

# PROJECT INTERKONEKSI SISTEM INSTRUMENTASI – VI231418

LAPORAN PROJECT MATA KULIAH INTERKONEKSI SISTEM INSTRUMENTASI

|  |  |
| --- | --- |
| **ANGGOTA :** |  |
| 1. Muhammad Naufal Zuhair | (2042231001) |
| 2. Galuh Pandu Satrio | (2042231019) |
| 3. Yanuar Rahmansyah | (2042231057) |
| **DOSEN PENGAMPU :**   1. Ahmad Radhy, S.Si., M.Si |  |

**PROGRAM STUDI D4 TEKNOLOGI REKAYASA INSTRUMENTASI DEPARTEMEN TEKNIK INSTRUMENTASI**

**FAKULTAS VOKASI**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER 2025**

# DAFTAR ISI

[DAFTAR ISI 2](#_Toc201558166)

[BAB I PENDAHULUAN 1](#_Toc201558167)

[1.1 Latar Belakang 1](#_Toc201558168)

[1.2 Rumusan Masalah 2](#_Toc201558169)

[1.3 Tujuan 3](#_Toc201558170)

[BAB II TINJAUAN PUSTAKA 4](#_Toc201558171)

[2.1 Sensor SHT20 4](#_Toc201558172)

[2.2 MODBUS CLIENT 4](#_Toc201558173)

[2.3 TCP SERVER 5](#_Toc201558174)

[2.4 INFLUX DB 5](#_Toc201558175)

[2.5 GRAFANA 6](#_Toc201558176)

[2.6 BLOCKCHAIN 6](#_Toc201558177)

[2.7 WEB 3 7](#_Toc201558178)

[BAB III METODE 8](#_Toc201558179)

[3.1 Alat & Bahan 8](#_Toc201558180)

[3.2 Desain Arsitektur Sistem 8](#_Toc201558181)

[3.3 Tahap pelaksanaan 9](#_Toc201558182)

[BAB IV 24](#_Toc201558183)

[4.1 Konfigurasi InfluxDb dan Integrasi 24](#_Toc201558184)

[4.2 Analisa 27](#_Toc201558185)

[4.2.1 Analisa Hasil Percobaan 27](#_Toc201558186)

[BAB V KESIMPULAN DAN SARAN 29](#_Toc201558187)

[5.2 Saran 29](#_Toc201558188)

[DAFTAR PUSTAKA 30](#_Toc201558189)

[LAMPIRAN 31](#_Toc201558190)

# BAB I PENDAHULUAN

## Latar Belakang

Jamur tiram (Pleurotus ostreatus) membutuhkan kondisi lingkungan optimal, yaitu suhu 22–28°C dan kelembapan 80–90%. Sistem monitoring berbasis IoT telah banyak diterapkan, seperti oleh (Hudhoifah & Mulyana 2024) yang menggunakan NodeMCU dan DHT22, serta (Ntihung et al. 2024) dengan Arduino dan DHT11. Meski efektif, keduanya masih memiliki keterbatasan akurasi sensor, reliabilitas data, dan ketergantungan mikrokontroler .

Untuk itu, dikembangkan sistem monitoring menggunakan sensor SHT20 Modbus RTU tanpa mikrokontroler, terhubung langsung ke TCP server berbasis Rust, dengan penyimpanan data di InfluxDB dan visualisasi real-time via Grafana. Sistem ini juga dilengkapi integrasi blockchain dan Web3 untuk keamanan dan transparansi data, menjadikannya lebih akurat, stabil, dan aman untuk budidaya jamur tiram.

Ketidaksesuaian kondisi lingkungan, baik terlalu kering maupun terlalu panas, dapat menghambat pertumbuhan miselium dan menurunkan produktivitas jamur. Selain itu, pertumbuhan miselium memerlukan waktu inkubasi selama 40–60 hari, diikuti oleh fase pembentukan tubuh buah yang terjadi dalam 1–2 minggu setelah media dibuka. Lingkungan yang stabil selama proses ini sangat menentukan hasil panen, baik dari segi kualitas maupun kuantitas. Oleh karena itu, pemantauan dan pengaturan suhu serta kelembaban menjadi hal yang krusial dalam budidaya jamur tiram. Oleh karena itu, dibutuhkan sebuah sistem monitoring otomatis yang mampu melakukan pencatatan suhu dan kelembaban secara *real-time*, menyimpan data secara *time-series*, dan dapat diakses baik secara lokal maupun jarak jauh.

Dalam perkembangan teknologi saat ini, kombinasi antara Rust Programming, protokol Modbus RTU, TCP Server, InfluxDB, dan Grafana dapat menjadi solusi ideal. Protokol Modbus RTU telah banyak digunakan dalam sistem industri karena kesederhanaan, efisiensi, dan interoperabilitasnya. Dalam analisis performa Modbus pada berbagai konfigurasi jaringan fisik, sistem ini terbukti tetap efisien dalam pengumpulan data pada lingkungan industri, bahkan saat terdapat hambatan seperti interferensi elektromagnetik atau variasi topologi jaringan (Gaitan & Zagan, 2022).

Untuk menyimpan data suhu dan kelembaban yang diukur secara berkala, InfluxDB digunakan karena kemampuannya dalam menangani data time-series dengan performa tinggi. Dalam studi terkini mengenai pemantauan rantai pasok cerdas berbasis IoT, InfluxDB digunakan sebagai basis penyimpanan data lingkungan karena efisiensinya dalam pencatatan, kueri data, serta kompatibilitasnya dengan platform visualisasi seperti Grafana (Chbaik et al., 2024).

Grafana digunakan untuk menampilkan data suhu dan kelembaban dalam bentuk grafik real-time yang mudah dipahami. Dashboard ini dapat memberikan informasi historis dan terkini, serta peringatan dini bila terjadi penyimpangan dari parameter yang diharapkan. Dalam penelitian sistem pemantauan logistik berbasis blockchain, Grafana terbukti efektif dalam menyediakan visualisasi yang mudah digunakan untuk pemantauan suhu dan kelembaban secara *realtime* (Chbaik et al., 2024).

Dari sisi perangkat lunak, Rust menjadi pilihan utama karena desainnya yang mengutamakan keamanan memori, performa tinggi, dan efisiensi pada sistem dengan batasan sumber daya. Dalam penelitian terkini oleh (Zhang et al., 2024), Rust dinilai unggul dalam mencegah kesalahan yang tidak dapat dipulihkan (unrecoverable errors) berkat sistem tipenya dan kontrol ketat terhadap kepemilikan memori, yang sangat penting dalam aplikasi instrumentasi yang berjalan *realtime*.

Ditambah lagi dengan adanya aplikasi GUI berbasis Qt untuk memantau dan mengendalikan sistem secara lokal, petani kopi dapat memanfaatkan teknologi digital untuk menjaga kualitas Inkubasi secara optimal. Kombinasi ini tidak hanya membantu dalam menjaga kestabilan parameter lingkungan, tetapi juga meningkatkan efisiensi produksi dan kemampuan pelaku usaha untuk melakukan analisis data secara historis.

