaban(Raina et al., 2024). Fluktuasi lingkungan ini dapat memicu kegagalan peralatan dan berpotensi membahayakan keselamatan operasional. Sensor Modbus SHT40, yang mendukung akurasi tinggi pada pengukuran suhu dan kelembaban, banyak digunakan sebagai perangkat penginderaan awal. Namun, penggunaan sensor tunggal tanpa integrasi sistem yang baik masih rentan terhadap keterlambatan data dan integritas yang diragukan(He & Lv, 2021).

Untuk memperbaiki hal ini, dikembangkan sistem berbasis IoT dengan arsitektur Modbus TCP Server sebagai pengumpul utama(He & Lv, 2021). Data sensor dikirim secara real-time melalui TCP, disimpan di InfluxDB dan divisualisasikan menggunakan Grafana. Konsep ini didukung oleh praktik industri yang menunjukkan bahwa kombinasi Modbus–InfluxDB–Grafana mampu menyediakan pemantauan efisien dan responsif untuk data suhu dan kelembaban. Telegraf sebagai jembatan antara Modbus dan InfluxDB juga dikenal memudahkan pengumpulan data industri. Antarmuka interaktif (GUI) disediakan sebagai front-end yang memudahkan pengguna melakukan monitoring dan interaksi sistem lewat ikon, tombol, menu, serta jendela. GUI ini penting untuk meningkatkan user experience dalam operasional control room yang dinamis.

Untuk menjamin keamanan dan keandalan data, teknologi blockchain digunakan bersama Metamask sebagai digital wallet. Data suhu dan kelembaban dikemas dalam smart contract, dikirim ke blockchain, serta diverifikasi menggunakan *Metamask* untuk otentikasi pengguna dan penyimpanan jejak transaksi. Pendekatan ini mengikuti tren industri IIoT yang mengadopsi blockchain guna melindungi integritas data sensor dan distribusi kepercayaan tanpa otoritas sentral.

Integrasi IoT–blockchain di lingkungan industri telah terbukti efektif dalam bidang lain seperti distribusi *supply chain* dan pelacakan aset kritikal, dimana teknologi ini membantu menjamin keamanan, keandalan data, serta transparansi log operasi. Model hierarkis dan efisien (seperti ChainSplitter) cocok untuk aplikasi berskala besar dan real-time seperti pada *control room* industri(Chbaik et al., 2024).

Dengan demikian, sistem monitoring suhu dan kelembaban yang mengintegrasikan sensor Modbus SHT40, Modbus TCP Server, InfluxDB, Grafana, GUI, Metamask, dan blockchain menawarkan solusi holistik dan inovatif. Sistem ini tidak hanya mendukung pengumpulan data akurat dan visualisasi responsif, tetapi juga menjamin keamanan dan integritas melalui penyimpanan desentralisasi, otentikasi digital, serta transparansi data yang dapat diaudit. Dengan skema tersebut, sistem ini mampu meningkatkan keandalan pemantauan control room di industri minyak dan gas, menjawab kebutuhan operasional serta regulasi yang ketat.

# Rumusan Masalah

1. Bagaimana merancang sistem monitoring suhu dan kelembaban yang dapat bekerja secara otomatis dan real-time untuk kebutuhan industri Oil & Gas?
2. Bagaimana sistem dapat memastikan integritas, keamanan, dan keterlacakan (traceability) data lingkungan melalui integrasi teknologi blockchain?
3. Bagaimana cara mengimplementasikan smart contract dan DApp untuk mendukung transparansi dan verifikasi kondisi lingkungan oleh pihak eksternal secara independen?
4. Sejauh mana efektivitas integrasi sensor SHT40, komunikasi Modbus RTU, dan penyimpanan data time-series dalam mendukung kebutuhan pemantauan lingkungan industri?

# Tujuan Proyek

1. Membangun sistem monitoring suhu dan kelembaban berbasis sensor digital SHT40 yang terhubung melalui konverter RS485 dengan protokol Modbus RTU.
2. Mengembangkan sistem pengiriman data lingkungan secara real-time ke server melalui jaringan TCP dan menyimpannya ke dalam basis data time-series (InfluxDB).
3. Mengintegrasikan teknologi blockchain (Etherium) untuk mencatat hash data lingkungan guna menjamin keutuhan dan keabsahan data (data integrity dan immutability).
4. Merancang dan membangun smart contract serta DApp yang memungkinkan pihak eksternal untuk memverifikasi kondisi lingkungan secara transparan dan terdesentralisasi.

# Metodologi dan Arsitektur Sistem

### Desain Arsitektur Sistem

Lampirkan diagram arsitektur sistem yang mencakup:

A diagram of a computer program

AI-generated content may be incorrect.

### Deskripsi Komponen

1. **Sensor Modbus SHT20**

Sensor Modbus SHT20 adalah device pengukur suhu dan kelembaban yang menggabungkan sensor SHT20 dengan protokol komunikasi Modbus. SHT40 merupakan generasi terbaru dari sensor digital suhu dan kelembaban yang dikenal karena akurasi tinggi, konsumsi daya rendah, dan ukuran yang ringkas. Ketika dikombinasikan dengan antarmuka Modbus, sensor ini dapat dengan mudah diintegrasikan ke dalam sistem otomasi dan monitoring berbasis jaringan. Penggunaan Modbus memungkinkan sensor ini untuk mengirimkan data secara efisien dan andal ke perangkat pengendali seperti PLC, HMI, atau komputer industri, sehingga cocok digunakan dalam berbagai aplikasi seperti pemantauan lingkungan, ruang penyimpanan, sistem HVAC, dan industri manufaktur.

Dalam sistem monitoring suhu dan kelembapan di sektor *Oil & Gas*, sensor ini ditempatkan di *control room* terhadap perubahan suhu dan kelembapan. Data dari sensor ditransmisikan secara real-time melalui jaringan RS-485 ke sistem pusat untuk dianalisis dan disimpan. Keunggulan RS-485 adalah kemampuannya mentransfer data jarak jauh dengan ketahanan tinggi terhadap gangguan elektromagnetik, menjadikannya ideal untuk lingkungan industri berat.

1. **TCP Server (Rust)**

Server TCP yang dibangun menggunakan bahasa pemrograman Rust ini berfungsi sebagai pengumpul data utama dari sensor. Server ini mendengarkan koneksi TCP yang masuk dari gateway, lalu menerima payload data dalam format JSON. Setelah parsing, data akan dikirimkan ke InfluxDB untuk disimpan sebagai data time-series. Penggunaan Rust memberikan keuntungan performa tinggi dan efisiensi memori, sangat cocok untuk aplikasi monitoring yang membutuhkan stabilitas dan keamanan tinggi.

1. **InfluxDB**

InfluxDB adalah database time-series yang dirancang khusus untuk menangani data berbasiskan waktu seperti suhu dan kelembapan. Setiap data yang masuk disimpan dengan timestamp, memungkinkan pemantauan tren historis dan pencatatan kondisi lingkungan secara terstruktur dan efisien. InfluxDB menjadi pusat penyimpanan data yang selanjutnya dapat divisualisasikan atau dianalisis.

1. **Grafana**

Grafana merupakan sebuah platform visualisasi data open-source yang digunakan untuk membuat dashboard interaktif dan real-time. Grafana mampu menampilkan data dari berbagai sumber, salah satunya adalah InfluxDB, dalam bentuk grafik, tabel, gauge, dan berbagai jenis visual lainnya yang mudah dipahami oleh para petani kopi. Keunggulan utama Grafana adalah kemampuannya menyajikan data secara visual, dinamis, dan dapat disesuaikan sesuai kebutuhan pengguna. Menampilkan dashboard suhu dan kelembaban.

