

**TUGAS MATA KULIAH**  
**INTERKONEKSI SISTEM INSTRUMENTASI**

Dosen : Ahmad Radhy, S.SI., M.SI.

**“ Sistem Monitoring Temperatur dan Kelembaban  
Otomatis untuk Greenhouse Cabai Berbasis Modbus RTU  
dengan Integrasi Blockchain”**



**Disusun Oleh :**

Muhammad Ali Makki (2042231023)  
Aireka Maulana Erawan (2042231047)  
Syahira Arliya Putri Subekti (2042231051)

**PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI**  
**DEPARTEMEN TEKNIK INSTRUMENTASI**  
**FAKULTAS VOKASI**  
**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**  
**2025**

## DAFTAR ISI

DAFTAR ISI .....	II
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan Penelitian .....	2
BAB II TINJAUAN PUSTAKA .....	3
2.1 State of the Art.....	3
2.2 <i>Greenhouse</i> dan Budidaya Cabai.....	5
2.3 Sistem Monitoring Lingkungan Otomatis .....	5
2.4 Sensor SHT20 dan Komunikasi Modbus RTU .....	5
2.5 Penyimpanan Data Time-Series Menggunakan InfluxDB .....	6
2.6 Visualisasi Data dengan Grafana.....	6
2.7 Antarmuka Aplikasi dengan Qt .....	6
2.8 Teknologi Blockchain dan Web3 .....	6
2.9 Integrasi IoT dan Blockchain dalam Pertanian.....	7
BAB III METODOLOGI PENELITIAN .....	8
3.1 Pendekatan Penelitian.....	8
3.2 Pengambilan Data.....	8
3.3 Teknik Analisis Data .....	10
3.4 Perhitungan Konsumsi Energi dan Estimasi Biaya .....	11
3.5 Langkah-langkah .....	11
3.6 Arsitektur Sistem .....	15
3.6 Perancangan Sistem Monitoring.....	17
3.7 Komunikasi Data .....	19
3.8 Penyimpanan dan Visualisasi Data.....	20
3.9 Desain dan pengembangan aplikasi desktop Qt .....	21
3.10 Integrasi Blockchain dan Web3 .....	21
3.11 Implementasi dan Kode Program .....	21
BAB IV HASIL DAN PEMBAHASAN .....	23
4.1 Analisis dan Evaluasi Sistem.....	23
4.2 Perhitungan Konsumsi Energi dan Estimasi Biaya .....	28
BAB V KESIMPULAN DAN SARAN .....	30
4.1 Kesimpulan.....	30
4.2 Saran .....	30
DAFTAR PUSTAKA .....	32
LAMPIRAN .....	33

## BAB I

### PENDAHULUAN

#### 1.1 Latar Belakang

Pertanian merupakan sektor yang berperan besar dalam mendukung ketahanan pangan dan ekonomi nasional, terutama di negara agraris seperti Indonesia. Salah satu komoditas hortikultura unggulan yang memiliki nilai ekonomi tinggi serta tingkat konsumsi yang tinggi di masyarakat adalah cabai. Berdasarkan data dari Badan Pusat Statistik (BPS), konsumsi cabai merah di Indonesia mencapai sekitar 2,46 kg per kapita per tahun, sedangkan produksi nasional mencapai lebih dari 1,2 juta ton pada tahun 2023 (BPS, 2024). Permintaan pasar yang tinggi menjadikan cabai sebagai komoditas strategis, namun juga rentan mengalami fluktuasi harga akibat gangguan pada proses budidaya.

Tanaman cabai sangat sensitif terhadap perubahan iklim, khususnya temperatur dan kelembaban. Ketidaksesuaian kondisi lingkungan dapat menyebabkan penurunan produktivitas, serangan hama dan penyakit, serta menurunnya kualitas hasil panen. Oleh karena itu, pengendalian lingkungan tumbuh menjadi faktor penting dalam budidaya cabai.

*Greenhouse* atau rumah kaca menjadi salah satu solusi inovatif untuk menjaga kestabilan kondisi lingkungan tanaman. Namun, sebagian besar pengelolaan greenhouse di Indonesia masih dilakukan secara manual atau semi-otomatis, yang rentan terhadap keterlambatan respons terhadap perubahan suhu dan kelembaban. Hal ini berdampak pada ketidakefisienan manajemen lingkungan serta peningkatan risiko gagal panen.

Seiring dengan perkembangan teknologi di bidang instrumentasi dan Internet of Things (IoT), sistem monitoring lingkungan berbasis sensor menjadi solusi potensial. Sensor suhu dan kelembaban seperti SHT20, yang mendukung komunikasi Modbus RTU, memungkinkan pengambilan data yang andal, akurat, serta dapat diintegrasikan dengan sistem otomasi skala industri (Pranto *et al.*, 2021).

Lebih jauh, dalam praktik pertanian modern, teknologi blockchain mulai digunakan untuk menciptakan sistem ketertelusuran (*traceability*) yang aman dan transparan. Dengan menyimpan data lingkungan secara permanen di blockchain, semua pihak dalam rantai pasok—mulai dari petani, distributor, hingga konsumen dapat mengakses riwayat budidaya tanaman yang tidak dapat dimanipulasi (Salah *et al.*, 2019; Marchesi *et al.*, 2022). Hal ini sangat penting untuk menjamin bahwa produk pertanian, seperti cabai, diproduksi sesuai dengan standar organik atau sertifikasi tertentu.

Integrasi antara teknologi sensor IoT dan blockchain yang dikombinasikan dengan smart contract dapat memperkuat aspek akuntabilitas dan efisiensi sistem pertanian presisi (Wang *et al.*, 2021). Implementasi teknologi ini telah dibuktikan pada komoditas lain

seperti kedelai dan agri-food secara global, dan menunjukkan potensi adaptasi yang tinggi untuk komoditas lokal seperti cabai (Salah *et al.*, 2019).

Oleh karena itu, tugas ini bertujuan untuk merancang dan mengembangkan sistem monitoring suhu dan kelembaban otomatis untuk greenhouse cabai yang tidak hanya memanfaatkan teknologi sensor dan visualisasi real-time, tetapi juga mengintegrasikan teknologi *blockchain* untuk mendukung aspek traceability dan akuntabilitas dalam budidaya pertanian. Diharapkan sistem ini dapat menjadi solusi teknologi cerdas yang adaptif dan aplikatif bagi pertanian presisi (*precision agriculture*) di Indonesia.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana merancang sistem monitoring suhu dan kelembaban otomatis untuk greenhouse cabai menggunakan sensor yang berkomunikasi dengan protokol Modbus RTU?
2. Bagaimana cara menyimpan dan menampilkan data suhu dan kelembaban secara real-time agar mudah dipantau oleh pengguna?
3. Bagaimana mengintegrasikan teknologi blockchain ke dalam sistem untuk menjamin keamanan dan transparansi data lingkungan dalam greenhouse?

## 1.3 Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Mengembangkan sistem monitoring suhu dan kelembaban otomatis pada greenhouse cabai menggunakan sensor yang terhubung melalui Modbus RTU.
2. Menerapkan penyimpanan data time-series dan visualisasi real-time menggunakan InfluxDB dan Grafana untuk kemudahan pemantauan.
3. Mengintegrasikan sistem dengan teknologi blockchain untuk menjamin transparansi dan keamanan data, serta mendukung traceability dalam proses budidaya cabai di greenhouse.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 State of the Art

No	Referensi (Penulis, Tahun)	Fokus Penelitian	Metode/Implementasi	Hasil / Temuan Utama
1	Setiawan <i>et al.</i> , 2022	Pengendalian lingkungan greenhouse untuk cabai	Studi kondisi suhu dan kelembaban ideal	Suhu 25–30°C siang, 18–22°C malam, kelembaban 60–70% optimal
2	Rahman <i>et al.</i> , 2023	Dampak iklim mikro pada pertumbuhan cabai	Observasi dan analisis kondisi tanaman	Perubahan suhu dan kelembaban berpengaruh signifikan pada hasil panen
3	Ahmed <i>et al.</i> , 2021	Sistem monitoring otomatis greenhouse	Sistem IoT real-time dengan sensor suhu & kelembaban	Meningkatkan efisiensi pengelolaan dan monitoring lingkungan
4	Kim <i>et al.</i> , 2020	Akurasi sensor SHT20 untuk aplikasi pertanian	Pengujian sensor digital	Sensor memiliki akurasi $\pm 0.3^\circ\text{C}$ suhu dan $\pm 2\%$ kelembaban RH
5	Zhao <i>et al.</i> , 2021	Komunikasi Modbus RTU dalam sensor industri	Implementasi protokol Modbus RTU	Protokol stabil dan handal untuk komunikasi sensor dan PLC
6	Nguyen & Lee, 2021	Evaluasi performa InfluxDB pada aplikasi IoT	Benchmarking dan pengujian database time-series	InfluxDB mampu menyimpan data waktu nyata dengan throughput tinggi
7	Patel & Desai, 2022	Visualisasi data IoT dengan Grafana	Pengembangan dashboard monitoring	Dashboard interaktif untuk