## Rumusan Masalah

Berdasarkan latar belakang tersebut, maka rumusan masalah dalam proyek ini adalah:

1. Bagaimana membangun sistem monitoring suhu dan kelembapan kumbung jamur tiram menggunakan sensor industrial dengan protokol Modbus RTU?
2. Bagaimana menerapkan arsitektur client-server menggunakan komunikasi TCP untuk menerima dan mengelola data dari sensor?
3. Bagaimana menyimpan data hasil monitoring secara time-series menggunakan InfluxDB?
4. Bagaimana menyediakan visualisasi data secara real-time melalui dashboard Grafana?
5. Bagaimana membangun aplikasi desktop berbasis Qt untuk memonitor data secara lokal?
6. Bagaimana mengintegrasikan sistem monitoring ini dengan blockchain dan Web3 untuk menjamin keamanan dan transparansi data?

## Tujuan

Proyek ini bertujuan untuk:

1. Membangun sistem monitoring suhu dan kelembapan kumbung jamur tiram menggunakan sensor industrial SHT20 Modbus RTU.
2. Menerapkan arsitektur client-server dengan komunikasi TCP untuk pengiriman data sensor ke server.
3. Menyimpan data suhu dan kelembapan secara time-series menggunakan InfluxDB.
4. Menyediakan visualisasi real-time kondisi kumbung jamur tiram melalui dashboard Grafana.
5. Membuat aplikasi desktop berbasis Qt untuk monitoring data secara lokal.
6. Mengintegrasikan sistem monitoring dengan teknologi blockchain dan Web3 untuk keamanan, keabsahan, dan desentralisasi data.

# BAB II TINJAUAN PUSTAKA

## Sensor SHT20

**A close-up of a white device

AI-generated content may be incorrect.**

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).

## 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.

## 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.

## INFLUX DB

A blue and black logo

AI-generated content may be incorrect.

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.

## GRAFANA

A logo with a circle and a circle with a circle and a circle with a circle with a circle with a circle with a circle with a circle with a circle with a circle with a circle with

AI-generated content may be incorrect.

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.

## 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).

**Karakteristik Blockchain**

* Desentralisasi: Tidak dikendalikan oleh satu entitas tunggal, memungkinkan banyak pihak berinteraksi secara langsung dan aman.
* Transparansi dan Immutabilitas: Transaksi yang telah divalidasi dan disimpan dalam blockchain tidak dapat diubah, sehingga meningkatkan kepercayaan dan integritas data.
* Kriptografi dan Signature Digital: Data dilindungi dengan algoritma kriptografi dan ditandatangani secara digital untuk memastikan keaslian dan mencegah pemalsuan (Alam, 2023).

## WEB 3

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).

**Ciri Khas Web3**

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.

# BAB III METODE

## Alat & Bahan

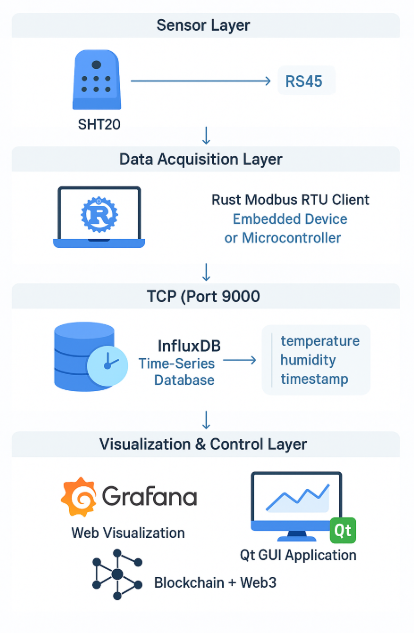
* Alat

1. Sensor SHT20
2. RS485
3. Laptop
4. VirtualBox
5. Ubuntu

* Bahan

1. Dataset Percobaan
2. Modbus
3. InfluxDB
4. Qt
5. Grafana
6. Web3
7. Blockchain

## Desain Arsitektur Sistem



**Arsitektur Sistem Monitoring IoT Berbasis Blockchain**

1. SensorLayer

Sensor SHT20 mengukur suhu dan kelembapan, lalu mengirim data melalui protokol RS485.

1. Data Acquisition Layer

Mikrokontroler atau perangkat embedded menjalankan klien Modbus RTU berbasis Rust untuk membaca data sensor dan mengirimkannya melalui TCP (port 9000).

1. Communication & Storage Layer

Data diterima oleh server dan disimpan di InfluxDB sebagai basis data time-series yang mencatat suhu, kelembapan, dan waktu.

1. Visualization & Control Layer

* Grafana: Menampilkan data secara real-time melalui web.
* Qt GUI: Aplikasi desktop untuk monitoring dan kontrol.
* Blockchain + Web3: Menyimpan data penting secara aman dan transparan di blockchain, serta menyediakan antarmuka kontrol berbasis Web3 DApp.

## Tahap pelaksanaan

1. Kode Rust Modbus Client

- Penjelasan cara membaca sensor menggunakan Modbus RTU.

**Mengatur koneksi serial**

Port serial /dev/ttyUSB0 dibuka dengan pengaturan: baud rate 9600, parity none, stop bit 1, data bit 8, dan timeout 1 detik.

**Menghubungkan ke perangkat sensor**

Koneksi Modbus RTU dibuat ke slave dengan ID tertentu (misalnya ID 1).

**Membaca register sensor** Program membaca 2 register input mulai dari alamat 1:Register 1: Suhu (dalam skala 0.1°C)Register 2: Kelembapan relatif (dalam skala 0.1%)

**Mengolah data mentah** Nilai register dikonversi menjadi: Suhu = register[0] / 10.0 Kelembapan = register[1] / 10.0

- Cuplikan kode utama dan cara kirim data ke TCP Server.

|  |
| --- |
| 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 std::error::Error;  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\_sensor(slave: u8) -> Result<Vec<u16>, Box<dyn Error>> {      let builder = tokio\_serial::new("/dev/ttyUSB0", 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)).await?;      let response = ctx.read\_input\_registers(1, 2).await?;      Ok(response)  }  #[tokio::main]  async fn main() -> Result<(), Box<dyn Error>> {      loop {          match read\_sensor(1).await {              Ok(response) if response.len() == 2 => {                  let temp = response[0] as f32 / 10.0;                  let rh = response[1] as f32 / 10.0;                  println!(" Temp: {:.1} °C | RH: {:.1} %", temp, rh);                  let data = 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(&data)?;                  match TcpStream::connect("127.0.0.1:9000").await {                      Ok(mut stream) => {                          stream.write\_all(json.as\_bytes()).await?;                          stream.write\_all(b"\n").await?;                          println!(" Data dikirim ke TCP server");                      },                      Err(e) => {                          println!(" Gagal konek ke TCP server: {}", e);                      }                  }              },              Ok(other) => {                  println!(" Data tidak lengkap: {:?}", other);              },              Err(e) => {                  println!(" Gagal baca sensor: {}", e);              }          }          sleep(Duration::from\_secs(2)).await;      }  } |

**Membentuk data sensor**

Data dikemas dalam struct SensorData yang memuat waktu, ID sensor, lokasi, tahap proses, suhu, dan kelembapan.