Grafana membantu para petani untuk dapat memantau kondisi fermentasi kopi secara langsung dan real-time melalui grafik tanpa perlu membaca data mentah atau angka-angka kompleks. Sebagai contoh, jika suhu fermentasi naik di atas ambang batas optimal, maka Grafana dapat menampilkan peringatan visual yang mudah dipahami oleh orang awam. Selain itu, Grafana juga memungkinkan pengguna untuk melihat tren historis, membandingkan performa antar hari, dan melakukan analisis visual terhadap data lingkungan selama proses fermentasi. Dengan adanya visualisasi ini, proses pengambilan keputusan menjadi lebih cepat, tepat, dan berbasis data, sehingga kualitas hasil akhir fermentasi kopi dapat lebih terjamin.

1. **GUI (Graphical User Interface)**

Interface berbasis grafis yang memungkinkan pengguna berinteraksi dengan sistem komputer atau perangkat lunak melalui elemen visual seperti ikon, tombol, jendela, dan menu, tanpa harus menggunakan perintah berbasis teks. Dalam Sistem Monitoring Suhu dan Kelembaban Pasca Panen Kopi untuk Menjaga Kualitas Fermentasi, GUI berfungsi sebagai jembatan antara pengguna dan seluruh sistem monitoring yang mencakup sensor, database, hingga visualisasi data. GUI memungkinkan pengguna untuk melihat data suhu dan kelembaban secara real-time, mengakses histori data, memantau status koneksi sensor, dan menerima notifikasi atau peringatan jika terjadi kondisi yang tidak sesuai (misalnya suhu terlalu tinggi). GUI ini biasanya dibangun di atas platform visualisasi seperti Grafana atau aplikasi web khusus, yang terhubung langsung ke database (seperti InfluxDB) dan server data. Dengan adanya GUI, pengelola fermentasi kopi dapat melakukan pemantauan dan pengambilan keputusan dengan lebih cepat, mudah, dan efisien, tanpa perlu memahami aspek teknis di balik sistem tersebut.

1. **MetaMask**

MetaMask adalah dompet digital berbasis perangkat lunak yang digunakan untuk menyimpan dan mengelola aset kripto seperti Ethereum dan token lainnya. MetaMask tersedia sebagai ekstensi browser dan aplikasi mobile, serta memungkinkan pengguna untuk terhubung dengan aplikasi terdesentralisasi (dApps) seperti DeFi, NFT marketplace, dan layanan berbasis blockchain lainnya. Selain itu, MetaMask bersifat non-custodial, artinya pengguna memegang kendali penuh atas private key dan seed phrase mereka.

Dalam sistem monitoring berbasis blockchain, MetaMask dapat digunakan sebagai identitas digital atau alat autentikasi untuk mengakses data yang disimpan di jaringan seperti Ethereum. Dengan integrasi ini, data seperti suhu dan kelembaban dari sensor IoT dapat dikirim ke server blockchain dan divisualisasikan secara real-time, sambil menjaga keamanan dan transparansi data.

1. **Hardhat**

Hardhat merupakan sebuah *Ethereum development environment* yang digunakan untuk menyusun, menguji, dan menyebarkan (*deploy*) smart contract. Dengan fitur seperti jaringan lokal, debugging, serta integrasi plugin seperti Ethers.js, Hardhat memungkinkan pengembang untuk menjalankan simulasi kontrak pintar secara efisien sebelum diunggah ke jaringan blockchain sebenarnya. Hardhat juga menyediakan fleksibilitas dalam menulis skrip otomatisasi dan pengujian unit, yang sangat berguna dalam proses pengembangan sistem terdesentralisasi.

1. **Web3**

Web3.js, yaitu sebuah pustaka JavaScript yang memungkinkan aplikasi frontend atau backend untuk berinteraksi langsung dengan blockchain Ethereum. Melalui Web3.js, aplikasi dapat melakukan transaksi, membaca data dari smart contract, serta menghubungkan dompet digital pengguna seperti MetaMask. Web3.js berperan sebagai jembatan komunikasi antara smart contract yang telah dideploy melalui Hardhat dengan antarmuka pengguna.

1. **Etherscan**

Etherscan dimanfaatkan sebagai alat verifikasi dan pemantauan transaksi serta smart contract yang telah diterbitkan ke jaringan Ethereum. Etherscan merupakan blockchain explorer yang memungkinkan pengguna dan pengembang untuk melihat riwayat transaksi, status kontrak, serta log aktivitas yang terjadi dalam jaringan secara transparan. Setelah kontrak berhasil dideploy, informasi dan kode kontrak dapat diverifikasi secara publik melalui Etherscan agar pihak lain dapat membaca dan mengaudit isi smart contract dengan mudah.

# Implementasi dan Kode Program

* + - 1. **ModBus RTU**

Modbus RTU adalah protokol komunikasi serial yang digunakan untuk membaca data dari sensor dengan sistem master-slave. Master mengirim permintaan berisi alamat sensor dan alamat register, lalu sensor merespons dengan data. Komunikasi dilakukan lewat RS-485, dan data biasanya berupa angka dari register yang harus dikonversi sesuai kebutuhan. Untuk membaca data, digunakan function code seperti 0x03 (read holding register), dan detail alamat register bisa dilihat di datasheet sensor.