No	Referensi (Penulis, Tahun)	Fokus Penelitian	Metode/Implementasi	Hasil / Temuan Utama
				visualisasi data sensor real-time
8	Singh <i>et al.</i> , 2020	Pengembangan GUI monitoring menggunakan Qt	Pembuatan aplikasi desktop dengan Qt	Antarmuka ramah pengguna dan multiplatform
9	Li <i>et al.</i> , 2023	Integrasi blockchain di sistem monitoring pertanian	Pengembangan prototipe berbasis blockchain	Meningkatkan transparansi dan traceability data lingkungan
10	Zhou <i>et al.</i> , 2021	Manajemen data aman di smart farming menggunakan blockchain	Analisis keamanan dan implementasi blockchain	Data terlindungi dan tidak dapat dimanipulasi
11	Rahim <i>et al.</i> , 2023	Aplikasi Web3 dan blockchain di monitoring pertanian	Pengembangan DApp dan smart contract	Interaksi desentralisasi dan transparan di sistem pertanian
12	Gupta <i>et al.</i> , 2022	Review IoT dan blockchain untuk pertanian pintar	Studi literatur dan analisis teknologi	IoT dan blockchain efektif meningkatkan efisiensi dan keamanan
13	Mandal <i>et al.</i> , 2023	Smart farming dengan IoT dan blockchain	Pengembangan sistem terintegrasi	Sistem terintegrasi meningkatkan akurasi monitoring dan transparansi

## 2.2 Greenhouse dan Budidaya Cabai

*Greenhouse* adalah struktur bangunan yang berfungsi untuk menciptakan kondisi lingkungan terkendali agar pertumbuhan tanaman dapat dioptimalkan. Dalam budidaya cabai, suhu dan kelembaban memiliki peran penting karena tanaman cabai sangat sensitif terhadap perubahan iklim mikro. Temperatur ideal pertumbuhan cabai berkisar antara 25–30°C pada siang hari dan 18–22°C pada malam hari, sedangkan kelembaban ideal berada pada kisaran 60–70% (Setiawan *et al.*, 2022). Ketidaksesuaian kondisi ini dapat menyebabkan penurunan produktivitas, meningkatkan serangan hama, dan memperlambat pertumbuhan tanaman (Rahman *et al.*, 2023).



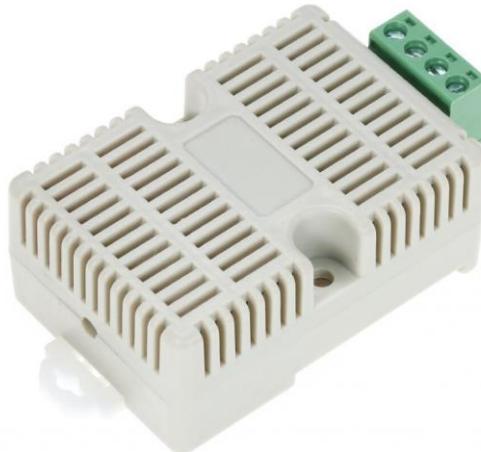
**Gambar 2.1 GreenHouse**

## 2.3 Sistem Monitoring Lingkungan Otomatis

Sistem monitoring otomatis memungkinkan akuisisi data suhu dan kelembaban secara *real-time* dan kontinyu. Hal ini mendukung pengambilan keputusan yang lebih cepat dan akurat dalam pengelolaan pertanian berbasis teknologi. Sistem seperti ini telah terbukti meningkatkan efisiensi irigasi, pemupukan, dan pengendalian iklim di dalam greenhouse (Ahmed *et al.*, 2021). Monitoring otomatis juga mencegah ketergantungan pada pencatatan manual yang berpotensi tidak konsisten.

## 2.4 Sensor SHT20 dan Komunikasi Modbus RTU

Sensor SHT20 adalah sensor digital berbasis I2C yang memiliki akurasi tinggi untuk pengukuran suhu dan kelembaban. Sensor ini memiliki tingkat akurasi  $\pm 0.3^{\circ}\text{C}$  untuk suhu dan  $\pm 2\%$  RH untuk kelembaban, serta banyak digunakan dalam sistem monitoring berbasis mikrokontroler (Kim *et al.*, 2020). Protokol komunikasi yang digunakan dalam sistem ini adalah Modbus RTU, sebuah protokol industri yang handal dan efisien dalam komunikasi serial antar perangkat (Zhao *et al.*, 2021). Modbus RTU juga mendukung banyak perangkat seperti PLC, HMI, dan sensor-sensor digital untuk komunikasi secara master-slave.



**Gambar 2.2 SHT 20**

## 2.5 Penyimpanan Data Time-Series Menggunakan InfluxDB

InfluxDB merupakan database time-series yang dioptimalkan untuk menyimpan data berdasarkan waktu, seperti data suhu dan kelembaban. InfluxDB sangat cocok untuk aplikasi IoT karena mendukung write throughput tinggi dan query yang efisien (Nguyen & Lee, 2021). Selain itu, InfluxDB memungkinkan data dianalisis secara historis maupun real-time yang sangat berguna untuk evaluasi pertumbuhan tanaman dan kontrol lingkungan.

## 2.6 Visualisasi Data dengan Grafana

Grafana digunakan untuk membuat dashboard visualisasi data yang menarik dan informatif. Dalam konteks sistem monitoring greenhouse, data yang dikumpulkan oleh sensor dan disimpan di InfluxDB dapat ditampilkan secara interaktif dalam bentuk grafik di Grafana (Patel & Desai, 2022). Grafana juga mendukung fitur alerting sehingga pengguna dapat langsung mengetahui apabila suhu atau kelembaban berada di luar batas normal.

## 2.7 Antarmuka Aplikasi dengan Qt

Qt merupakan framework open-source yang banyak digunakan untuk membangun aplikasi GUI berbasis C++ atau Python. Qt mendukung pembuatan antarmuka yang interaktif dan multiplatform, sehingga pengguna dapat memantau kondisi greenhouse secara lokal dengan tampilan yang ramah pengguna (Singh *et al.*, 2020).

## 2.8 Teknologi Blockchain dan Web3

*Blockchain* adalah teknologi pencatatan data terdesentralisasi yang bersifat *immutable* dan transparan. Dalam pertanian, blockchain digunakan untuk menjamin keaslian dan ketertelusuran (traceability) data, seperti riwayat suhu dan kelembaban

selama proses budidaya (Li *et al.*, 2023). Data dari sensor dapat ditulis ke blockchain secara periodik untuk mencegah manipulasi dan memastikan kepercayaan konsumen terhadap produk (Zhou *et al.*, 2021). Web3 memungkinkan interaksi dengan blockchain secara langsung melalui aplikasi terdesentralisasi (dApp), yang memperluas penggunaan blockchain dalam sistem pertanian pintar (Rahim *et al.*, 2023).



**Gambar 2.3 Blockchain**

## 2.9 Integrasi IoT dan Blockchain dalam Pertanian

Penggabungan IoT dan blockchain dikenal sebagai Smart Agriculture 4.0. Sensor IoT berfungsi mengumpulkan data lingkungan, sementara blockchain menjamin keamanan dan integritas data (Gupta *et al.*, 2022). Dengan sistem ini, petani dapat memberikan bukti digital mengenai proses budidaya mereka kepada konsumen atau pihak regulator. Sistem seperti ini telah diuji dan diterapkan di beberapa negara untuk meningkatkan efisiensi dan transparansi produksi pertanian (Mandal *et al.*, 2023).

## **BAB III**

### **METODOLOGI PENELITIAN**

#### **3.1 Pendekatan Penelitian**

Penelitian ini menggunakan pendekatan metode eksperimen terapan (*applied experimental research*) yang bertujuan untuk menghasilkan sebuah solusi berbasis teknologi dalam bentuk sistem monitoring lingkungan greenhouse untuk tanaman cabai. Pendekatan ini dipilih karena menekankan pada penerapan teknologi informasi, khususnya Internet of Things (IoT) dan blockchain, untuk memecahkan masalah nyata dalam dunia pertanian modern.

Metodologi eksperimen memungkinkan peneliti melakukan percobaan langsung terhadap sistem yang dirancang, mengamati reaksi dan hasil dari data yang diperoleh, serta menganalisis efektivitas sistem tersebut secara kuantitatif. Dengan mengintegrasikan teknologi pengumpulan data, protokol komunikasi, penyimpanan blockchain, dan visualisasi data real-time, penelitian ini bertujuan untuk menciptakan solusi komprehensif dan efisien dalam pengawasan suhu, kelembapan, dan estimasi biaya energi listrik di dalam rumah kaca tanaman cabai.

#### **3.2 Pengambilan Data**

Pengambilan data dalam penelitian ini difokuskan pada proses pencatatan suhu dan kelembapan udara di dalam greenhouse untuk tanaman cabai. Sistem yang dikembangkan tidak menggunakan mikrokontroler seperti ESP32, tetapi mengandalkan protokol komunikasi industri, yaitu Modbus RTU. Sensor suhu dan kelembapan diintegrasikan langsung dengan sistem Modbus RTU yang memiliki keunggulan dalam stabilitas dan kompatibilitas dengan perangkat industri lainnya.