**Menyusun data dalam format JSON**

Struct tersebut diubah menjadi string JSON menggunakan serde\_json.

**Mengirim ke server**

Program mencoba terhubung ke server TCP di alamat 127.0.0.1:9000. Jika berhasil:

* Data JSON dikirim melalui koneksi TCP.
* Diberi karakter newline (\n) sebagai pemisah antar data.

**Seluruh proses dijalankan secara terus-menerus setiap 2 detik menggunakan delay asynchronous.**

1. Rust TCP Listener: Simpan Data Sensor ke InfluxDB dan Blockchain Ethereum

- Cuplikan kode fungsi utama.

|  |
| --- |
| 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;  #[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 setup ---      let influx\_url = "http://localhost:8086/api/v2/write?org=rival%20team&bucket=sensor\_data&precision=s";      let influx\_token = "9BuYLHAV7W\_q1-V3HPHMOtv0xEQMGc24LRxD7FHVYgRDsUHlqD5mkuTblSlz8ZAmJ8sSqUnCtuAEdb2PoVxvZw==";      let http\_client = Client::new();      // --- Ethereum setup ---      let provider = Provider::<Http>::try\_from("http://localhost:8545")?;      let wallet: LocalWallet = "0x95e79c9f7a2a0b1b7e6b3f15732d01b76b3aa3b2bb339e1c2c8b8ba112c135a4"          .parse::<LocalWallet>()?          .with\_chain\_id(1337u64);      let client = Arc::new(SignerMiddleware::new(provider, wallet));      // Baca dan parse ABI dan bytecode dengan benar      let abi\_str = fs::read\_to\_string("build/SensorStorage.abi")?;      let bytecode\_str = fs::read\_to\_string("build/SensorStorage.bin")?;      let abi: Abi = serde\_json::from\_str(&abi\_str)?;      let bytecode = bytecode\_str.trim().parse::<Bytes>()?;      let factory = ContractFactory::new(abi, bytecode, client.clone());      let contract = factory.deploy(())?.send().await?;      println!("Smart contract deployed at: {:?}", contract.address());      // --- TCP Server ---      let listener = TcpListener::bind("0.0.0.0:9000").await?;      println!("TCP Server listening on port 9000...");      loop {          let (socket, addr) = listener.accept().await?;          println!("New connection from {}", addr);          let influx\_url = influx\_url.to\_string();          let influx\_token = influx\_token.to\_string();          let http\_client = http\_client.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 {                  match serde\_json::from\_str::<SensorData>(&line) {                      Ok(data) => {                          println!("Received sensor data: {:?}", data);                          // --- InfluxDB Write ---                          let timestamp = DateTime::parse\_from\_rfc3339(&data.timestamp)                              .unwrap()                              .timestamp();                          let line\_protocol = 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,                              timestamp                          );                          match http\_client                              .post(&influx\_url)                              .header("Authorization", format!("Token {}", influx\_token))                              .header("Content-Type", "text/plain")                              .body(line\_protocol)                              .send()                              .await                          {                              Ok(resp) if resp.status().is\_success() => {                                  println!("InfluxDB: data written");                              }                              Ok(resp) => {                                  println!("InfluxDB error: {}", resp.status());                              }                              Err(e) => {                                  println!("InfluxDB HTTP error: {}", e);                              }                          }                          // --- Ethereum Contract Write ---                          let method\_call = contract      .method::<\_, H256>("storeData", (          timestamp as u64,          data.sensor\_id.clone(),          data.location.clone(),          data.process\_stage.clone(),          (data.temperature\_celsius \* 100.0) as i64,          (data.humidity\_percent \* 100.0) as i64,      ))      .unwrap();  let tx = method\_call.send().await;                          match tx {                              Ok(pending\_tx) => {                                  println!("Ethereum: tx sent: {:?}", pending\_tx);                              }                              Err(e) => {                                  println!("Ethereum tx error: {:?}", e);                              }                          }                      }                      Err(e) => println!("Invalid JSON received: {}", e),                  }              }          });      }  } |

- Penjelasan cara menerima data JSON, parsing, dan simpan ke InfluxDB.

Program ini berfungsi sebagai **TCP server** yang menerima data sensor dalam format **JSON**, memprosesnya, dan menyimpannya ke **database InfluxDB** menggunakan **Line Protocol**.

1. **Menerima Data JSON via TCP**

Server mendengarkan koneksi masuk pada alamat 0.0.0.0:9000 menggunakan TcpListener. Ketika ada klien yang terhubung, data dibaca baris per baris menggunakan BufReader.

1. **Parsing JSON**

Setiap baris data yang diterima diasumsikan berbentuk JSON. Data ini diparsing ke dalam struct SensorData menggunakan serde\_json::from\_str.

1. **Konversi ke Line Protocol (InfluxDB)**

Setelah berhasil diparsing, data dikonversi ke format Line Protocol, yaitu format standar yang digunakan untuk menulis data ke InfluxDB. Format mencakup measurement, tag, field, dan timestamp.

1. **Menyimpan ke InfluxDB**

Data dikirim menggunakan HTTP POST ke endpoint InfluxDB dengan autentikasi token.