**Kode Sensor yang terhubung ke TCP server, dan mengirim data ke influx db setelah itu mengirim server etherium**

| use std::sync::Arc;  use std::time::Duration;  use tokio::sync::Mutex;  use tokio::net::TcpListener;  use tokio::io::AsyncWriteExt;  use anyhow::{Context as AnyhowContext, Result};  use serde::{Deserialize, Serialize};  use std::fs::{File, OpenOptions};  use std::io::{BufReader, Write};  use std::env;  use influxdb2::Client;  use influxdb2::models::DataPoint;  use chrono::{Utc, DateTime, NaiveDateTime};  use futures\_util::stream::iter;  use ethers::prelude::\*;  use ethers::abi::Abi;  use tokio\_serial::{SerialPortBuilderExt, DataBits, Parity, StopBits};  use tokio\_modbus::prelude::\*;  use tokio\_modbus::client::rtu;  use rand::{Rng, SeedableRng};  use rand::rngs::StdRng;  const TCP\_SERVER\_ADDRESS: &str = "127.0.0.1:8080";  const INFLUXDB\_URL: &str = "http://localhost:8086";  const INFLUXDB\_ORG: &str = "monitoring";  const INFLUXDB\_BUCKET: &str = "datasensor";  const INFLUXDB\_TOKEN: &str = "CBnS419-R1vL9EOWQkTXTnkOLLkTBoCBxF17g3EpKyvPfeb6qR1pDc1QziOxGvcr\_bCT3ZpMk2jSJH6qG9c3OQ==";  const DEVICE\_ID: &str = "device-01";  #[derive(Debug, Clone, Default, Serialize, Deserialize)]  struct SensorData {  temperature: f32,  humidity: f32,  timestamp: i64,  }  #[derive(Serialize)]  struct InfluxFormatJson {  timestamp: String,  sensor\_id: String,  location: String,  process\_stage: String,  temperature\_celsius: f32,  humidity\_percent: f32,  }  fn save\_to\_format\_json(data: &SensorData) -> Result<()> {  let naive = NaiveDateTime::from\_timestamp\_opt(data.timestamp / 1\_000\_000\_000, 0)  .unwrap\_or\_else(|| NaiveDateTime::from\_timestamp(0, 0));  let datetime: DateTime<Utc> = DateTime::<Utc>::from\_utc(naive, Utc);  let json = InfluxFormatJson {  timestamp: datetime.to\_rfc3339(),  sensor\_id: DEVICE\_ID.to\_string(),  location: "Gudang Fermentasi 1".to\_string(),  process\_stage: "Fermentasi".to\_string(),  temperature\_celsius: data.temperature,  humidity\_percent: data.humidity,  };  let mut file = OpenOptions::new()  .create(true)  .write(true)  .truncate(true)  .open("format.json")?;  write!(file, "{}", serde\_json::to\_string\_pretty(&json)?)?;  println!("[Writer] 📄 format.json disimpan.");  Ok(())  }  #[tokio::main]  async fn main() -> Result<()> {  dotenv::dotenv().ok();  println!("✅ File .env telah dimuat.");  let use\_dummy = false;  let shared\_data = Arc::new(Mutex::new(SensorData::default()));  let tcp\_task\_data = Arc::clone(&shared\_data);  let writer\_task\_data = Arc::clone(&shared\_data);  let sensor\_handle = if use\_dummy {  let dummy\_data = Arc::clone(&shared\_data);  Some(tokio::spawn(async move {  run\_dummy\_sensor(dummy\_data).await?;  Ok::<(), anyhow::Error>(())  }))  } else {  let real\_data = Arc::clone(&shared\_data);  Some(tokio::spawn(run\_serial\_reader(real\_data)))  };  let tcp\_server\_handle = tokio::spawn(run\_tcp\_server(tcp\_task\_data));  let writer\_handle = tokio::spawn(run\_database\_writer(writer\_task\_data));  println!("🚀 Semua service telah dimulai.");  if use\_dummy {  println!("🧪 Mode: Dummy Sensor Aktif");  } else {  println!("📡 Mode: Pembacaan Sensor Asli Aktif pada /dev/ttyUSB0");  }  println!("🔌 TCP Server mendengarkan di {}", TCP\_SERVER\_ADDRESS);  println!("💾 Database writer berjalan.");  let \_ = tokio::try\_join!(sensor\_handle.unwrap(), tcp\_server\_handle, writer\_handle)?;  Ok(())  }  async fn run\_dummy\_sensor(data: Arc<Mutex<SensorData>>) -> Result<()> {  let mut rng = StdRng::from\_entropy();  loop {  let dummy = SensorData {  temperature: rng.gen\_range(20.0..30.0),  humidity: rng.gen\_range(40.0..60.0),  timestamp: Utc::now().timestamp\_nanos\_opt().unwrap\_or(0),  };  {  let mut guard = data.lock().await;  \*guard = dummy.clone();  }  println!("[Dummy] ✅ Data dummy diperbarui: {:?}", dummy);  tokio::time::sleep(Duration::from\_secs(2)).await;  }  }  async fn run\_serial\_reader(data: Arc<Mutex<SensorData>>) -> Result<()> {  let port\_name = "/dev/ttyUSB0";  let baud\_rate = 9600;  let slave\_id = Slave(0x01);  let builder = tokio\_serial::new(port\_name, baud\_rate)  .data\_bits(DataBits::Eight)  .parity(Parity::None)  .stop\_bits(StopBits::One)  .timeout(Duration::from\_secs(1));  let port = builder.open\_native\_async().context("Gagal membuka port serial")?;  let mut ctx = rtu::attach\_slave(port, slave\_id);  println!("[Serial] ✅ Listening Modbus RTU pada {}", port\_name);  loop {  match ctx.read\_input\_registers(1, 2).await {  Ok(response) if response.len() == 2 => {  let temp = response[0] as f32 / 10.0;  let hum = response[1] as f32 / 10.0;  let parsed = SensorData {  temperature: temp,  humidity: hum,  timestamp: Utc::now().timestamp\_nanos\_opt().unwrap\_or(0),  };  let mut guard = data.lock().await;  \*guard = parsed;  println!("[Serial] ✅ Data diterima: {:?}", \*guard);  }  Ok(response) => {  println!("[Serial] ⚠ Response tidak lengkap: {:?}", response);  }  Err(e) => {  eprintln!("[Serial] ❌ Gagal baca dari sensor: {}", e);  }  }  tokio::time::sleep(Duration::from\_secs(2)).await;  }  }  async fn run\_tcp\_server(data: Arc<Mutex<SensorData>>) -> Result<()> {  let listener = TcpListener::bind(TCP\_SERVER\_ADDRESS).await?;  loop {  let (mut stream, addr) = listener.accept().await?;  println!("[TCP Server] 🔗 Koneksi dari: {}", addr);  let data\_clone = Arc::clone(&data);  tokio::spawn(async move {  let data\_guard = data\_clone.lock().await;  let response = serde\_json::to\_string(&\*data\_guard).unwrap\_or\_default();  if let Err(e) = stream.write\_all(response.as\_bytes()).await {  eprintln!("[TCP Server] ❌ Gagal kirim: {}", e);  } else {  println!("[TCP Server] ✅ Data dikirim ke {}", addr);  }  });  }  }  async fn run\_database\_writer(data: Arc<Mutex<SensorData>>) -> Result<()> {  let eth\_rpc\_url = env::var("ETH\_RPC\_URL").expect("ETH\_RPC\_URL harus diset di .env");  let contract\_address = env::var("CONTRACT\_ADDRESS").expect("CONTRACT\_ADDRESS harus diset di .env");  let gateway\_private\_key = env::var("GATEWAY\_PRIVATE\_KEY").expect("GATEWAY\_PRIVATE\_KEY harus diset di .env");  let influx\_client = Client::new(INFLUXDB\_URL, INFLUXDB\_ORG, INFLUXDB\_TOKEN);  let provider = Provider::<Http>::try\_from(eth\_rpc\_url)?;  let wallet: LocalWallet = gateway\_private\_key.parse::<LocalWallet>()?.with\_chain\_id(provider.get\_chainid().await?.as\_u64());  let signer = SignerMiddleware::new(provider, wallet.clone());  let abi\_file = File::open("src/abi/DataRegistry.json")?;  let reader = BufReader::new(abi\_file);  let abi: Abi = serde\_json::from\_reader(reader)?;  let contract = Contract::new(contract\_address.parse::<Address>()?, abi, Arc::new(signer));  let mut last\_written\_time = 0i64;  loop {  tokio::time::sleep(Duration::from\_secs(10)).await;  let current\_data: SensorData;  {  let data\_guard = data.lock().await;  if data\_guard.timestamp > last\_written\_time {  current\_data = data\_guard.clone();  last\_written\_time = current\_data.timestamp;  } else {  println!("[Writer] ⏸ Menunggu data sensor baru...");  continue;  }  }  // Simpan format.json  if let Err(e) = save\_to\_format\_json(&current\_data) {  eprintln!("[Writer] ❌ Gagal simpan format.json: {}", e);  }  let point = DataPoint::builder("sensor\_data")  .tag("device\_id", DEVICE\_ID)  .field("temperature", current\_data.temperature as f64)  .field("humidity", current\_data.humidity as f64)  .timestamp(current\_data.timestamp)  .build()?;  match influx\_client.write(INFLUXDB\_BUCKET, iter(vecpoint])).await {  Ok(\_) => {  println!(  "[Writer] ✅ Data ditulis ke InfluxDB: {:.2}°C, {:.2}%RH @ {}",  current\_data.temperature, current\_data.humidity, current\_data.timestamp  );  }  Err(e) => {  eprintln!("[Writer] ❌ Gagal tulis ke InfluxDB: {}", e);  }  }  println!("[Writer] 🚀 Mengirim ke Blockchain...");  let temp\_for\_chain = (current\_data.temperature \* 100.0) as i64;  let hum\_for\_chain = (current\_data.humidity \* 100.0) as i64;  let call = contract.method::<\_, ()>("recordData", (DEVICE\_ID.to\_string(), temp\_for\_chain, hum\_for\_chain))?;  match call.send().await {  Ok(pending\_tx) => {  println!("[Writer] ✅ Transaksi Blockchain terkirim! Hash: {:?}", pending\_tx.tx\_hash());  }  Err(e) => {  eprintln!("[Writer] ❌ Gagal kirim Blockchain: {}", e);  }  }  }  } |
| --- |