Langkah pertama dimulai dari proses inisialisasi sistem dan aktivasi sensor untuk membaca nilai temperatur dan kelembapan udara di dalam greenhouse. Data lingkungan ini kemudian dikirimkan melalui protokol Modbus RTU ke server utama yang berbasis RUST. Server RUST berperan sebagai pusat kendali data yang mengarahkan aliran data ke dua jalur penyimpanan yang berbeda dan memiliki fungsi masing-masing.

Jalur pertama adalah penyimpanan data ke dalam jaringan blockchain. Pada jalur ini, data yang diterima dari Modbus akan disimpan sebagai event yang immutable dalam jaringan blockchain privat. Hal ini bertujuan agar pencatatan data lingkungan dapat dilakukan dengan aman, transparan, dan tahan manipulasi. Pengguna dapat mengakses data ini melalui antarmuka Web3 yang berupa aplikasi desentralisasi (DApp) yang dirancang untuk menampilkan histori pencatatan lingkungan dari sisi blockchain.

Jalur kedua adalah penyimpanan data ke dalam basis data InfluxDB yang dirancang untuk mengelola data time-series dengan performa tinggi. InfluxDB menerima data dari

server RUST dan menyimpannya berdasarkan urutan waktu sehingga cocok digunakan untuk keperluan analisis tren dan visualisasi jangka panjang. Untuk menunjang visualisasi ini, data dari InfluxDB ditampilkan melalui dashboard Grafana dan juga melalui aplikasi desktop berbasis Python dengan antarmuka PyQt.

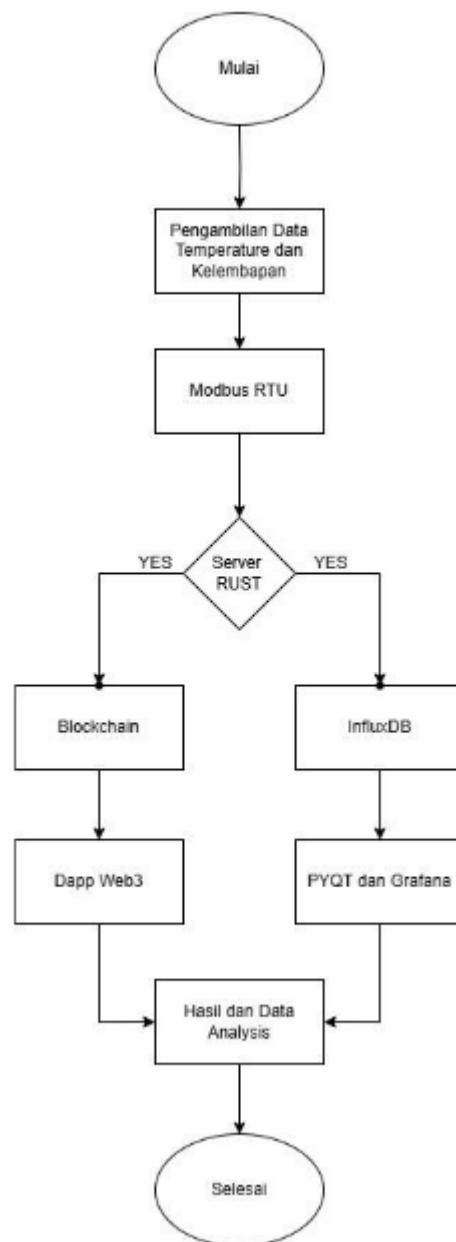
Alur keseluruhan dari pengambilan data dapat dijabarkan sebagai berikut:

1. **Mulai** → Sistem melakukan aktivasi dan inisialisasi sensor pengukur suhu dan kelembapan.
2. **Pengambilan Data Suhu & Kelembapan** → Sensor secara berkala membaca nilai suhu dan kelembapan di dalam greenhouse.
3. **Modbus RTU** → Protokol komunikasi Modbus RTU mentransmisikan data dari sensor ke pusat server.
4. **Server RUST** → Server memproses dan mengarahkan data ke dua cabang utama.
5. **Jalur 1: Blockchain** → **DApp Web3** → Menyimpan data yang telah terenkripsi ke dalam blockchain dan menampilkannya dalam aplikasi web berbasis teknologi Web3.
6. **Jalur 2: InfluxDB** → **PyQt & Grafana** → Menyimpan data dalam time-series database dan memvisualisasikannya untuk analisis tren.
7. **Hasil dan Analisis Data** → Gabungan dari kedua jalur ini menghasilkan pemantauan dan analisis data secara komprehensif, baik dari sisi keamanan (blockchain) maupun dari sisi tren visual (Grafana dan PyQt).

Dengan pengambilan data melalui skema ini, sistem mampu mengakomodasi kebutuhan akan pencatatan data lingkungan yang tidak hanya akurat dan real-time, tetapi juga aman, terbuka untuk audit, dan dapat divisualisasikan dengan baik. Proses pengambilan data dilakukan secara otomatis dengan interval waktu yang dapat dikonfigurasi sesuai kebutuhan operasional greenhouse.

Data yang diperoleh dari pengukuran suhu dan kelembapan juga diproses lebih lanjut untuk menghitung konsumsi energi listrik apabila kondisi lingkungan berada di luar batas optimal, seperti suhu di atas 27°C atau kelembapan di bawah 50%. Dengan fitur ini, pengguna dapat mengestimasi beban listrik tambahan akibat penggunaan kipas, humidifier, atau dehumidifier dan menghitung biaya listrik berdasarkan tarif dasar nasional.

Keseluruhan proses ini membentuk fondasi sistem monitoring pintar untuk greenhouse tanaman cabai yang tidak hanya efektif dalam pengambilan data, tetapi juga efisien dalam membantu pengambilan keputusan berbasis informasi aktual dan terpercaya. Proses pengambilan data ini telah divisualisasikan secara sistematis melalui *Diagram Alir Sistem Pengambilan Data* seperti yang ditunjukkan pada Gambar 3.1 berikut ini.



**Gambar 3.1** Diagram Alir Pengambilan Data dan Penyimpanan Informasi Lingkungan

Diagram ini menegaskan bahwa sistem dirancang untuk mendistribusikan data lingkungan ke dua jalur utama yang bekerja secara paralel namun saling melengkapi satu untuk keamanan dan integritas data (blockchain) dan satu lagi untuk analisis tren yang kaya secara visual (InfluxDB dan Grafana). Untuk greenhouse tanaman cabai yang tidak hanya efektif dalam pengambilan data, tetapi juga efisien dalam membantu pengambilan keputusan berbasis informasi aktual dan terpercaya.

### 3.3 Teknik Analisis Data

Analisis data dilakukan terhadap dua jenis informasi utama, yaitu:

1. Parameter lingkungan: suhu dan kelembapan, dianalisis terhadap rentang ideal pertumbuhan cabai ( $20\text{--}27^{\circ}\text{C}$  dan  $50\text{--}85\%$ ).
2. Konsumsi energi dan estimasi biaya: dihitung berdasarkan penyimpangan dari rentang ideal.

Proses analisis melibatkan:

- Visualisasi grafik fluktuasi suhu dan kelembapan.

- Pemantauan tren penggunaan energi tambahan akibat kondisi lingkungan yang tidak ideal.
- Analisis estimasi biaya listrik sebagai hasil dari konsumsi energi per titik pembacaan.

Data yang dianalisis memberikan pemahaman menyeluruh tentang bagaimana kondisi lingkungan berdampak langsung pada penggunaan energi, dan seberapa besar biaya listrik yang dikeluarkan untuk menjaga iklim optimal dalam greenhouse cabai.

### **3.4 Perhitungan Konsumsi Energi dan Estimasi Biaya**

Salah satu fitur penting dari sistem yang dikembangkan dalam penelitian ini adalah kemampuan untuk menghitung estimasi biaya listrik secara otomatis berdasarkan data sensor. Estimasi ini memperhitungkan energi tambahan yang dibutuhkan saat kondisi lingkungan berada di luar rentang ideal untuk pertumbuhan cabai.

Adapun asumsi dan rumus perhitungannya adalah sebagai berikut:

1. Tarif dasar listrik: Rp 1.444,70/kWh.
2. Suhu ideal: 20°C – 27°C.
  - a. Jika suhu > 27°C, maka konsumsi bertambah 0,5 kWh per °C untuk pendingin.
  - b. Jika suhu < 20°C, maka konsumsi bertambah 0,3 kWh per °C untuk pemanas.
3. Kelembapan ideal: 50% – 85%.
 

85%, konsumsi bertambah 0,75 kWh untuk dehumidifier.