1. Kode Python pyqt

|  |
| --- |
| import tkinter as tk  from tkinter import ttk  import requests  import threading  import time  import csv  from io import StringIO  from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  from matplotlib.figure import Figure  from collections import deque  # === Konfigurasi InfluxDB ===  INFLUX\_QUERY\_URL = "http://localhost:8086/api/v2/query"  ORG = "gamtenk"  BUCKET = "kelompokgacor"  TOKEN = "yBoxgtA5n1iXvYpHPEiYnUS8ZIkEtZ5QfiTvo3FKgckP2GE4MYyspnH9xTRaKGx-2WMJ7Y6WCk2cOqzfo7R25g=="  # === Riwayat Data Realtime ===  history\_length = 50  temp\_history = deque(maxlen=history\_length)  rh\_history = deque(maxlen=history\_length)  time\_history = deque(maxlen=history\_length)  def get\_latest\_data():  flux\_query = f'''  from(bucket: "{BUCKET}")  |> range(start: -1m)  |> filter(fn: (r) => r.\_measurement == "monitoring")  |> filter(fn: (r) => r.\_field == "temperature" or r.\_field == "humidity")  |> last()  '''  headers = {  "Authorization": f"Token {TOKEN}",  "Content-Type": "application/vnd.flux",  "Accept": "application/csv"  }  try:  response = requests.post(INFLUX\_QUERY\_URL, params={"org": ORG}, headers=headers, data=flux\_query)  reader = csv.DictReader(StringIO(response.text))  data = {}  for row in reader:  try:  field = row["\_field"]  value = float(row["\_value"])  data[field] = value  except:  continue  if "temperature" in data and "humidity" in data:  return data["temperature"], data["humidity"]  return None  except Exception as e:  print("❌ Exception query Influx:", e)  return None  def get\_data\_range(start\_time, end\_time):  flux\_query = f'''  from(bucket: "{BUCKET}")  |> range(start: {start\_time}, stop: {end\_time})  |> filter(fn: (r) => r.\_measurement == "monitoring")  |> filter(fn: (r) => r.\_field == "temperature" or r.\_field == "humidity")  '''  headers = {  "Authorization": f"Token {TOKEN}",  "Content-Type": "application/vnd.flux",  "Accept": "application/csv"  }  try:  response = requests.post(INFLUX\_QUERY\_URL, params={"org": ORG}, headers=headers, data=flux\_query)  reader = csv.DictReader(StringIO(response.text))  rows = []  for row in reader:  try:  time\_str = row["\_time"]  field = row["\_field"]  value = float(row["\_value"])  rows.append((time\_str, field, value))  except:  continue  return rows  except Exception as e:  print("❌ Exception query Influx:", e)  return []  def update\_data():  while True:  result = get\_latest\_data()  current\_time = time.strftime('%H:%M:%S')  if result:  temp, rh = result  label\_temp.config(text=f"Suhu: {temp:.1f} °C")  label\_rh.config(text=f"Kelembaban: {rh:.1f} %")  status\_label.config(text="✅ Data dari Influx")  temp\_history.append(temp)  rh\_history.append(rh)  time\_history.append(current\_time)  plot\_graph()  else:  label\_temp.config(text="Suhu: ---")  label\_rh.config(text="Kelembaban: ---")  status\_label.config(text="❌ Gagal ambil data")  time.sleep(2)  def plot\_graph():  if not time\_history or not temp\_history or not rh\_history:  return # hindari plot kosong  ax1.clear()  ax2.clear()  fig.patch.set\_facecolor('black')  ax1.set\_facecolor('black')  ax2.set\_facecolor('black')  x = list(range(len(time\_history)))  times = list(time\_history)  temps = list(temp\_history)  rhs = list(rh\_history)  ax1.plot(x, temps, label='Suhu (°C)', color='red', marker='o', linestyle='-')  ax2.plot(x, rhs, label='Kelembaban (%)', color='cyan', marker='x', linestyle='-')  ax1.set\_title("Grafik Suhu", color='white')  ax2.set\_title("Grafik Kelembaban", color='white')  ax1.set\_ylabel("°C", color='white')  ax2.set\_ylabel("%", color='white')  interval = max(1, len(times) // 5)  tick\_positions = x[::interval]  tick\_labels = times[::interval]  ax1.set\_xticks(tick\_positions)  ax1.set\_xticklabels(tick\_labels, rotation=45, ha="right", color='white')  ax2.set\_xticks(tick\_positions)  ax2.set\_xticklabels(tick\_labels, rotation=45, ha="right", color='white')  for ax in [ax1, ax2]:  ax.tick\_params(axis='y', colors='white')  ax.tick\_params(axis='x', colors='white')  ax.grid(True, linestyle='--', alpha=0.3, color='gray')  for spine in ax.spines.values():  spine.set\_color('white')  fig.tight\_layout()  canvas.draw()  # Untuk mencatat waktu unik yang sudah ditambahkan ke tabel  existing\_times = set()  def auto\_refresh\_table():  rows = get\_data\_range("-5m", "now()")  if rows:  grouped = {}  for t, f, v in rows:  if t not in grouped:  grouped[t] = {}  grouped[t][f] = v  for t in sorted(grouped.keys()):  if t in existing\_times:  continue # skip jika sudah ditampilkan  row = grouped[t]  temp = row.get("temperature", "--")  rh = row.get("humidity", "--")  table.insert("", "end", values=(t[11:19], f"{temp:.1f}", f"{rh:.1f}"))  existing\_times.add(t)  root.after(5000, auto\_refresh\_table)  # === GUI ===  root = tk.Tk()  root.title("Monitor SHT20 dari InfluxDB")  root.geometry("900x700")  notebook = ttk.Notebook(root)  tab\_realtime = ttk.Frame(notebook)  tab\_table = ttk.Frame(notebook)  notebook.add(tab\_realtime, text="📡 Realtime")  notebook.add(tab\_table, text="📋 Tabel Historis")  notebook.pack(fill="both", expand=True)  # Realtime Widgets  label\_temp = tk.Label(tab\_realtime, text="Suhu: -- °C", font=("Helvetica", 16))  label\_temp.pack(pady=5)  label\_rh = tk.Label(tab\_realtime, text="Kelembaban: -- %", font=("Helvetica", 16))  label\_rh.pack(pady=5)  status\_label = tk.Label(tab\_realtime, text="Status: ---", fg="blue")  status\_label.pack(pady=5)  fig = Figure(figsize=(7, 4.5), dpi=100)  ax1 = fig.add\_subplot(211)  ax2 = fig.add\_subplot(212)  canvas = FigureCanvasTkAgg(fig, master=tab\_realtime)  canvas.get\_tk\_widget().pack(pady=10)  # Tabel Historis  cols = ("Waktu", "Suhu (°C)", "Kelembaban (%)")  table = ttk.Treeview(tab\_table, columns=cols, show="headings", height=20)  for col in cols:  table.heading(col, text=col)  table.column(col, width=150, anchor="center")  table.pack(padx=10, pady=10, fill="both", expand=True)  # Start Background Thread dan Refresh  threading.Thread(target=update\_data, daemon=True).start()  auto\_refresh\_table()  # Start App  root.mainloop() |

1. Web 3 dan Blockchain
   * + Program scrip.Js

|  |
| --- |
| const contractAddress = "0x43dd0a7c03dac44e63681edaea59946618d2a8b5"; // Ganti dengan alamat kontrak dari Ganache  const abiPath = "abi/SensorStorage.abi";  let chart;  async function loadSensorData() {  const abiRes = await fetch(abiPath);  const abi = await abiRes.json();  const provider = new ethers.BrowserProvider(window.ethereum || "http://localhost:8545");  await provider.send("eth\_requestAccounts", []);  const signer = await provider.getSigner();  const contract = new ethers.Contract(contractAddress, abi, signer);  const filter = contract.filters.DataStored();  const events = await contract.queryFilter(filter, 0, "latest");  const tableBody = document.querySelector("#sensorTable tbody");  tableBody.innerHTML = "";  const labels = [];  const temps = [];  const hums = [];  events.forEach((e) => {  const data = e.args;  const timeStr = new Date(Number(data.timestamp) \* 1000).toLocaleString();  const temp = Number(data.temperature) / 100;  const hum = Number(data.humidity) / 100;  tableBody.innerHTML += `  <tr>  <td>${timeStr}</td>  <td>${data.sensorId}</td>  <td>${data.location}</td>  <td>${data.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,  borderColor: 'rgba(255, 99, 132, 1)',  fill: false  },  {  label: "Humidity (%)",  data: hums,  borderColor: 'rgba(54, 162, 235, 1)',  fill: false  }  ]  },  options: {  responsive: true,  scales: {  y: { beginAtZero: true }  }  }  });  } |