**Deploy Private Key**

| const hre = require("hardhat");  const fs = require("fs");  const path = require("path");  async function main() {  // 1. Deploy kontrak  const dataRegistry = await hre.ethers.deployContract("DataRegistry");  await dataRegistry.waitForDeployment();  const contractAddress = dataRegistry.target;  console.log(DataRegistry deployed to: ${contractAddress} on network ${hre.network.name});  // 2. Tentukan konfigurasi berdasarkan jaringan yang aktif  let config;  if (hre.network.name === "sepolia") {  config = {  ETH\_RPC\_URL: process.env.SEPOLIA\_RPC\_URL, // Ambil dari .env Hardhat  GATEWAY\_PRIVATE\_KEY: process.env.WALLET\_PRIVATE\_KEY, // Ambil dari .env Hardhat  CONTRACT\_ADDRESS: contractAddress,  INFLUXDB\_TOKEN: "YOUR\_API\_TOKEN",  };  } else { // Asumsikan localhost jika bukan sepolia  config = {  ETH\_RPC\_URL: "http://127.0.0.1:8545",  GATEWAY\_PRIVATE\_KEY: "0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80",  CONTRACT\_ADDRESS: contractAddress,  INFLUXDB\_TOKEN: "YOUR\_API\_TOKEN",  };  }    // 3. Tulis konfigurasi ke file .env Rust dan React  const rustEnvPath = '/home/adam/Documents/ISI\_AAM/sensor-gateaway/.env';  writeEnvFile(rustEnvPath, config);  console.log(\n✅ Konfigurasi untuk Backend Rust berhasil ditulis.);  const reactConfig = { REACT\_APP\_CONTRACT\_ADDRESS: contractAddress };  const reactEnvPath = '/home/adam/Documents/ISI\_AAM/frontend-dapp/.env';  writeEnvFile(reactEnvPath, reactConfig);  console.log(✅ Konfigurasi untuk Frontend React berhasil ditulis.);  }  /\*\*  \* Fungsi helper untuk membuat direktori (jika belum ada) dan menulis file .env.  \* @param {string} filePath - Path lengkap ke file .env yang akan ditulis.  \* @param {object} configObject - Objek yang berisi key-value untuk konfigurasi.  \*/  function writeEnvFile(filePath, configObject) {  const dirPath = path.dirname(filePath);  if (!fs.existsSync(dirPath)) {  fs.mkdirSync(dirPath, { recursive: true });  console.log(Direktori dibuat: ${dirPath});  }  const envFileContent = Object.entries(configObject)  .map(([key, value]) => ${key}=${value})  .join("\n");  fs.writeFileSync(filePath, envFileContent);  }  main().catch((error) => {  console.error(error);  process.exitCode = 1;  }); |
| --- |

**2. Kode Rust TCP Server**

Kode Rust TCP Server ini dirancang untuk menerima data dari client melalui koneksi TCP, memproses data dalam format JSON, dan kemudian menyimpannya ke database InfluxDB. Setelah server berhasil dibuka pada port 9000 menggunakan TcpListener, setiap koneksi masuk akan ditangani secara paralel melalui tokio::spawn. Di dalam task asynchronous ini, socket yang terhubung dibaca baris per baris menggunakan BufReader dan lines() untuk menangkap data yang dikirim oleh client.

Setiap baris data yang diterima diasumsikan berbentuk string JSON dan akan diparsing menjadi struktur SensorData menggunakan serde\_json::from\_str. Jika parsing berhasil, data akan ditampilkan ke konsol dengan println! sebagai log. Setelah data berhasil diparsing, informasi seperti sensor\_id, location, process\_stage, suhu, dan kelembaban akan diformat ke dalam Line Protocol, yaitu format standar yang digunakan oleh InfluxDB. Format ini mencakup measurement (monitoring), tag (seperti sensor\_id, location, dan stage), field (temperature, humidity), dan timestamp yang diambil dari field timestamp dalam data.

Data yang sudah diformat ini kemudian dikirim ke InfluxDB menggunakan HTTP POST dengan bantuan library reqwest, di mana endpoint, token otentikasi, dan header dikonfigurasi sebelumnya. Bila pengiriman berhasil, sistem akan mencetak konfirmasi bahwa data telah dikirim, dan jika gagal, akan muncul pesan error. Dengan alur ini, sistem mampu secara real-time menerima, memproses, dan menyimpan data sensor ke dalam database time-series untuk keperluan monitoring lebih lanjut.

**Kode File Contract**

| // SPDX-License-Identifier: MIT  pragma solidity ^0.8.24;  contract DataRegistry {  // Struct untuk menyimpan satu data pembacaan sensor  struct SensorReading {  int64 temperature; // Suhu dikali 100 (misal: 25.50°C disimpan sbg 2550)  int64 humidity; // Kelembapan dikali 100 (misal: 60.75% disimpan sbg 6075)  uint256 timestamp; // Waktu data dicatat  }  // Mapping untuk menyimpan riwayat data berdasarkan ID perangkat  mapping(string => SensorReading[]) public readingsByDevice;  // Event yang akan dipancarkan setiap kali data baru dicatat  event DataRecorded(string deviceId, int64 temperature, int64 humidity);  /\*\*  \* @dev Mencatat data sensor baru ke blockchain.  \* @param deviceId ID unik dari perangkat/sensor.  \* @param \_temperature Suhu yang akan dicatat (sudah dikali 100).  \* @param \_humidity Kelembapan yang akan dicatat (sudah dikali 100).  \*/  function recordData(string memory deviceId, int64 \_temperature, int64 \_humidity) public {  readingsByDevice[deviceId].push(  SensorReading({  temperature: \_temperature,  humidity: \_humidity,  timestamp: block.timestamp  })  );  emit DataRecorded(deviceId, \_temperature, \_humidity);  }  } |
| --- |