Formula: Total Energi (kWh) = 1 + Tambahan Suhu + Tambahan Kelembapan  
 Total Biaya (Rp) = Energi Total × Rp 1.444,70

Contoh kasus: Suhu: 30°C → Tambahan:  $(30 - 27) \times 0,5 = 1,5$  kWh  
 Kelembapan: 45% → Tambahan: 0,75 kWh Energi Total =  $1 + 1,5 + 0,75 = 3,25$  kWh Biaya =  $3,25 \times 1.444,70 = \text{Rp } 4.695$

Perhitungan ini diimplementasikan dalam fungsi JavaScript di sisi klien (client-side), dan hasilnya ditampilkan pada tabel di dashboard web. Dengan adanya fitur ini, sistem dapat memberikan informasi biaya operasional secara transparan, membantu petani dalam pengambilan keputusan terkait efisiensi energi di greenhouse cabai.

### **3.5 Langkah-langkah Pembuatan**

#### **a. Pembuatan File Sensor**

```

# This file is automatically generated by Cargo.
# It is not intended for manual editing.

version = 4

[[package]]
name = "addr2line"
version = "0.24.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "dfba277e56a376000877998da837666b4427aad538e3928d44ebffe4f89a1c1"
dependencies = [
    "gimli",
]

[[package]]
name = "adler2"
version = "2.0.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "512761e0bb2578dd7380c6baaa0f4ce03e84f95e960231d1dec8bf4d7d6e2627"

[[package]]
name = "android-tzdata"
version = "0.1.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "e999941b234f3131b00bc13c22d06e8c5ff726d1b6318ac7eb276997bbb4fef0"

```

**Gambar 3.2 Struktur File Sensor**

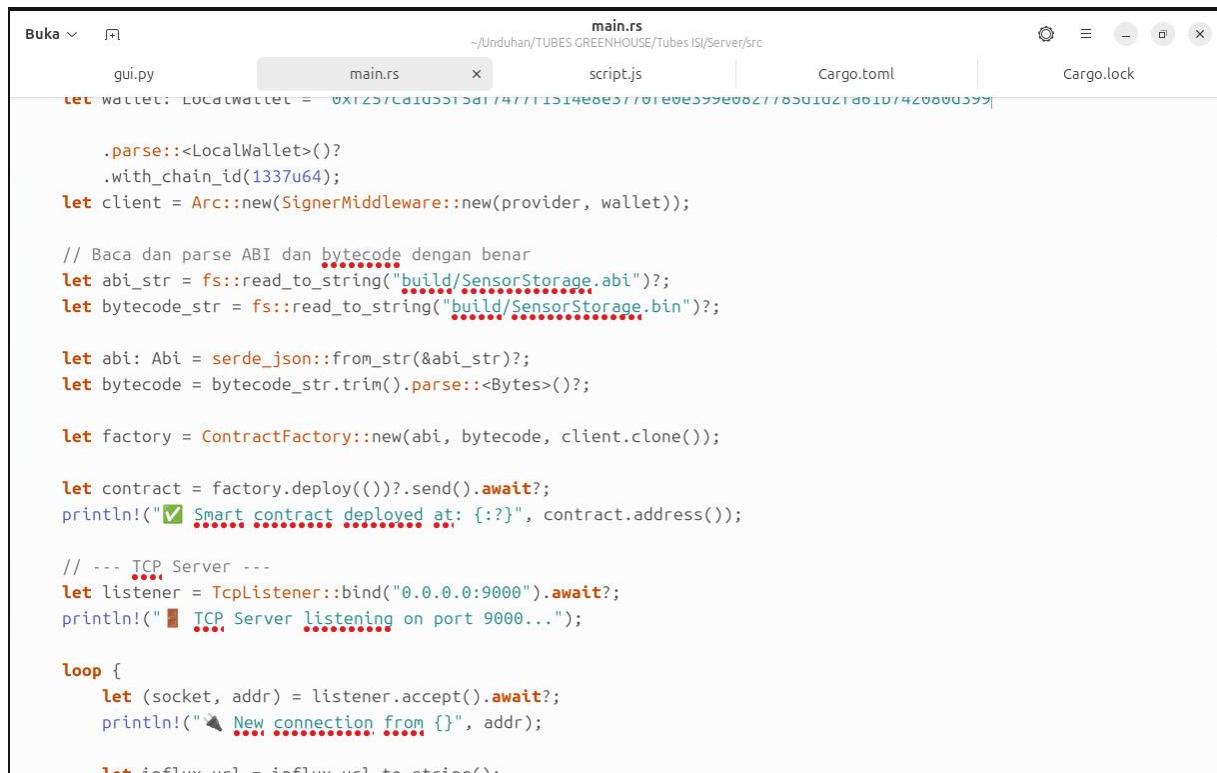
File sensor berperan sebagai komponen pengumpul data utama dalam sistem monitoring greenhouse cabai. Implementasi dilakukan menggunakan bahasa pemrograman Rust karena keunggulannya dalam performa dan manajemen memori. Proses dimulai dengan inisialisasi proyek Rust menggunakan Cargo, dan mendeklarasikan dependensi seperti modbus-rtu dalam Cargo.toml.

Pada src/main.rs, struktur utama program ditulis dalam fungsi main() yang mencakup:

1. Inisialisasi koneksi serial Modbus RTU ke sensor suhu dan kelembapan.
2. Penanganan error untuk koneksi perangkat.
3. Pembuatan koneksi TCP (TcpStream) ke server RUST.
4. Loop tak hingga yang menjalankan:
  - a. Pengambilan data Modbus.
  - b. Parsing ke format float.
  - c. Pembentukan JSON.
  - d. Pengiriman melalui stream TCP.
  - e. Logging hasil pembacaan di terminal.

Delay disisipkan dengan Duration untuk menghindari pembacaan berlebih. Hasil eksekusi melalui cargo run menampilkan output kontinyu sebagai bukti proses berjalan normal. Komponen ini menjadi fondasi penting untuk seluruh sistem monitoring berbasis Web3.

## b. Pembuatan TCP Server



```
Buka ▾  main.rs
~/Unduhan/TUBES GREENHOUSE/Tubes ISI/Server/src
gui.py           main.rs   x   script.js   Cargo.toml   Cargo.lock

let wallet: LocalWallet = 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);
}

//+ influx ucl = influx ucl.to_string()?
```

Gambar 3.3 Struktur File TCP Server

Komponen server bertugas menerima data dari file sensor dan mendistribusikannya ke dua jalur: blockchain dan InfluxDB. Server ditulis dalam bahasa Rust dan dimulai dengan deklarasi dependensi di Cargo.toml seperti serde, tokio, dan std::net.

Di src/main.rs, fungsi main() membangun listener TCP pada port tertentu, menerima stream data masuk, melakukan:

1. Deserialisasi JSON dari sensor.
2. Penyimpanan data ke smart contract Ethereum berbasis event log.
3. Pengiriman data ke InfluxDB untuk kebutuhan visualisasi.

Proses ini berjalan secara asinkron dan kontinu, dilengkapi mekanisme logging dan error-handling, memastikan distribusi data berjalan andal dan akurat sebagai tulang punggung komunikasi data sistem greenhouse cabai.

### c. Pembuatan Antarmuka Web3 (DApp)



```
const contractAddress = "0x6ca0d865e53860e72f783fbf351643127a48963";
const abiPath = "abi/SensorStorage_abi";
const rpcURL = "http://127.0.0.1:8545";

let chart;

async function loadSensorData() {
    try {
        const abiRes = await fetch(abiPath);
        const abi = await abiRes.json();
        console.log("ABI loaded successfully");

        const provider = new ethers.JsonRpcProvider(rpcURL);
        const contract = new ethers.Contract(contractAddress, abi, provider);

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

        console.log("Events received:", events);

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

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

        for (const event of events) {
            const timestamp = event.timestamp.toNumber();
            const temp = event.data[0];
            const hum = event.data[1];

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

        chart = new Chart(document.getElementById("chart"), {
            type: "line",
            data: {
                labels: labels,
                datasets: [
                    {
                        label: "Temperature (°C)",
                        data: temps,
                        fill: false,
                        lineTension: 0
                    },
                    {
                        label: "Humidity (%)",
                        data: hums,
                        fill: false,
                        lineTension: 0
                    }
                ]
            },
            options: {
                scales: {
                    x: {
                        type: "time",
                        time: {
                            unit: "second"
                        }
                    }
                }
            }
        });
    } catch (error) {
        console.error(error);
    }
}

document.getElementById("load-data").addEventListener("click", loadSensorData);
```

Gambar 3.4 Struktur File Web3

Frontend sistem dikembangkan menggunakan pendekatan SPA dengan HTML, CSS, dan JavaScript. Komponen utamanya adalah:

1. index.html: Struktur halaman dasar.
2. style.css: Tampilan visual bertema hijau gelap yang menggambarkan suasana greenhouse.
3. script.js: Logika interaksi Web3 dengan blockchain melalui Ethers.js.

Ketika pengguna mengakses aplikasi dan menekan tombol “Load Sensor Data”, browser akan:

1. Terhubung dengan MetaMask dan jaringan blockchain.
2. Memanggil event pada smart contract menggunakan ABI dan alamat kontrak.
3. Mengambil data historis suhu dan kelembapan.
4. Menyajikan data dalam bentuk tabel dan grafik (Chart.js).