* + - Program Style.css

|  |
| --- |
| body {  font-family: sans-serif;  margin: 20px;  background: #f9f9f9;  }  h1 {  color: #333;  }  table {  width: 100%;  border-collapse: collapse;  margin-top: 20px;  background: white;  box-shadow: 0 2px 4px rgba(0,0,0,0.1);  }  th, td {  padding: 10px;  border: 1px solid #ddd;  text-align: center;  }  button {  padding: 10px 15px;  font-size: 16px;  margin-bottom: 10px;  cursor: pointer;  } |

* + - Program Contrac.sol Sensor

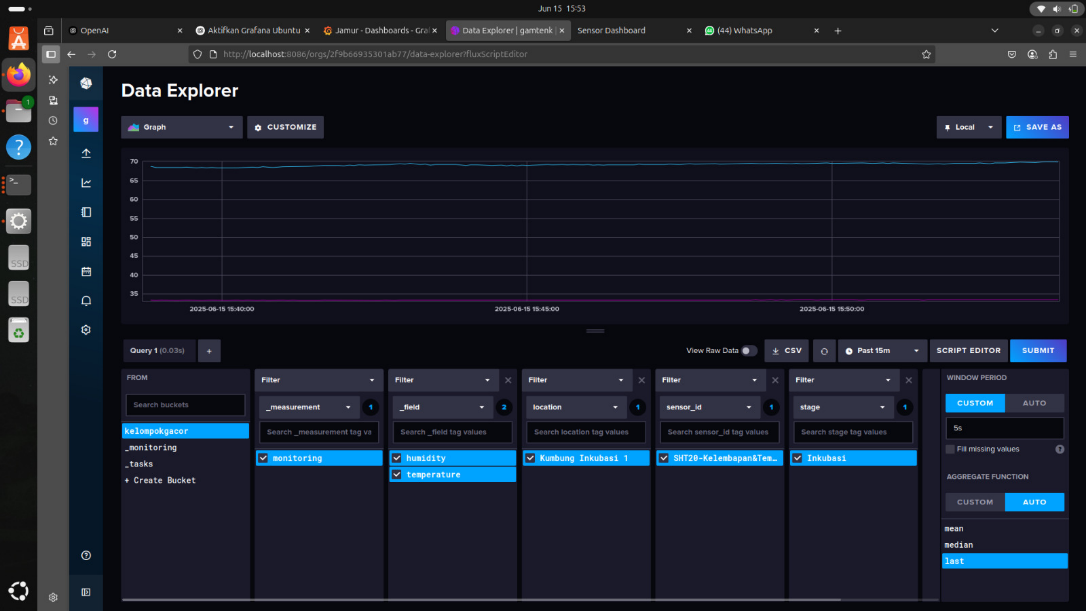
|  |
| --- |
| // SPDX-License-Identifier: MIT  pragma solidity ^0.8.0;  contract SensorStorage {  event DataStored(  uint256 timestamp,  string sensorId,  string location,  string stage,  int256 temperature,  int256 humidity  );  function storeData(  uint256 timestamp,  string memory sensorId,  string memory location,  string memory stage,  int256 temperature,  int256 humidity  ) public {  emit DataStored(timestamp, sensorId, location, stage, temperature, humidity);  }  }  contrac.sol dibagian sensor |