**3. Kode WEB3**

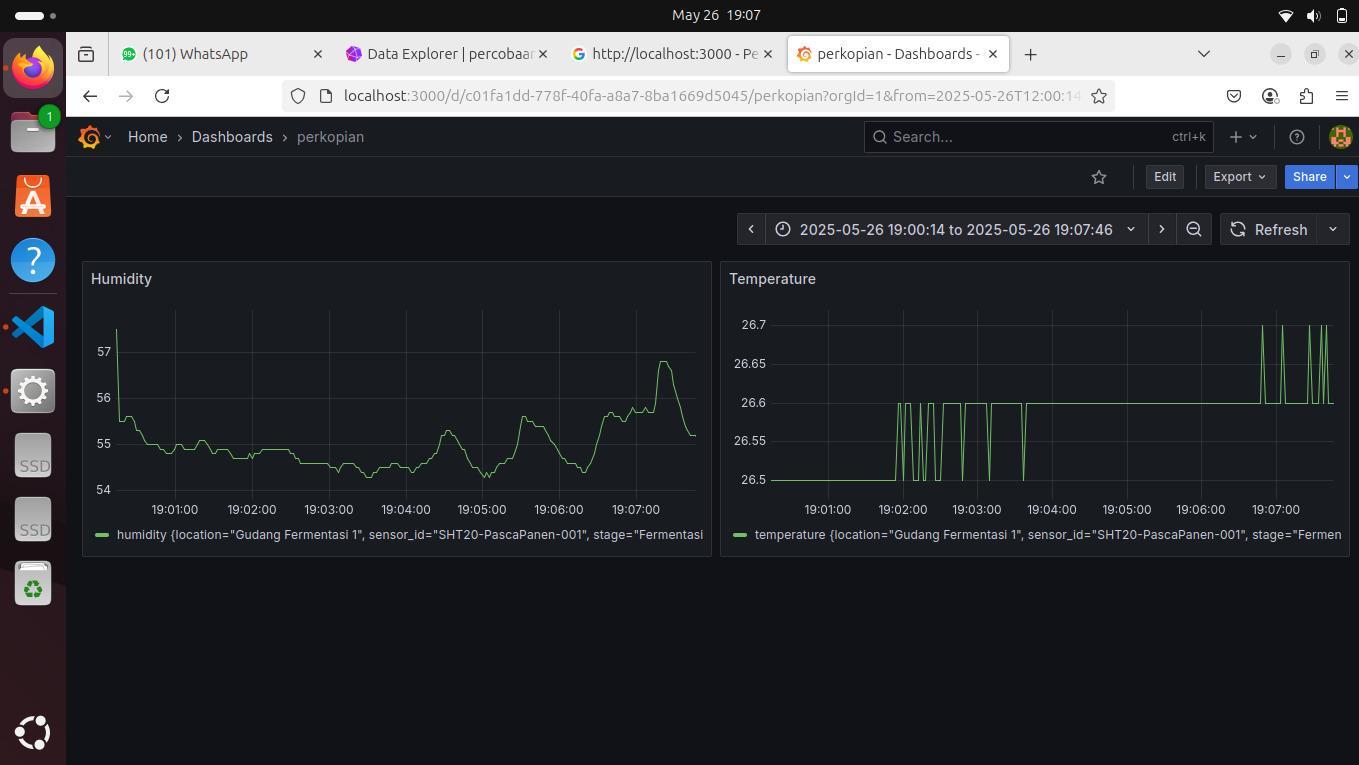
| import { useState, useEffect, useRef } from 'react';  import { ethers } from 'ethers';  import DataRegistryABI from './abi/DataRegistry.json';  import {  LineChart, Line, XAxis, YAxis, Tooltip, CartesianGrid, Legend, ResponsiveContainer  } from 'recharts';  const contractAddress = process.env.REACT\_APP\_CONTRACT\_ADDRESS;  const DEVICE\_ID = "device-01";  function App() {  const [account, setAccount] = useState(null);  const [readings, setReadings] = useState([]);  const [contract, setContract] = useState(null);  const [provider, setProvider] = useState(null);  const readingsRef = useRef([]);  const connectWallet = async () => {  if (window.ethereum) {  try {  const accounts = await window.ethereum.request({ method: 'eth\_requestAccounts' });  const provider = new ethers.BrowserProvider(window.ethereum);  const signer = await provider.getSigner();  const contractInstance = new ethers.Contract(contractAddress, DataRegistryABI, signer);  setAccount(accounts[0]);  setProvider(provider);  setContract(contractInstance);  console.log("🔗 Terhubung ke kontrak:", contractAddress);  } catch (error) {  console.error("❌ Gagal menghubungkan dompet:", error);  }  } else {  alert("❗ Harap instal MetaMask!");  }  };  useEffect(() => {  if (!contract || !provider) return;  console.log("🟢 Mendengarkan event DataRecorded...");  const onDataRecorded = async (deviceId, temperature, humidity, event) => {  try {  if (deviceId !== DEVICE\_ID) return;  const block = await provider.getBlock(event.blockNumber);  const newReading = {  temp: Number(temperature) / 100.0,  hum: Number(humidity) / 100.0,  time: new Date(block.timestamp \* 1000).toLocaleTimeString('id-ID'),  };  readingsRef.current = [...readingsRef.current, newReading];  setReadings([...readingsRef.current]);  console.log("📥 Data baru:", newReading);  } catch (error) {  console.error("❌ Gagal memproses data event:", error);  }  };  contract.on("DataRecorded", onDataRecorded);  return () => {  contract.off("DataRecorded", onDataRecorded);  };  }, [contract, provider]);  return (  <div className="App" style={{ fontFamily: 'sans-serif', maxWidth: '900px', margin: 'auto', padding: '20px' }}>  <header>  <h1>📊 Dashboard Sensor Blockchain (Realtime)</h1>  {!account ? (  <button onClick={connectWallet} style={{ padding: '10px 15px', fontSize: '16px' }}>  Hubungkan Dompet MetaMask  </button>  ) : (  <div>  <p><strong>Dompet Terhubung:</strong> {account}</p>  <p><strong>Alamat Kontrak:</strong> {contractAddress}</p>  </div>  )}  </header>  <main style={{ marginTop: '30px' }}>  <h2>Data Sensor Masuk (Real-Time)</h2>  <div style={{ height: '300px', marginBottom: '30px' }}>  {readings.length > 0 ? (  <ResponsiveContainer width="100%" height="100%">  <LineChart data={readings}>  <CartesianGrid strokeDasharray="3 3" />  <XAxis dataKey="time" />  <YAxis yAxisId="left" domain={['auto', 'auto']} />  <YAxis yAxisId="right" orientation="right" domain={['auto', 'auto']} />  <Tooltip />  <Legend />  <Line yAxisId="left" type="monotone" dataKey="temp" stroke="#ff7300" name="Suhu (°C)" />  <Line yAxisId="right" type="monotone" dataKey="hum" stroke="#387908" name="Kelembapan (%RH)" />  </LineChart>  </ResponsiveContainer>  ) : (  <p>⏳ Menunggu data masuk...</p>  )}  </div>  <div style={{ border: '1px solid #ccc', padding: '10px', background: '#f9f9f9' }}>  {readings.length === 0 ? (  <p>Menunggu data dari blockchain...</p>  ) : (  <table style={{ width: '100%', borderCollapse: 'collapse' }}>  <thead>  <tr style={{ background: '#eee' }}>  <th style={{ padding: '8px', border: '1px solid #ddd' }}>Waktu</th>  <th style={{ padding: '8px', border: '1px solid #ddd' }}>Suhu (°C)</th>  <th style={{ padding: '8px', border: '1px solid #ddd' }}>Kelembapan (%RH)</th>  </tr>  </thead>  <tbody>  {readings.map((reading, index) => (  <tr key={index}>  <td style={{ padding: '8px', border: '1px solid #ddd' }}>{reading.time}</td>  <td style={{ padding: '8px', border: '1px solid #ddd' }}>{reading.temp.toFixed(2)}</td>  <td style={{ padding: '8px', border: '1px solid #ddd' }}>{reading.hum.toFixed(2)}</td>  </tr>  ))}  </tbody>  </table>  )}  </div>  </main>  </div>  );  }  export default App; |
| --- |