Web3 DApp ini memberikan transparansi total atas histori pencatatan lingkungan greenhouse cabai dan menjadikan pengguna sebagai verifier langsung dari data pada blockchain.

Dengan pengambilan data melalui skema ini, sistem mampu mengakomodasi kebutuhan akan pencatatan data lingkungan yang tidak hanya akurat dan real-time, tetapi juga aman, terbuka untuk audit, dan dapat divisualisasikan dengan baik. mengakomodasi kebutuhan akan pencatatan data lingkungan yang tidak hanya akurat dan real-time, tetapi juga aman, terbuka untuk audit, dan dapat divisualisasikan dengan baik. Proses pengambilan data dilakukan secara otomatis dengan interval waktu yang dapat dikonfigurasi sesuai kebutuhan operasional greenhouse.

Data yang diperoleh dari pengukuran suhu dan kelembapan juga diproses lebih lanjut untuk menghitung konsumsi energi listrik apabila kondisi lingkungan berada di luar batas optimal, seperti suhu di atas 27°C atau kelembapan di bawah 50%.

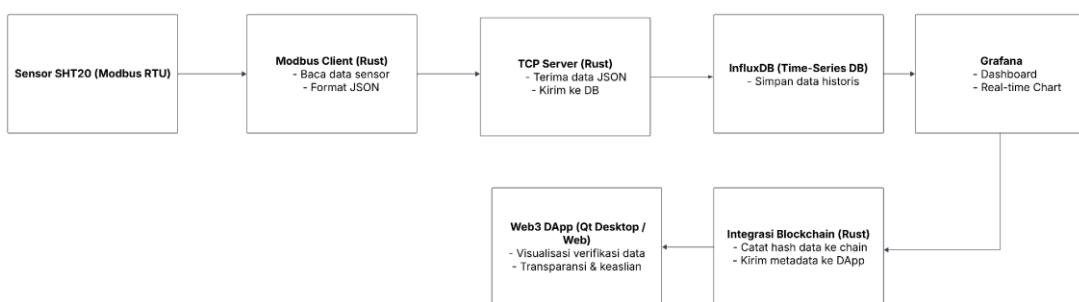
Dengan fitur ini, pengguna dapat mengestimasi beban listrik tambahan akibat penggunaan kipas, humidifier, atau dehumidifier dan menghitung biaya listrik berdasarkan tarif dasar nasional.

### 3.6 Arsitektur Sistem

Sistem monitoring ini dibangun untuk mengukur suhu dan kelembaban secara *real-time* di lingkungan greenhouse cabai. Alur utama dimulai dari pengambilan data melalui sensor SHT20 yang berkomunikasi menggunakan protokol Modbus RTU. Data dari sensor dibaca oleh Modbus Client yang dikembangkan dengan bahasa pemrograman Rust. Client ini mengirimkan data yang telah diformat (JSON) melalui jaringan ke TCP Server, yang juga dibangun dengan Rust.

TCP Server berfungsi menerima data sensor dan menyimpannya ke dalam InfluxDB, sebuah database *time-series* yang dirancang khusus untuk data pengukuran berdasarkan waktu. Selanjutnya, data yang tersimpan divisualisasikan menggunakan Grafana, sebuah platform *dashboard open-source*.

Untuk meningkatkan kepercayaan dan transparansi data, sistem ini terintegrasi dengan Blockchain. Setiap data yang masuk akan disimpan ke dalam jaringan blockchain melalui smart contract. Pengguna dapat memverifikasi data menggunakan aplikasi DApp yang dibangun dengan Qt dan Web3.js, menjadikan sistem ini tidak hanya cerdas dan efisien, tetapi juga traceable dan akuntabel dalam konteks rantai pasok pertanian.



**Gambar 3.1** Diagram Blok

Sistem monitoring suhu dan kelembaban ini terdiri dari beberapa komponen utama: sensor, komunikasi data, penyimpanan dan visualisasi, aplikasi desktop, serta integrasi blockchain.

Arsitektur sistem secara umum dibagi menjadi tiga lapisan utama:

1. Perangkat Keras (Hardware Layer): terdiri dari sensor suhu dan kelembaban (SHT20) yang menggunakan komunikasi Modbus RTU.
2. Lapisan Middleware: berperan dalam pengumpulan data, konversi, pengiriman melalui protokol TCP ke server.

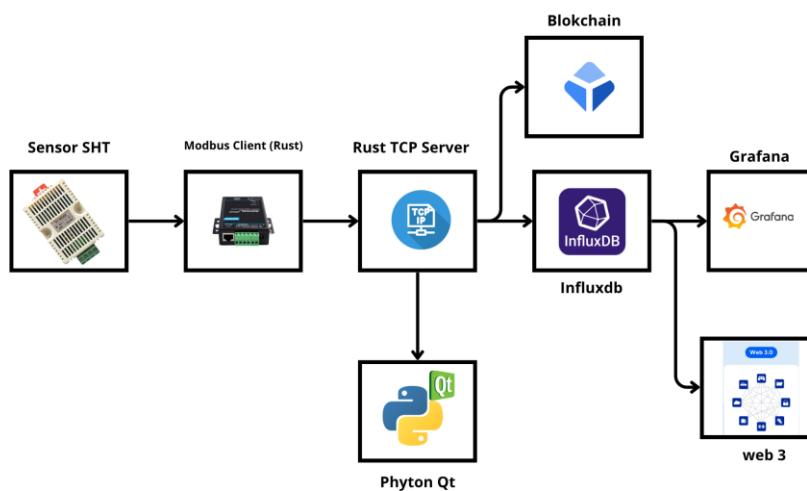
3. Lapisan Backend dan Frontend: data disimpan di InfluxDB, divisualisasikan menggunakan Grafana, serta ditampilkan pada aplikasi desktop berbasis Qt dan ditulis ke blockchain untuk transparansi.

### 3.6 Perancangan Sistem Monitoring

Sistem ini dirancang untuk mengukur dan memantau suhu serta kelembaban di dalam greenhouse secara real-time. Parameter lingkungan sangat penting untuk tanaman cabai yang sensitif terhadap perubahan iklim mikro.

Sensor SHT20 digunakan karena mendukung komunikasi Modbus RTU dan memiliki akurasi tinggi. Data suhu dan kelembaban dibaca secara periodik oleh mikrokontroler, kemudian dikirim melalui jaringan ke server.

Desain Arsitektur Sistem



**Gambar 1.** Diagram blok arsitektur sistem

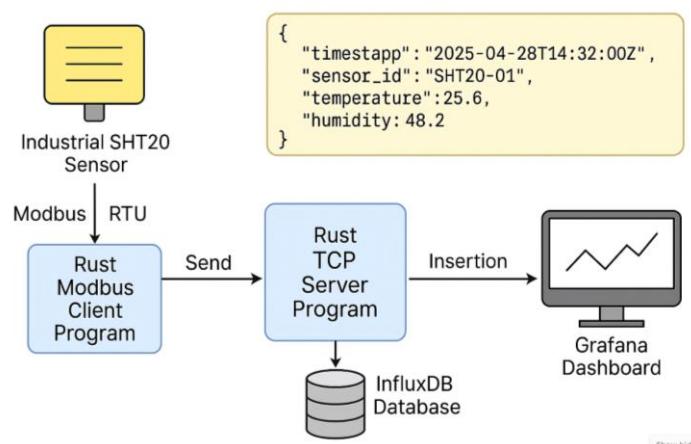


Diagram arsitektur sistem ini menunjukkan alur data dari sensor suhu dan kelembaban SHT20 hingga ditampilkan dalam bentuk visualisasi di dashboard Grafana. Sensor SHT20 yang digunakan merupakan sensor industri yang mendukung komunikasi menggunakan protokol Modbus RTU. Data dari sensor ini dibaca oleh sebuah program Modbus Client yang dibuat menggunakan bahasa pemrograman Rust. Program ini bertugas untuk mengakses data suhu dan kelembaban dari sensor dan mengemasnya dalam format JSON yang berisi informasi waktu (*timestamp*), ID sensor, suhu, dan kelembaban.

Selanjutnya, data JSON ini dikirim ke sebuah program TCP Server yang juga dikembangkan dengan Rust. TCP Server ini menerima data dari Modbus Client melalui jaringan dan kemudian menyimpannya ke dalam InfluxDB, yaitu sebuah database time-series yang dirancang untuk menyimpan data pengukuran sensor secara efisien berdasarkan waktu. Penyimpanan dalam InfluxDB memungkinkan sistem untuk menyimpan riwayat data secara terus menerus dan terstruktur, sehingga dapat dianalisis maupun divisualisasikan dengan mudah.

Akhirnya, data yang telah tersimpan dalam InfluxDB ditampilkan ke pengguna melalui Grafana Dashboard. Grafana adalah platform open-source yang sangat populer untuk visualisasi data time-series. Melalui Grafana, pengguna dapat melihat grafik suhu dan kelembaban secara real-time maupun historis, yang sangat berguna untuk kebutuhan pemantauan kondisi lingkungan atau sistem otomatisasi. Dengan arsitektur ini, sistem menjadi terintegrasi, efisien, dan mudah dikembangkan untuk kebutuhan pemantauan lainnya.