# BAB IV

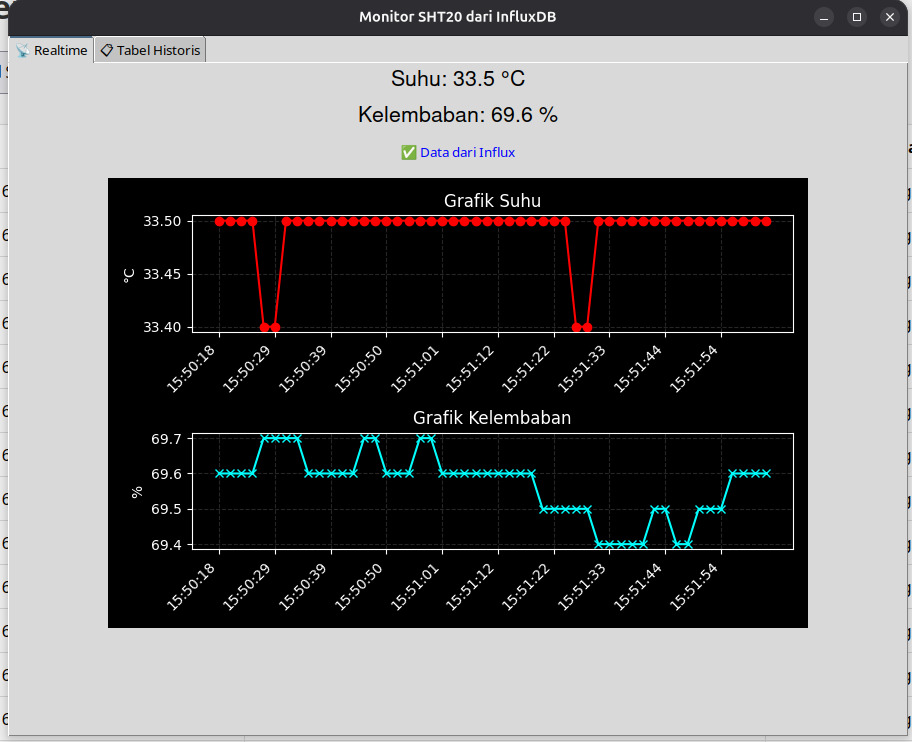
**KONFIGURASI DAN ANALISIS**

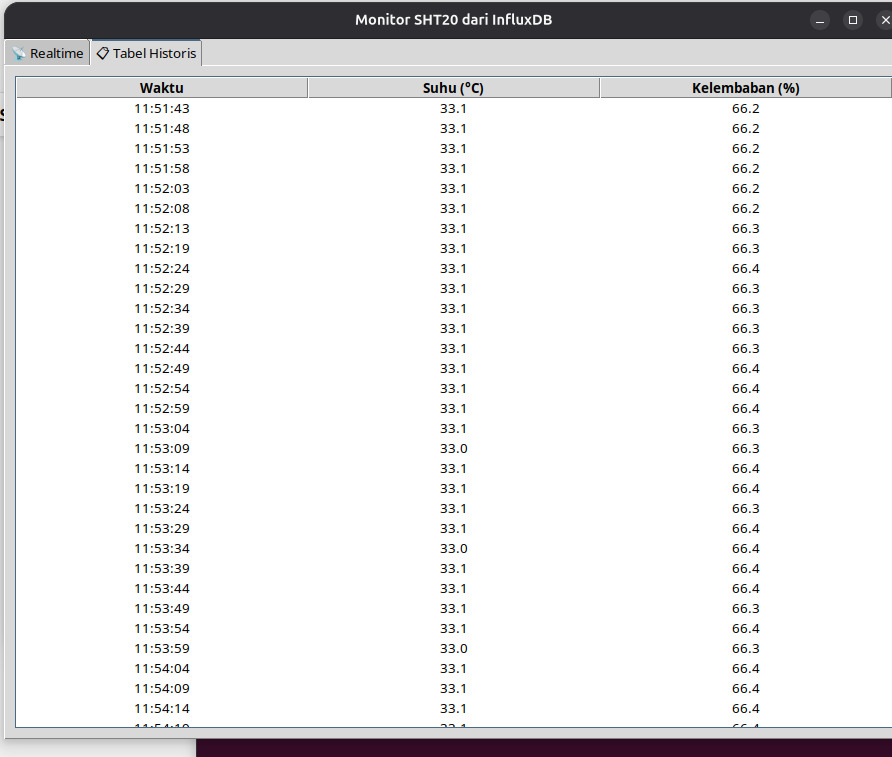
## Konfigurasi InfluxDb dan Integrasi

Hasil *Data Explorer* InfluxDB

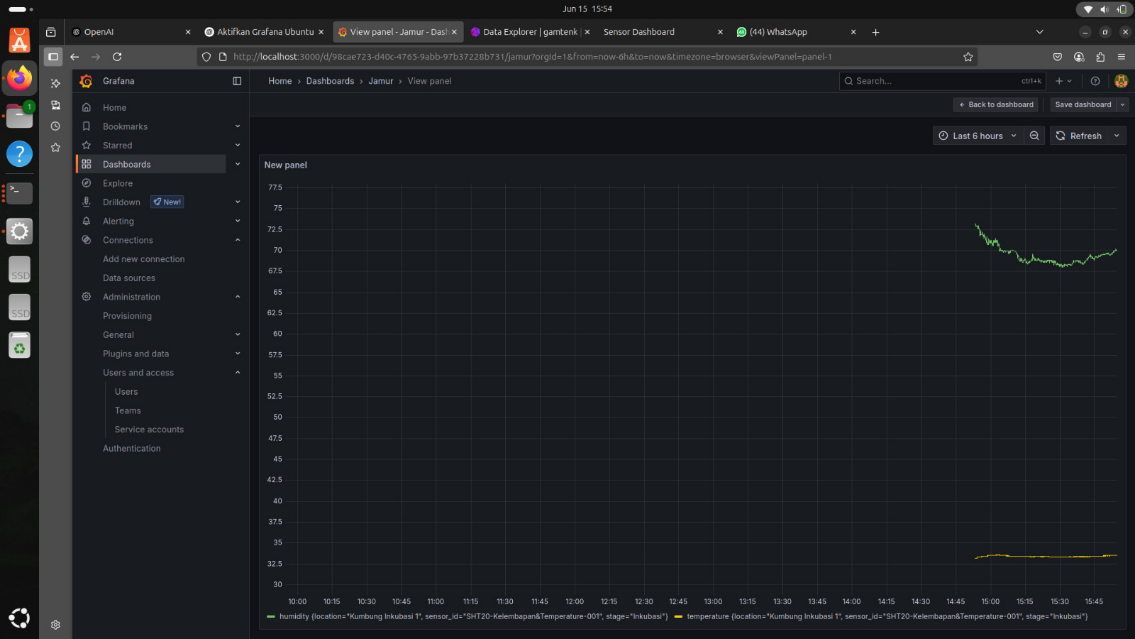


Hasil *Dashboard* Pyqt

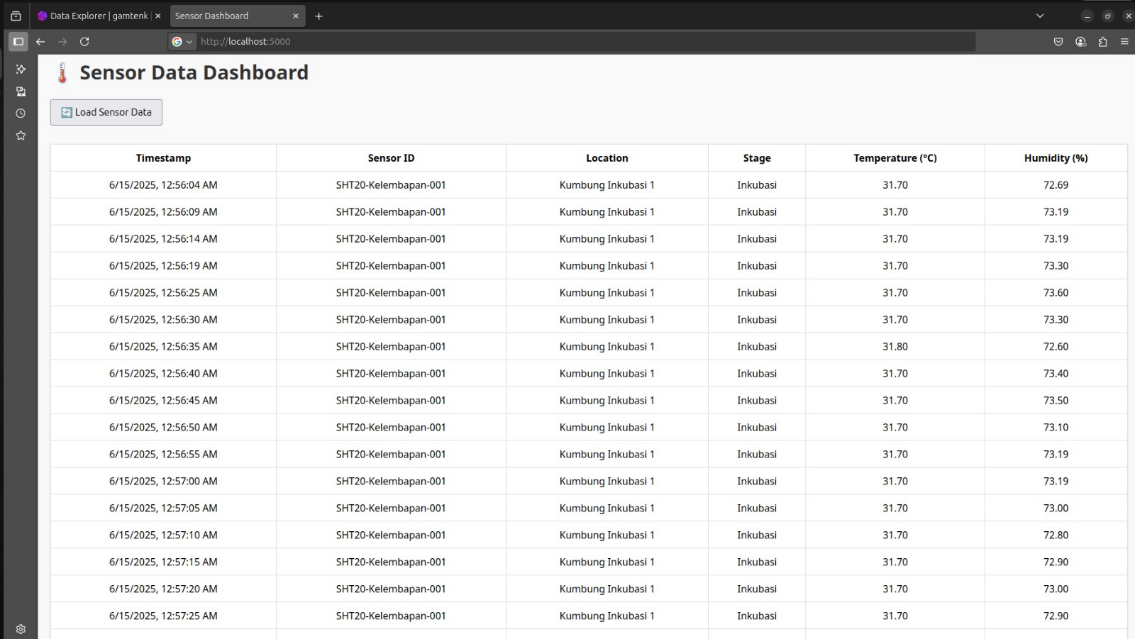


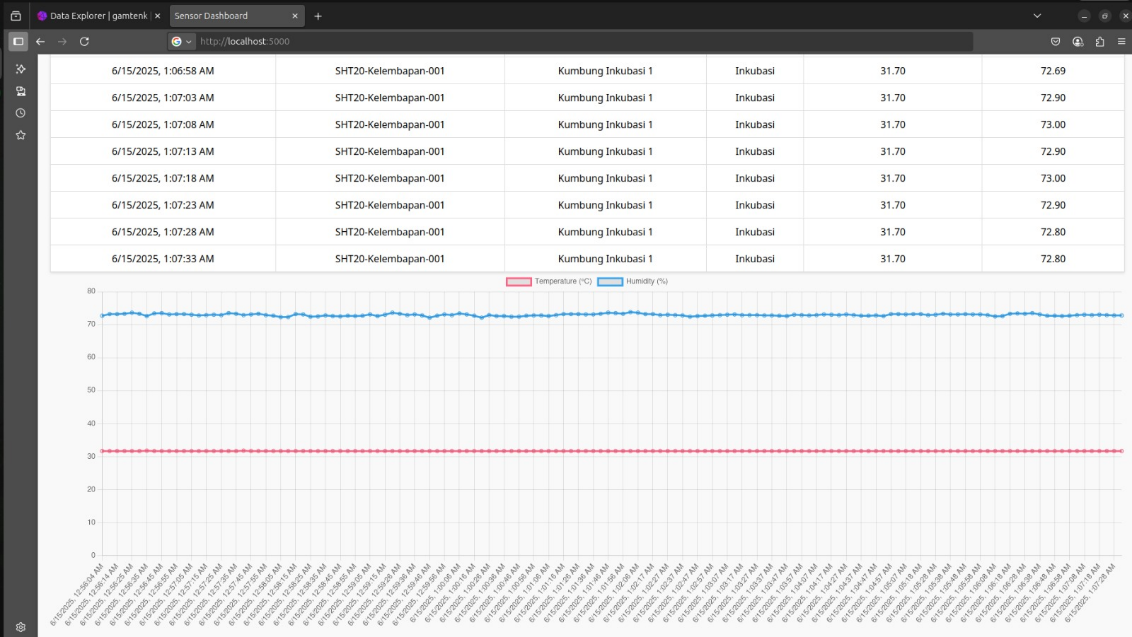


Hasil *Dashboard* Grafa

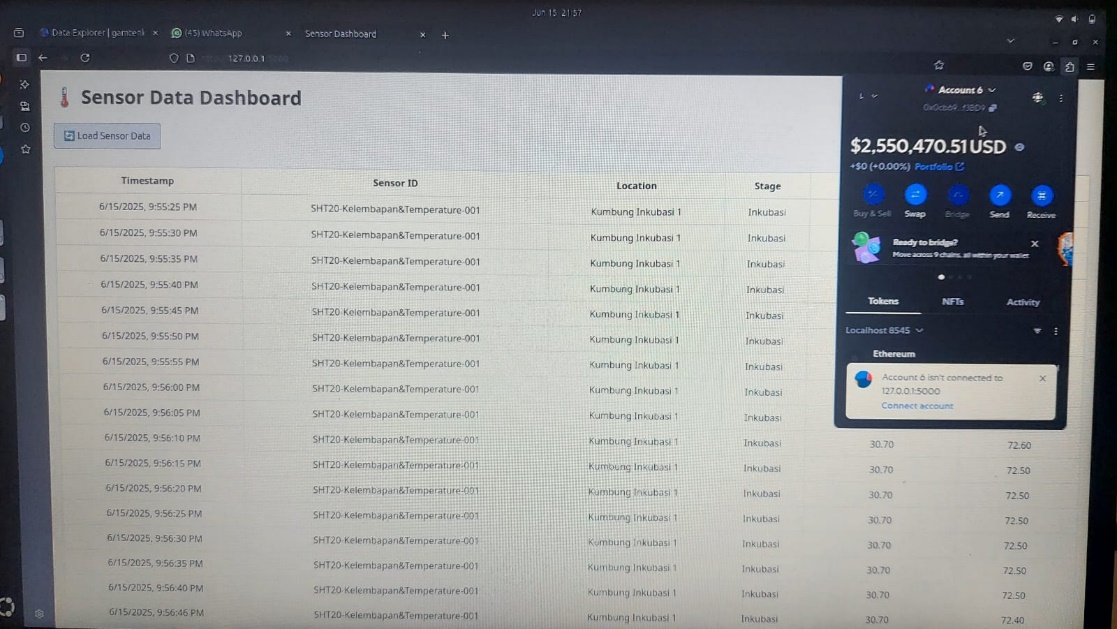


 Dashboard Web3 & Blockchain





Hasil BlockChain



## Analisa

## Analisa Hasil Percobaan

Hasil percobaan menunjukkan bahwa Dashboard yang ditampilkan merupakan antarmuka pemantauan real-time yang mengambil data suhu dan kelembaban dari sensor SHT20 yang tersimpan di InfluxDB. Pada bagian utama dashboard, ditampilkan informasi suhu terkini sebesar 28.5 °C dan kelembaban sebesar 76.3 %, dengan indikator centang hijau yang menunjukkan bahwa data diambil secara aktif dari InfluxDB. Grafik suhu yang ditampilkan berwarna merah dengan garis yang menghubungkan titik-titik data.

Sumbu vertikal grafik ini menunjukkan nilai suhu dalam satuan derajat Celsius (°C), sementara sumbu horizontal menampilkan waktu pengambilan data dalam format jam, menit, detik. Dari grafik suhu terlihat bahwa nilai suhu relatif stabil di sekitar 28.5 °C, namun terdapat dua lonjakan sesaat yang mencapai 28.6 °C pada sekitar pukul 16:49:10 dan 16:49:25. Sementara itu, grafik kelembaban ditampilkan dalam warna biru muda (cyan) dengan konfigurasi sumbu yang sama. Nilai kelembaban menunjukkan tren menurun secara bertahap dari sekitar 77.0 % menjadi 76.3 %, dengan sedikit fluktuasi kecil di tengah grafik. Secara keseluruhan, dashboard ini menampilkan data suhu