**4. File Qt**

| import sys  import socket  import json  import pandas as pd  from PyQt5.QtCore import Qt, QTimer, QPointF  from PyQt5.QtWidgets import (  QApplication, QMainWindow, QVBoxLayout, QWidget, QTableWidget,  QTableWidgetItem, QHeaderView, QPushButton, QHBoxLayout  )  from PyQt5.QtChart import QChart, QChartView, QLineSeries, QValueAxis  from PyQt5.QtGui import QPainter  TCP\_IP = '127.0.0.1'  TCP\_PORT = 8080  class SensorChart(QMainWindow):  def \_init\_(self):  super().\_init\_()  self.setWindowTitle("Sensor Chart + Transaction Viewer")  self.resize(1000, 700)  self.counter = 0  self.history = []  self.init\_ui()  self.start\_timer()  def init\_ui(self):  self.main\_widget = QWidget()  self.setCentralWidget(self.main\_widget)  main\_layout = QVBoxLayout(self.main\_widget)  # Chart  self.series\_temp = QLineSeries(name="Temperature (°C)")  self.series\_hum = QLineSeries(name="Humidity (%)")  self.chart = QChart()  self.chart.addSeries(self.series\_temp)  self.chart.addSeries(self.series\_hum)  self.chart.setTitle("Sensor Data Chart")  self.axis\_x = QValueAxis()  self.axis\_x.setLabelFormat("%d")  self.axis\_x.setTitleText("Sample")  self.axis\_x.setRange(0, 10)  self.chart.addAxis(self.axis\_x, Qt.AlignBottom)  self.series\_temp.attachAxis(self.axis\_x)  self.series\_hum.attachAxis(self.axis\_x)  self.axis\_y = QValueAxis()  self.axis\_y.setRange(0, 100)  self.axis\_y.setTitleText("Value")  self.chart.addAxis(self.axis\_y, Qt.AlignLeft)  self.series\_temp.attachAxis(self.axis\_y)  self.series\_hum.attachAxis(self.axis\_y)  chart\_view = QChartView(self.chart)  chart\_view.setRenderHint(QPainter.Antialiasing)  main\_layout.addWidget(chart\_view)  # Table  self.table = QTableWidget(0, 3)  self.table.setHorizontalHeaderLabels(["Temperature", "Humidity", "Tx Hash"])  self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  main\_layout.addWidget(self.table)  # Export Button  button\_layout = QHBoxLayout()  self.export\_btn = QPushButton("Export to Excel")  self.export\_btn.clicked.connect(self.export\_excel)  button\_layout.addWidget(self.export\_btn)  main\_layout.addLayout(button\_layout)  def start\_timer(self):  self.timer = QTimer()  self.timer.timeout.connect(self.read\_socket)  self.timer.start(2000) # Baca tiap 2 detik  def read\_socket(self):  try:  with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  s.connect((TCP\_IP, TCP\_PORT))  data = s.recv(1024)  if data:  obj = json.loads(data.decode())  temp = float(obj.get("temperature", 0))  hum = float(obj.get("humidity", 0))  tx\_hash = obj.get("tx\_hash", "N/A")  self.update\_data(temp, hum, tx\_hash)  except Exception as e:  print("Socket error:", e)  def update\_data(self, temp, hum, tx\_hash):  self.counter += 1  self.series\_temp.append(QPointF(self.counter, temp))  self.series\_hum.append(QPointF(self.counter, hum))  # Grafik menumpuk dari kiri, semua data terlihat  self.axis\_x.setRange(0, self.counter)  row = self.table.rowCount()  self.table.insertRow(row)  self.table.setItem(row, 0, QTableWidgetItem(str(temp)))  self.table.setItem(row, 1, QTableWidgetItem(str(hum)))  self.table.setItem(row, 2, QTableWidgetItem(tx\_hash))  self.history.append({  "Temperature": temp,  "Humidity": hum,  "Tx Hash": tx\_hash  })  def export\_excel(self):  df = pd.DataFrame(self.history)  df.to\_excel("sensor\_data.xlsx", index=False)  print("Data exported to sensor\_data.xlsx")  if \_name\_ == "\_main\_":  app = QApplication(sys.argv)  win = SensorChart()  win.show()  sys.exit(app.exec\_()) |
| --- |

**5. Dashboard Grafana**

Berikut merupakan hasil dari dashboard pada Grafana:



**6. Integrasi Blockchain dan Web3**

6.1. Persiapan Lingkungan Pengembangan Blockchain:

* Instalasi Node.js dan npm.
* Instalasi dan konfigurasi Hardhat sebagai lingkungan pengembangan Ethereum lokal.
* Instalasi MetaMask sebagai dompet digital browser untuk interaksi DApp.

6.2. Pengembangan Smart Contract Monitoring.sol (Solidity):

* Menggunakan Solidity versi ^0.8.20.
* Mendefinisikan struct DataPoint untuk menyimpan timestamp, location, temperature (int, dikali 10), dan humidity (int, dikali 10).
* Mendefinisikan array public allDataPoints untuk menyimpan semua DataPoint.
* Mendefinisikan owner untuk kontrol akses.
* Membuat fungsi addDataPoint() dengan modifier onlyOwner untuk menambahkan data baru.
* Membuat event DataPointAdded untuk notifikasi penambahan data.
* Membuat fungsi getDataPointCount() untuk melihat jumlah data.
* Kompilasi smart contract menggunakan npx hardhat compile.

6.3. Deployment Smart Contract ke Jaringan Lokal Hardhat:

* Menjalankan node Hardhat lokal (npx hardhat node) di satu terminal.
* Membuat skrip deployment (scripts/deploy.js) menggunakan ethers.js (via Hardhat).
* Menjalankan skrip deploy (npx hardhat run scripts/deploy.js --network localhost) di terminal lain untuk menerbitkan kontrak ke node Hardhat lokal.
* Mencatat alamat kontrak yang dihasilkan setelah proses deploy.

6.4. Modifikasi Server TCP Rust untuk Mencatat Data ke Blockchain:

* Menambahkan dependencies ethers-rs, dotenv, dan hex ke tcp\_server/Cargo.toml.
* Membuat file .env di root workspace Rust (TUGAS4) untuk menyimpan PRIVATE\_KEY (dari akun Hardhat Node #0) dan CONTRACT\_ADDRESS (dari hasil deploy).
* Menyalin file ABI Monitoring.json (dari proyek-blockchain/artifacts/...) ke dalam folder tcp\_server/abi/.
* Menggunakan macro abigen! di tcp\_server/src/main.rs untuk menghasilkan binding Rust dari ABI.
* Membuat fungsi asinkron write\_to\_blockchain yang:
* Terhubung ke Hardhat Node (http://127.0.0.1:8545).
* Memuat wallet menggunakan PRIVATE\_KEY.
* Membuat instance dari smart contract menggunakan alamat dan ABI.
* Memanggil fungsi addDataPoint di smart contract dengan data sensor yang relevan.
* Mengintegrasikan pemanggilan write\_to\_blockchain di dalam fungsi handle\_connection server Rust.

6.5. Pengembangan DApp Sederhana (HTML, CSS, JavaScript):

* Membuat file index.html (struktur dan sedikit CSS) dan app.js di dalam folder proyek-blockchain.
* Menggunakan Ethers.js (via CDN atau file lokal) di app.js.
* Implementasi fungsi di app.js untuk:
* Menghubungkan ke MetaMask (window.ethereum).
* Membuat instance dari smart contract menggunakan alamat kontrak (yang sudah di-deploy) dan ABI-nya.
* Mengambil data dari smart contract (memanggil getDataPointCount dan allDataPoints).
* Menampilkan data yang diambil dalam format tabel HTML (Waktu (WIB), Lokasi, Suhu, Kelembapan).
* Mengonfigurasi MetaMask untuk terhubung ke jaringan lokal Hardhat (Network Name: Hardhat Local, RPC URL: http://127.0.0.1:8545, Chain ID: 31337).