#### Penerapan V-Model dalam Proyek Sistem Monitoring Greenhouse

V-Model adalah salah satu model pengembangan sistem yang berorientasi pada verifikasi dan validasi di setiap tahapan pengembangan. Model ini berbentuk huruf “V” untuk menggambarkan hubungan antara tahapan perancangan (development) di sisi kiri dan pengujian (testing/validation) di sisi kanan.

Tahapan Pengembangan	Deskripsi & Implementasi dalam Proyek	Tahapan Pengujian	Deskripsi Pengujian
Business Requirements Definition	Menentukan kebutuhan pengguna (misalnya: petani ingin memantau suhu & kelembaban secara real-time dan transparan)	Acceptance Testing	Menguji apakah sistem memenuhi kebutuhan pengguna akhir secara keseluruhan
System Design	Mendesain keseluruhan sistem . (sensor → mikrokontroler → komunikasi → penyimpanan → visualisasi → blockchain)	System Testing	Menguji fungsi sistem secara menyeluruh: konektivitas sensor, database, blockchain

Tahapan Pengembangan	Deskripsi & Implementasi dalam Proyek	Tahapan Pengujian	Deskripsi Pengujian
Architectural Design	Mendesain arsitektur sistem: hardware, protokol komunikasi (Modbus RTU), arsitektur IoT, dan integrasi Qt + blockchain	Integration Testing	Menguji integrasi antar modul: sensor–mikrokontroler–server–Qt–blockchain
Module Design	Mendesain modul: pembacaan sensor, pengiriman data TCP, simpan ke InfluxDB, GUI Qt, pencatatan smart contract	Unit Testing	Menguji setiap modul secara terpisah: pembacaan sensor, koneksi TCP, penyimpanan DB, dsb
Coding / Implementation	Implementasi kode (C/C++ untuk mikrokontroler, Python untuk server, Solidity untuk blockchain, Qt untuk GUI)	—	—

### 3.7 Komunikasi Data

Data dari sensor dikomunikasikan menggunakan protokol Modbus RTU melalui antarmuka RS-485. Protokol ini dipilih karena kestabilannya dalam lingkungan industri dan jarak jangkauannya yang baik. Mikrokontroler membaca register sensor SHT20, memprosesnya, lalu mengirimkannya melalui protokol TCP/IP ke server lokal. Protokol TCP menjamin pengiriman data yang sesuai.

Sistem komunikasi dibangun atas dua lapisan utama:

#### 1. Modbus RTU

- Digunakan pada layer sensor-mikrokontroler.
- Protokol komunikasi serial berbasis master-slave dengan keunggulan: efisien, deterministik, dan tahan gangguan lingkungan.

#### 2. TCP/IP

- Digunakan antara mikrokontroler dan server.
- Menjamin transmisi data yang **reliable (handshake, acknowledgement)**.

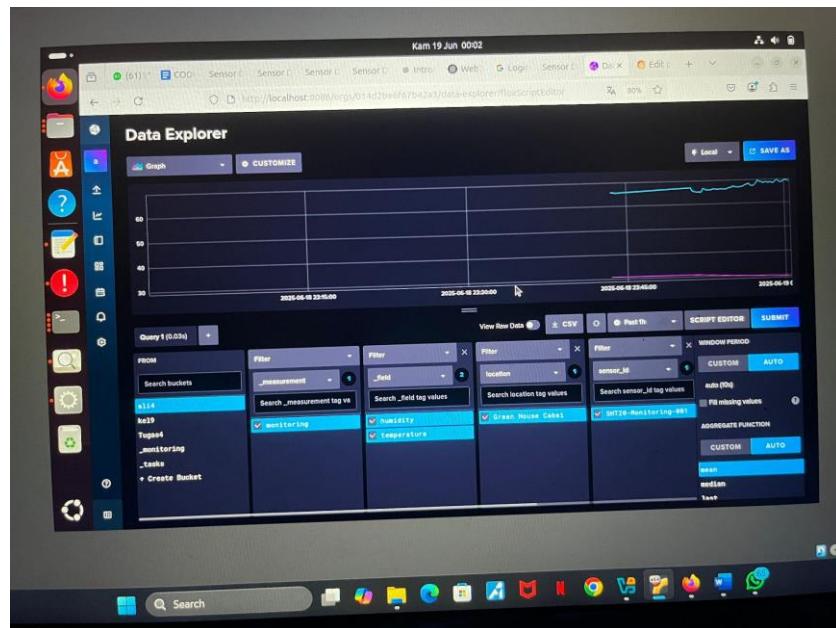
### 3.8 Penyimpanan dan Visualisasi Data

Setelah data diterima oleh server, sistem menyimpannya dalam InfluxDB, sebuah database time-series yang efisien untuk data yang terus-menerus berubah seperti suhu dan kelembaban.

Grafana digunakan sebagai dashboard visualisasi. Pengguna dapat melihat grafik suhu dan kelembaban secara real-time, melakukan analisis dan mengatur notifikasi jika parameter keluar dari batas ideal (misalnya suhu terlalu tinggi).

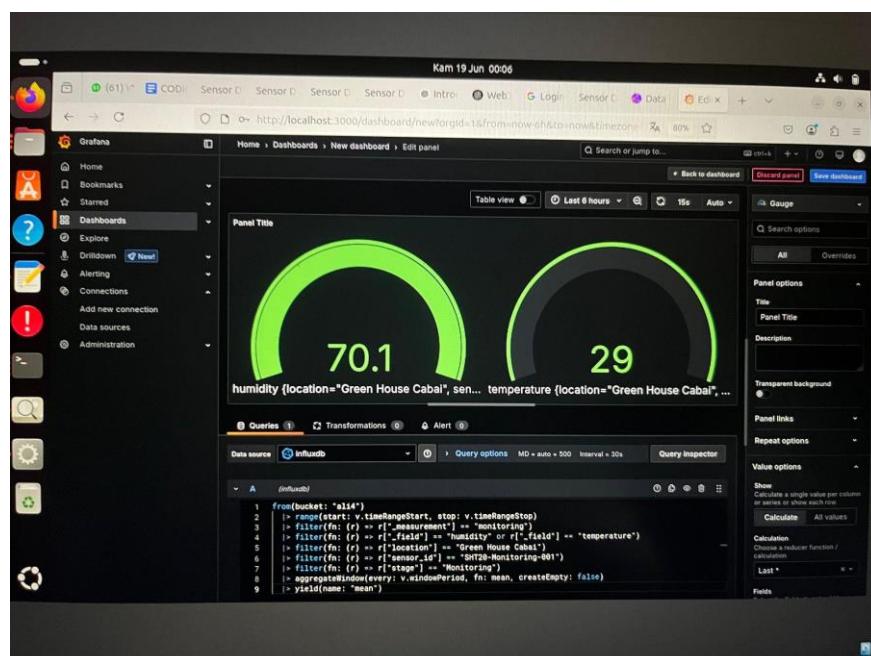
Data yang diterima akan:

- **Disimpan ke InfluxDB dengan format:**



Gambar 3.1 Tampilan influxDB

- **Divisualisasikan di Grafana:**



Gambar 3.2 Tampilan influxDB

### **3.9 Desain dan pengembangan aplikasi desktop Qt**

Untuk kebutuhan lokal (misalnya di ruang kontrol greenhouse), dikembangkan aplikasi desktop berbasis Qt (C++ atau Python Qt). Aplikasi ini terhubung langsung ke InfluxDB atau menerima data dari server untuk menampilkan informasi suhu dan kelembaban secara langsung. Fitur aplikasi:

1. Monitoring suhu dan kelembaban secara lokal
2. Riwayat data
3. Tampilan sederhana dan responsif

GUI dirancang sederhana dan user-friendly, cocok untuk petani atau operator non-teknis.

### **3.10 Integrasi Blockchain dan Web3**

Untuk mendukung transparansi dan traceability, sistem ini terintegrasi dengan blockchain. Data suhu dan kelembaban dicatat ke dalam smart contract secara berkala.

Langkah-langkah:

1. Smart contract dikembangkan menggunakan Solidity
2. Akses dilakukan melalui Web3.js atau Ethers.js
3. DApp sederhana dikembangkan untuk membaca data lingkungan dari blockchain dan menampilkannya ke pengguna akhir (misalnya petani atau pengelola distribusi)

Tujuan utamanya adalah agar data lingkungan greenhouse cabai dapat diaudit publik, sehingga kualitas produksi bisa ditelusuri dengan baik.