dan kelembaban secara real-time dengan pembacaan yang stabil dan akurat. Data suhu cenderung konstan dengan sedikit fluktuasi sesaat, sedangkan kelembaban mengalami penurunan bertahap. Sistem ini efektif untuk memantau kondisi lingkungan secara waktu nyata dengan visualisasi yang terintegrasi melalui database InfluxDB.

ruang untuk pengembangan lebih lanjut, khususnya dalam meningkatkan akurasi pada kondisi yang lebih variatif.

* 1. **Kesimpulan**

# BAB V KESIMPULAN DAN SARAN

Kesimpulan dari hasil percobaan ini menunjukkan bahwa Implementasi sistem monitoring suhu dan kelembapan menggunakan sensor SHT20, komunikasi Modbus RTU, dan pengiriman data ke InfluxDB melalui TCP telah berhasil dilakukan dengan baik. Data dari sensor berhasil dibaca secara berkala, diproses dalam format JSON, dikirim melalui koneksi TCP, dan disimpan dalam basis data time-series InfluxDB. Hasil monitoring dapat divisualisasikan secara historis dan real-time, memberikan gambaran kondisi lingkungan secara akurat dan terukur.

## Saran

Sebagai saran, untuk Untuk pengembangan sistem lebih lanjut, berikut beberapa rekomendasi yang dapat dipertimbangkan:

1. Peringatan Dini (Alarm**)**

Menerapkan sistem alarm (notifikasi suara, pesan teks, atau email) ketika suhu atau kelembapan melampaui batas aman, guna mencegah kerusakan atau penurunan kualitas produk.

1. Penyimpanan Data Redundan

Mengintegrasikan sistem penyimpanan cadangan (backup) atau replikasi data untuk menjaga keandalan dan keamanan data historis.

* 1. Penyiraman Air Secara Otomatis

Menambahkan sistem berbasis otomatis seperti menghidupkan sprinkler ketika kelembapan dan suhu kurang dari set point..