6.6. Pengujian Siklus Penuh

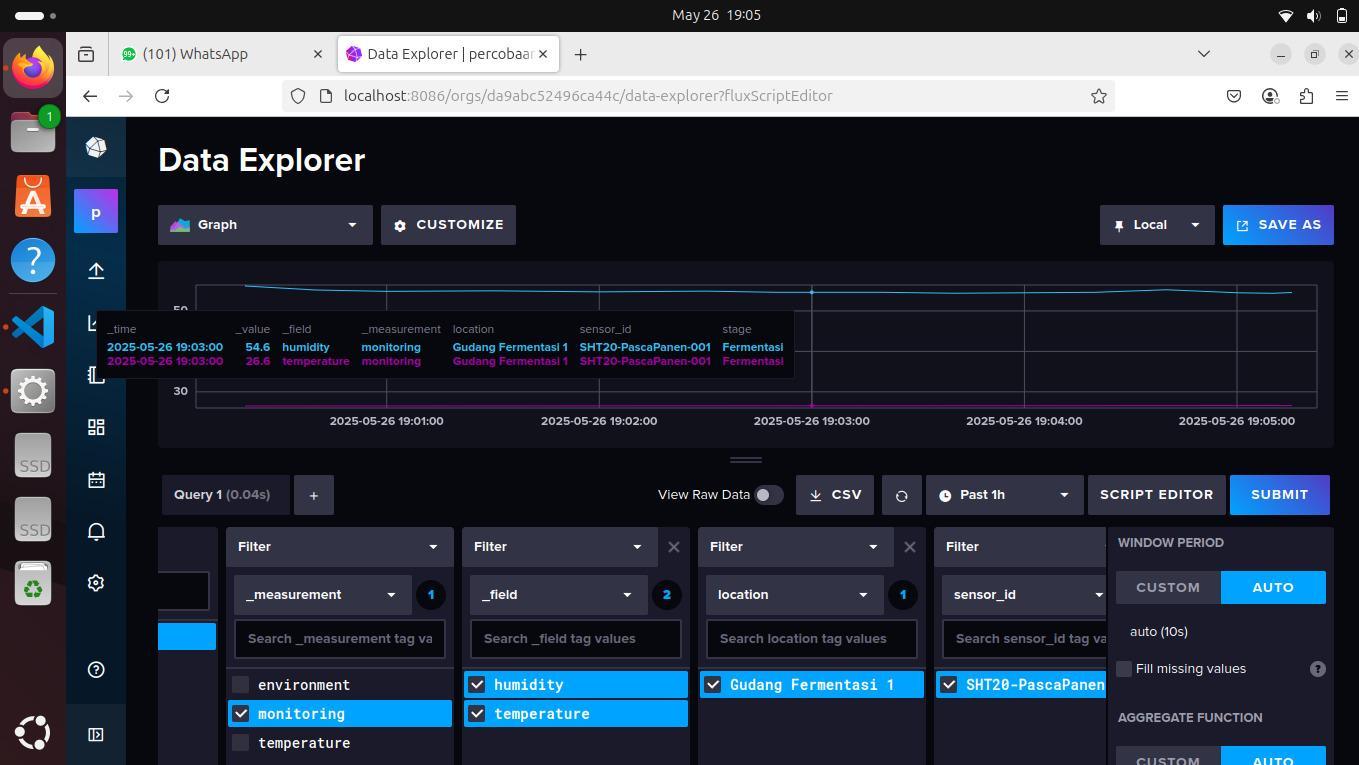
Menjalankan semua komponen (Hardhat Node, Server Rust, Client Rust, DApp di browser) untuk memverifikasi aliran data dari sensor hingga pencatatan dan pembacaan di blockchain.

### 7. Hasil yang Dicapai

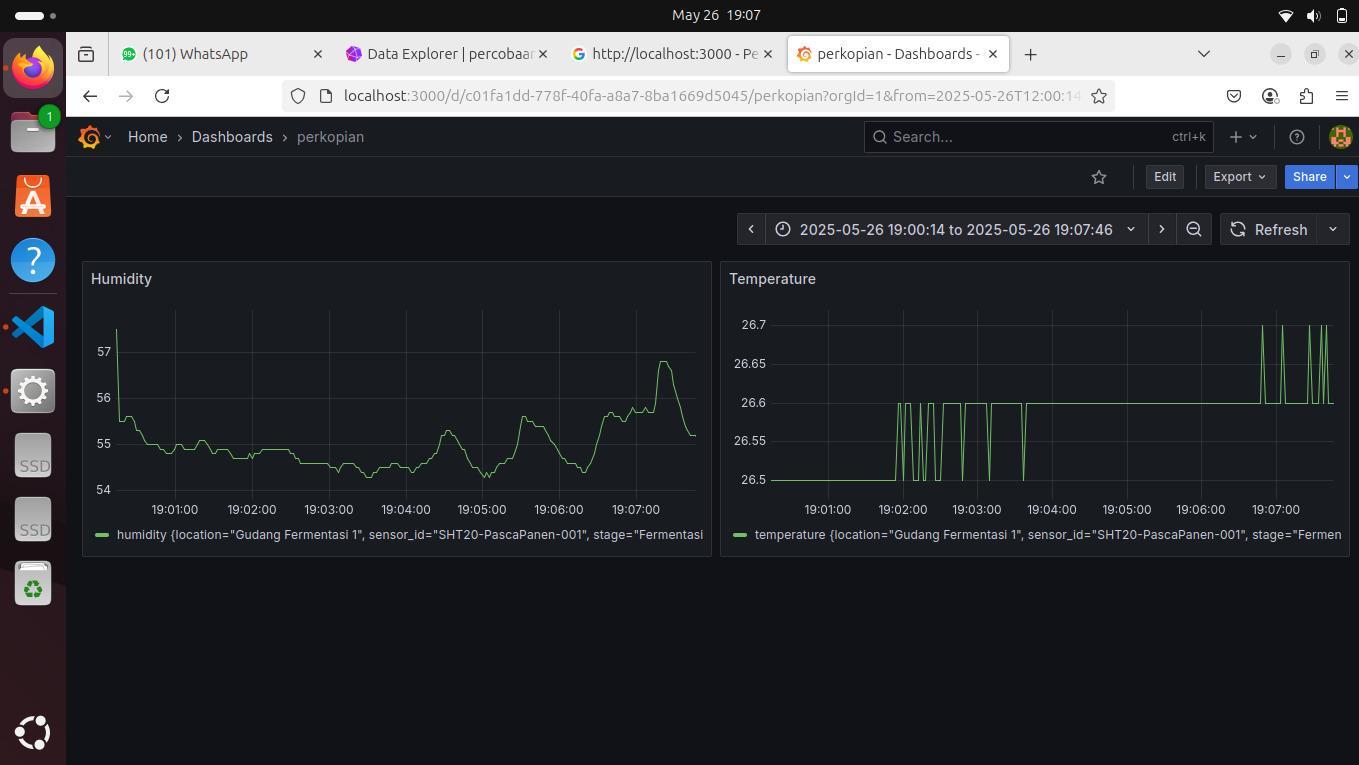
* Prototipe sistem monitoring suhu dan kelembaban real-time berhasil dibangun secara utuh, mulai dari pengambilan data sensor hingga penyimpanan data secara terpusat dan terdesentralisasi.
* Dashboard visualisasi data dikembangkan menggunakan Grafana dan aplikasi desktop PyQt, memungkinkan pengguna untuk melihat data suhu dan kelembaban dalam bentuk grafik dan indikator secara interaktif.
* Smart contract berbasis Solidity telah dirancang dan diuji untuk mencatat data sensor ke dalam blockchain secara transparan dan tidak dapat dimodifikasi.
* Aplikasi terdesentralisasi (DApp) sederhana berhasil dikembangkan untuk menampilkan data langsung dari smart contract, memberikan bukti akses terbuka dan transparansi pencatatan data.
* Integrasi sistem menyeluruh telah tercapai, menghubungkan seluruh komponen mulai dari sensor (disimulasikan), server pengolah data berbasis Rust, penyimpanan data time-series (InfluxDB), pencatatan blockchain (Hardhat/Solidity), hingga antarmuka Web3 melalui DApp.
* Arsitektur sistem end-to-end membuktikan bahwa data lingkungan dapat dipantau dan diamankan secara bersamaan, menunjukkan potensi besar untuk diterapkan pada berbagai kebutuhan pemantauan lingkungan di sektor industri.