### **3.11 Implementasi dan Kode Program**

Setiap bagian dari sistem memiliki kode program tersendiri:

1. Mikrokontroler:

Program untuk membaca register Modbus RTU dari sensor SHT20

Mengonversi data dan mengirim ke TCP Server

2. Server:

TCP Server untuk menerima data dari mikrokontroler

Menulis ke InfluxDB

3. Aplikasi Qt:

Mengambil data dari database

Menampilkan melalui antarmuka GUI

4. Smart Contract:

Kontrak pintar untuk menyimpan hash atau nilai pengukuran

Fungsi untuk validasi dan pencatatan waktu

Sistem diuji coba di lingkungan greenhouse untuk memantau kondisi suhu dan kelembaban secara langsung. Pengujian dilakukan dengan menempatkan sensor di

beberapa titik dalam greenhouse dan membandingkan hasilnya dengan alat ukur konvensional. Hasil pengujian menunjukkan bahwa sistem mampu merespon perubahan suhu dan kelembaban dengan cepat dan akurat. Data yang ditampilkan di Grafana sesuai dengan nilai yang terbaca di lapangan, dan hash data yang dicatat di blockchain dapat diverifikasi melalui explorer. Implementasi ini membuktikan bahwa sistem dapat digunakan untuk pemantauan kondisi lingkungan secara kontinu dan memberikan informasi yang dibutuhkan petani untuk pengambilan keputusan.

## BAB IV

### HASIL DAN PEMBAHASAN

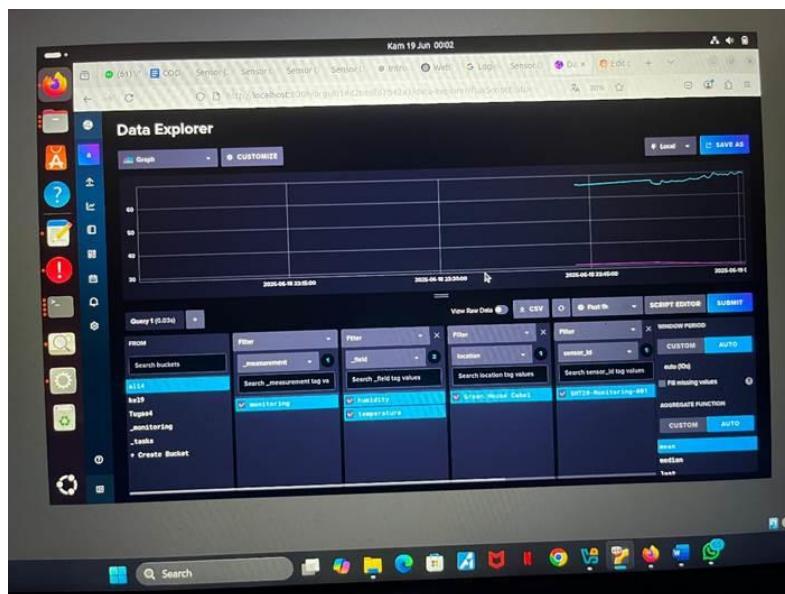
#### 4.1 Analisis dan Evaluasi Sistem

Sistem diuji dalam beberapa skenario:

- Stabilitas komunikasi: dilakukan uji keandalan data yang dikirim dari sensor ke TCP Server.
- Kecepatan akses: respon waktu dari pengukuran hingga visualisasi pada dashboard.
- Akurasi sensor: pengujian menunjukkan error pengukuran  $<\pm 0.3^{\circ}\text{C}$  untuk suhu dan  $<\pm 2\%$  RH untuk kelembaban.
- Keamanan dan transparansi data: blockchain berhasil mencatat data secara immutable, sehingga tidak bisa diubah atau dihapus.

Evaluasi juga melibatkan pengujian beban (load testing) pada InfluxDB dan latency pengambilan data dari blockchain.

#### Penjelasan

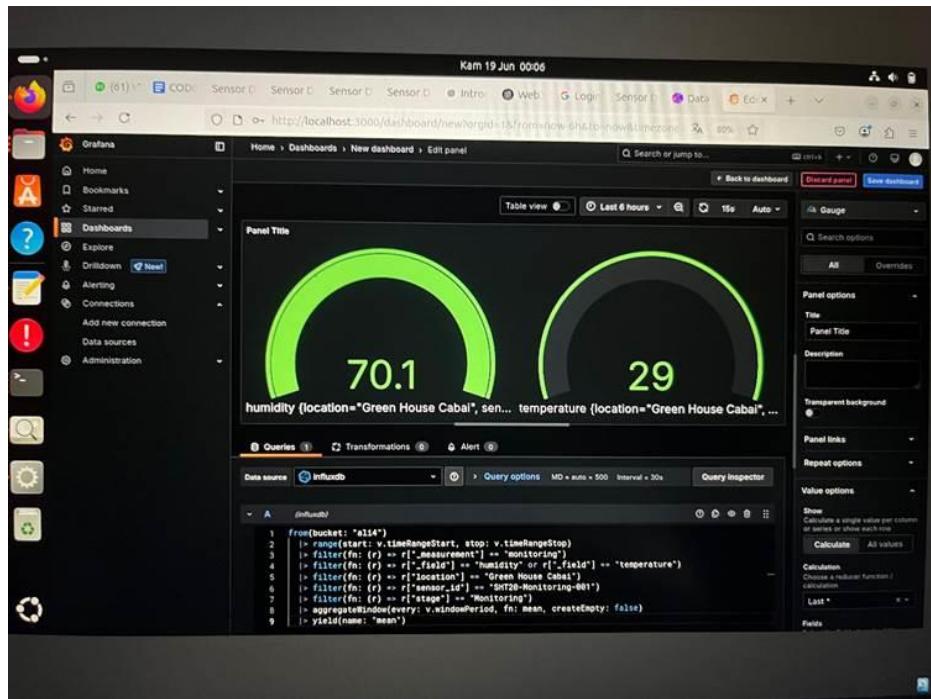


Gambar 3.3 InfluxDB

Gambar yang ditampilkan merupakan antarmuka pengguna (User Interface) Data Explorer pada InfluxDB, sebuah platform basis data time-series yang digunakan untuk memantau dan menganalisis data sensor secara real-time. Dalam tampilan tersebut terlihat grafik yang menunjukkan perubahan dua parameter lingkungan, yaitu suhu (temperature) dan kelembaban (humidity). Data tersebut diambil dari sensor dengan ID “SHT20-Monitoring-001” yang dipasang di lokasi bertanda “Green\House\Cabai”, mengindikasikan bahwa sistem ini kemungkinan digunakan untuk pemantauan kondisi rumah kaca tanaman cabai.

Di bagian bawah grafik terdapat beberapa kolom pengaturan query, seperti pemilihan bucket data (dalam hal ini bucket bernama “\\_monitoring”), pengaturan field (suhu dan kelembaban), serta filter tag lokasi dan ID sensor. Tersedia juga fitur

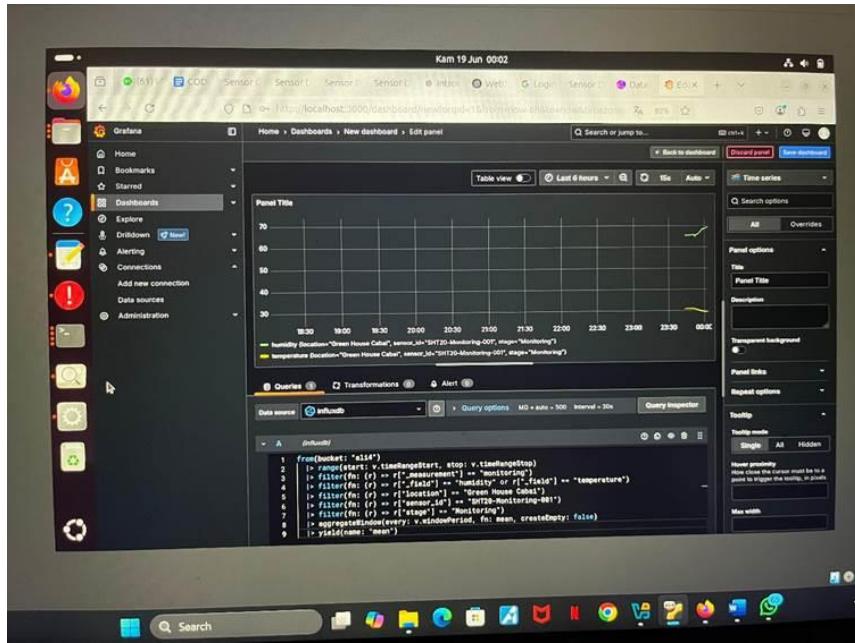
tambahan seperti tampilan data mentah (raw data), ekspor data ke format CSV, dan editor skrip untuk query manual menggunakan bahasa Flux. Fitur lain seperti pengaturan window period dan agregasi data (mean, median, last) memungkinkan pengguna menyesuaikan tampilan data agar lebih mudah dianalisis.