# 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*, 117504–117522. https://doi.org/10.1109/ACCESS.2024.3449253

Fatkhurrozi, B., & Setiawan, H. T. (2024). Implementasi Logika Fuzzy pada Sistem Kendali Suhu Dan Kelembaban Udara Ruangan Pengering Biji Kopi Berbasis Mikrokontroller. *Journal of Telecommunication Electronics and Control Engineering (JTECE)*, *6*(1), 50–59. https://doi.org/10.20895/jtece.v6i1.1319

Gaitan, V. G., & Zagan, I. (2022). Modbus Protocol Performance Analysis in a Variable Configuration of the Physical Fieldbus Architecture. *IEEE Access*, *10*, 123942–123955. https://doi.org/10.1109/ACCESS.2022.3224720

Zhang, Y., Li, P., Ding, Y., Wang, L., Williams, D., & Meng, N. (2024). Broadly Enabling KLEE to Effortlessly Find Unrecoverable Errors in Rust. *ACM International Conference Proceeding Series*, 441–451. https://doi.org/10.1145/3639477.3639714

Hudhoifah, M. A., & Mulyana, D. I. (2024). Implementation of Temperature and Humidity Monitoring in Mushroom House for Oyster Mushroom Cultivation Using NodeMCU – ESP8266 in Wirasana Village Purbalingga. MALCOM: Indonesian Journal of Machine Learning and Computer Science, 4(2), 472–480. https://doi.org/10.57152/malcom.v4i2.1222

Ntihung, M. E., Sugiartawan, P., & Willdahlia, A. G. (2024). Sistem Informasi Monitoring Kumbung Jamur Tiram Berbasis Internet of Things. IJEIS (Indonesian Journal of Electronics and Instrumentations Systems), 14(1), 91–100. <https://doi.org/10.22146/ijeis.95513>

Alam, T. (2023). *Blockchain-based Internet of Things: Review, current trends, applications, and future challenges*. Computers, 12(6), 1–30.<https://doi.org/10.3390/computers12010006>

Wang, Q., Li, R., Wang, Q., Chen, S., Ryan, M., & Hardjono, T. (2022). *Exploring Web3 From the View of Blockchain*. arXiv preprint arXiv:2206.08821. https://doi.org/10.48550/arXiv.2206.08821

# LAMPIRAN

 - Diagram sistem lengkap

|  |
| --- |
| [ SENSOR LAYER ]    └── SHT20 (Sensor Suhu & Kelembapan)         └── Komunikasi: RS485  [ DATA ACQUISITION LAYER ]    └── Rust Modbus RTU Client         └── Dijalankan di: Embedded Device         └── Output: Data sensor → TCP (port 9000)  [ COMMUNICATION & STORAGE LAYER ]    └── TCP Server (port 9000)         └── Menerima data dari perangkat akuisisi         └── Menyimpan ke:              └── InfluxDB (Time-series Database)                   ├── temperature                   ├── humidity                   └── timestamp  [ VISUALIZATION & CONTROL LAYER ]    ├── Grafana (Web Dashboard)    ├── Qt GUI App (Aplikasi Desktop)    ├── Web3 DApp (Aplikasi Web berbasis Blockchain)    └── Blockchain Layer (Solana / Ethereum)         ├── Menyimpan log penting (hash, timestamp)         └── Smart Contract untuk otomatisasi atau validasi data |

 - Listing kode penuh (Client dan Server)

Kode Rust Client

|  |
| --- |
| 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 std::error::Error;  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\_sensor(slave: u8) -> Result<Vec<u16>, Box<dyn Error>> {      let builder = tokio\_serial::new("/dev/ttyUSB0", 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)).await?;      let response = ctx.read\_input\_registers(1, 2).await?;      Ok(response)  }  #[tokio::main]  async fn main() -> Result<(), Box<dyn Error>> {      loop {          match read\_sensor(1).await {              Ok(response) if response.len() == 2 => {                  let temp = response[0] as f32 / 10.0;                  let rh = response[1] as f32 / 10.0;                  println!("📡 Temp: {:.1} °C | RH: {:.1} %", temp, rh);                  let data = SensorData {                      timestamp: Utc::now().to\_rfc3339(),                      sensor\_id: "SHT20-Kelembapan&Temperature-001".into(),                      location: "Kumbung Inkubasi 1".into(),                      process\_stage: "Inkubasi".into(),                      temperature\_celsius: temp,                      humidity\_percent: rh,                  };                  let json = serde\_json::to\_string(&data)?;                    match TcpStream::connect("127.0.0.1:9000").await {                      Ok(mut stream) => {                          stream.write\_all(json.as\_bytes()).await?;                          stream.write\_all(b"\n").await?;                          println!("✅ Data dikirim ke TCP server");                      },                      Err(e) => {                          println!("❌ Gagal konek ke TCP server: {}", e);                      }                  }              },              Ok(other) => {                  println!("⚠️ Data tidak lengkap: {:?}", other);              },              Err(e) => {                  println!("❌ Gagal baca sensor: {}", e);              }          }          sleep(Duration::from\_secs(5)).await;      }  } |

Kode Rust Server

|  |
| --- |
| use tokio::net::TcpListener;  use tokio::io::{AsyncBufReadExt, BufReader};  use serde::Deserialize;  use reqwest::Client;  #[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() -> Result<(), Box<dyn std::error::Error>> {      let listener = TcpListener::bind("0.0.0.0:9000").await?;      let influx\_url = "http://localhost:8086/api/v2/write?org=gamtenk&bucket=kelompokgacor&precision=s";      let token = "yBoxgtA5n1iXvYpHPEiYnUS8ZIkEtZ5QfiTvo3FKgckP2GE4MYyspnH9xTRaKGx-2WMJ7Y6WCk2cOqzfo7R25g==";      let client = Client::new();      println!("🚪 TCP Server listening on port 9000...");      loop {          let (socket, addr) = listener.accept().await?;          println!("🔌 Koneksi masuk dari {}", addr);          let client = client.clone();          let influx\_url = influx\_url.to\_string();          let token = token.to\_string();          tokio::spawn(async move {              let reader = BufReader::new(socket);              let mut lines = reader.lines();              while let Ok(Some(line)) = lines.next\_line().await {                  match serde\_json::from\_str::<SensorData>(&line) {                      Ok(data) => {                          println!("📥 Data diterima: {:?}", data);                          // Line Protocol format: measurement,tag1=value1 field1=val1,field2=val2 timestamp                          let line = 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,                              chrono::DateTime::parse\_from\_rfc3339(&data.timestamp)                                  .unwrap()                                  .timestamp()                          );                          // Kirim ke InfluxDB                          let res = client.post(&influx\_url)                              .header("Authorization", format!("Token {}", token))                              .header("Content-Type", "text/plain")                              .body(line)                              .send()                              .await;                          match res {                              Ok(resp) if resp.status().is\_success() => {                                  println!("✅ Data dikirim ke InfluxDB");                              },                              Ok(resp) => {                                  println!("⚠️ Gagal kirim ke InfluxDB: {}", resp.status());                              },                              Err(e) => {                                  println!("❌ HTTP Error: {}", e);                              }                          }                      },                      Err(e) => println!("❌ Format JSON tidak valid: {}", e),                  }              }          });      }  } |

* + - Link Repository GitHub

<https://github.com/MNaufalZhr/Sistem-Monitoring-Temperature-dan-Kelembaban-Kumbung-Jamur-Tiram-untuk-Menjaga-Kualitas-Inkubasi.git>