# Pengujian dan Hasil

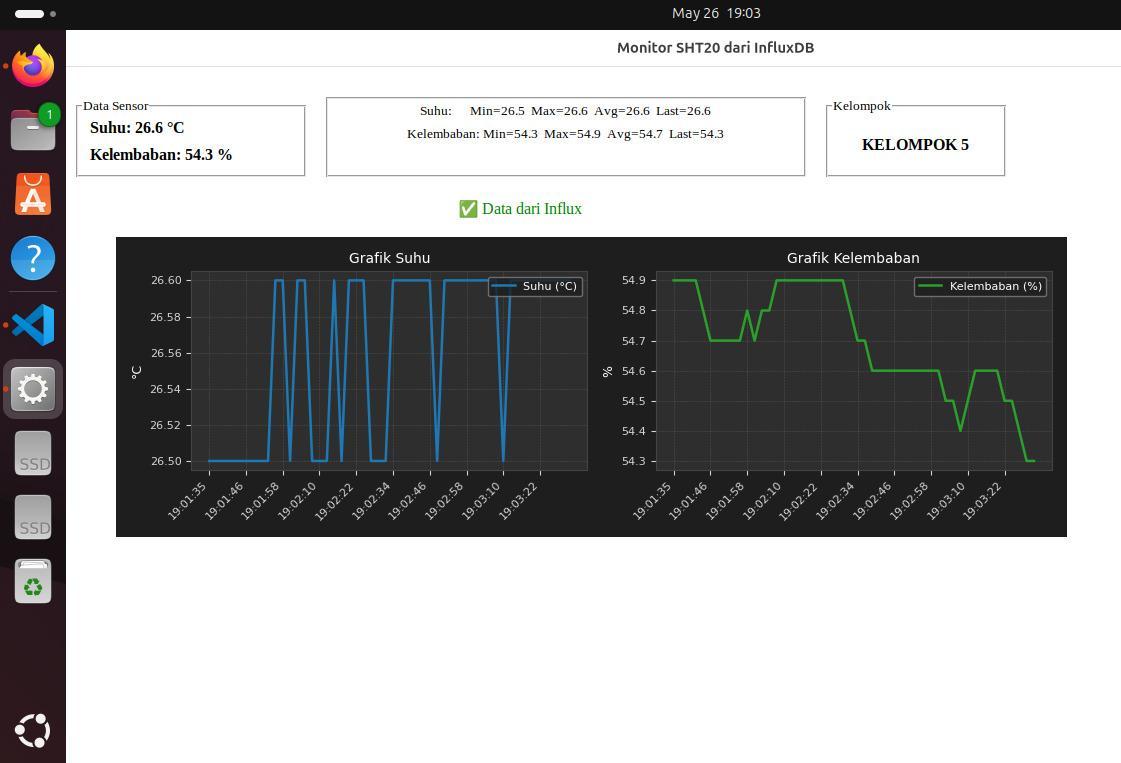
* Hasil penyimpanan di InfluxDB.



* Hasil real-time dashboard di Grafana.



* Analisis suhu dan kelembaban berada dalam rentang optimal (24–30 °C, 50–70%).



# Kesimpulan dan Rekomendasi

Implementasi sistem monitoring suhu dan kelembaban pasca panen kopi berbasis sensor Modbus SHT40 telah berhasil dilaksanakan dengan baik. Sistem ini mampu memantau kondisi fermentasi secara real-time dengan akurasi tinggi melalui pengukuran sensor yang dikirim ke TCP Server, disimpan di InfluxDB, dan divisualisasikan melalui dashboard Grafana. Hasil pengujian menunjukkan bahwa data suhu dan kelembaban berada dalam rentang optimal, sehingga sistem ini efektif dalam membantu petani menjaga kualitas fermentasi kopi secara konsisten dan efisien. Untuk pengembangan ke depan, sistem ini dapat ditingkatkan dengan menambahkan fitur kendali otomatis agar dapat mengatur suhu dan kelembaban secara real-time tanpa intervensi manual. Selain itu, disarankan adanya sistem alarm untuk mendeteksi kondisi ekstrem, penggunaan edge computing untuk mengurangi latensi dan meningkatkan keandalan saat jaringan terganggu, serta integrasi antarmuka berbasis web atau mobile untuk kemudahan akses. Pengembangan fitur analitik prediktif dan keamanan data juga penting untuk meningkatkan nilai tambah dan keberlanjutan sistem dalam jangka panjang.

# Daftar Pustaka

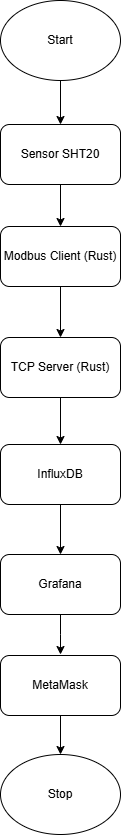
Chbaik, N., Khiat, A., Bahnasse, A., & Ouajji, H. (2024). Blockchain-Assisted IoT Wireless Framework for Equipment Monitoring in Smart Supply Chain: A Focus on Temperature and Humidity Sensing. *IEEE Access*, *12*(September), 117504–117522. https://doi.org/10.1109/ACCESS.2024.3449253

He, Y., & Lv, X. (2021). The Application of Modbus TCP in Universal Testing Machine. *IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, *2021*, 1878–1881. https://doi.org/10.1109/IAEAC50856.2021.9390633

Raina, R., Singh, K. J., & Kumar, S. (2024). Gas, Temperature & Humidity Sensors- based Onion Quality Monitoring System. *IEEE Sensors Letters*, *8*(10), 1–4. https://doi.org/10.1109/LSENS.2024.3462485

# Lampiran

* Diagram sistem lengkap



* InfluxDB

![A screenshot of a computer

AI-generated content may be incorrect.

* Web3

A screenshot of a computer

AI-generated content may be incorrect.

* Dashboard Qt

A screenshot of a computer

AI-generated content may be incorrect.

* Grafana

A screenshot of a computer

AI-generated content may be incorrect.

* Etherscan

A screenshot of a computer

AI-generated content may be incorrect.

* Alur Blockchain

A diagram of a computer program

AI-generated content may be incorrect.