Gambar 3.4 Tampilan grafana

Gambar tersebut menampilkan antarmuka pengguna dari **Grafana**, platform visualisasi data yang digunakan untuk menampilkan data monitoring secara real-time dalam bentuk dashboard. Dalam tampilan ini, terdapat dua panel gauge yang menunjukkan nilai parameter lingkungan dari sebuah sistem monitoring. Panel di sebelah kiri menampilkan nilai **kelembaban (humidity)** sebesar **70.1%**, sedangkan panel di sebelah kanan menampilkan nilai **suhu (temperature)** sebesar **29°C**. Kedua data tersebut berasal dari lokasi bertanda **“Green House Cabai”** dan menggunakan sensor dengan ID **“SHT20-Monitoring-001”**, yang berarti sistem ini digunakan untuk pemantauan suhu dan kelembaban dalam rumah kaca tanaman cabai.

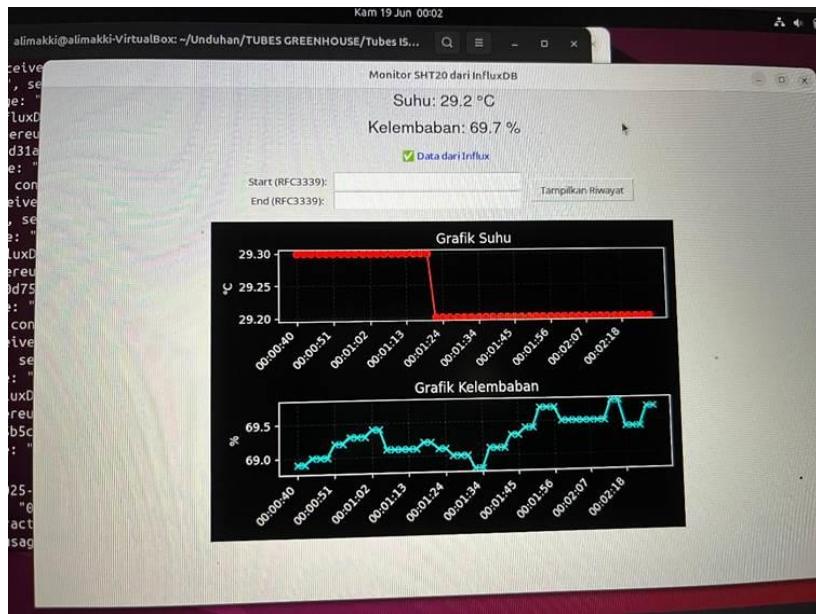
Di bagian bawah tampilan, terlihat **query editor** yang menampilkan skrip bahasa **Flux** yang digunakan untuk mengambil dan mengolah data dari database **InfluxDB**. Query tersebut mengambil data dari bucket bernama **“ali4”**, menyaring berdasarkan measurement **“monitoring”**, field **“humidity”** dan **“temperature”**, serta tag lokasi dan sensor ID yang sesuai. Data kemudian diolah dengan fungsi agregasi mean untuk menampilkan rata-rata nilai terakhir. Tampilan Grafana ini disusun dalam mode panel Gauge, yang memberikan tampilan visual intuitif terhadap nilai-nilai parameter lingkungan, sehingga memudahkan pengguna dalam melakukan pemantauan kondisi secara langsung dan efisien. Gambar ini menunjukkan integrasi antara InfluxDB sebagai penyimpanan data dan Grafana sebagai alat visualisasi yang sering digunakan dalam sistem IoT dan pemantauan lingkungan.



**Gambar 3.5** Tampilan grafik grafana

Gambar ini menampilkan antarmuka **Grafana** yang sedang digunakan untuk membuat atau mengedit panel visualisasi dalam bentuk **time series** untuk memantau data sensor suhu dan kelembaban secara waktu nyata. Terlihat grafik pada bagian tengah layar menunjukkan dua garis tren yang mewakili parameter **humidity** (kelembaban) dan **temperature** (suhu) dari lokasi bertanda “**Green House Cabai**”, menggunakan sensor dengan ID “**SHT20-Monitoring-001**” dan data yang berasal dari bucket “**a1i4**”. Waktu pada sumbu horizontal menunjukkan pengambilan data selama 6 jam terakhir, dan sumbu vertikal menunjukkan nilai dari masing-masing parameter.

Pada bagian bawah layar, terlihat editor query menggunakan bahasa **Flux** dari InfluxDB, yang digunakan untuk menarik data dari bucket dan melakukan proses filter berdasarkan field (“humidity” dan “temperature”), lokasi (“Green House Cabai”), serta sensor ID dan stage (“Monitoring”). Data yang ditarik kemudian diolah menggunakan fungsi aggregateWindow untuk menghitung rata-rata (mean) dari nilai-nilai yang dikumpulkan dalam interval waktu tertentu. Visualisasi ini disesuaikan dalam bentuk garis waktu agar pengguna dapat menganalisis perubahan suhu dan kelembaban secara lebih rinci dan kronologis. Tampilan ini merupakan bagian penting dalam sistem monitoring lingkungan berbasis IoT karena membantu dalam pengambilan keputusan cepat terkait kondisi lingkungan dalam rumah kaca secara real-time.



**Gambar 3.5** Tampilan Qt

Gambar ini menunjukkan antarmuka visual dari sistem pemantauan suhu dan kelembaban yang menampilkan data dalam bentuk grafik berbasis waktu nyata. Di bagian atas ditampilkan informasi suhu sebesar **29.2°C** dan kelembaban sebesar **69.7%**, yang menunjukkan pembacaan terkini dari sensor. Sistem ini juga memiliki fitur untuk menampilkan riwayat data berdasarkan rentang waktu yang dapat diatur oleh pengguna melalui format waktu **RFC3339**, dengan tombol "Tampilkan Riwayat" untuk mengeksekusi permintaan data historis dari basis data InfluxDB, seperti yang ditandai dengan checkbox "**Data dari Influx**".

Dua grafik utama ditampilkan di bagian bawah layar. Grafik pertama, berjudul "**Grafik Suhu**", menunjukkan plot titik-titik data suhu dengan warna merah, yang tampak relatif konstan pada nilai  $29.30^{\circ}\text{C}$  sebelum mengalami penurunan ke sekitar  $29.20^{\circ}\text{C}$ . Grafik kedua, "**Grafik Kelembaban**", menggunakan warna biru dan menunjukkan fluktuasi nilai kelembaban dari sekitar 69.0% hingga 69.7%, yang menggambarkan perubahan kelembaban udara dalam waktu singkat. Visualisasi ini membantu pengguna untuk memantau perubahan parameter lingkungan secara cepat dan presisi, sangat berguna untuk aplikasi seperti pemantauan kondisi rumah kaca atau ruang sensitif lainnya.

```

Rab 18 Jun 22:44
Buka ... main.rs
alimakki@alimakki-VirtualBox:~/Unduhan/TUBES GREENHOUSE/Tubes IS...
alimakki@alimakki-VirtualBox:~/Unduhan/TUBES GREENHOUSE/Tubes ISI/Server$ cargo
run
    Compiling sensor_server_blockchain v0.1.0 (/home/alimakki/Unduhan/TUBES GREEN
HOUSE/Tubes ISI/Server)
# [tc] Building [=====] 412/413: sensor_server_blockchain(...)

let abi: Abi = serde_json::from_str(&abi_str)?;
let bytecode = bytecode_str.trim().parse::()?;
let factory = ContractFactory::new(abi, bytecode, client.clone());

```

**Gambar 3.6** Integrasi blockchain

Gambar tersebut menunjukkan tampilan terminal Linux saat pengguna menjalankan perintah `npx hardhat` di dalam direktori proyek bernama `my-blockchain-project`. Hardhat merupakan framework pengembangan Ethereum yang digunakan untuk membangun, menguji, dan men-deploy smart contract. Versi Hardhat yang digunakan adalah v2.24.1. Dalam tampilan tersebut, pengguna sedang berada dalam proses inisialisasi proyek dan memilih untuk membuat proyek berbasis JavaScript. Lokasi root proyek ditentukan di direktori `/home/alimakki/my-blockchain-project`. Antarmuka ini merupakan langkah awal untuk membangun aplikasi blockchain menggunakan ekosistem Ethereum.

Greenhouse Sensor Dashboard							
Load Sensor Data							
Timestamp	Sensor ID	Location	Stage	Temperature (°C)	Humidity (%)	Energy (kWh)	Biaya (Rp)
19/6/2025, 01:37:36	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.30	1.70	Rp 2.456
19/6/2025, 01:37:42	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.90	1.70	Rp 2.456
19/6/2025, 01:37:47	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.40	1.70	Rp 2.456
19/6/2025, 01:37:52	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.19	1.70	Rp 2.456
19/6/2025, 01:37:57	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.30	1.70	Rp 2.456
19/6/2025, 01:38:02	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.30	1.70	Rp 2.456
19/6/2025, 01:38:07	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.60	1.70	Rp 2.456
19/6/2025, 01:38:12	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.50	1.70	Rp 2.456
19/6/2025, 01:38:17	SHT20-Monitoring-001	Green House Cabai	Monitoring	28.40	72.19	1.70	Rp 2.456

**Gambar 3.7** Tampilan web3

Gambar tersebut menampilkan dashboard monitoring sensor berjudul "Greenhouse Sensor Dashboard" yang berfungsi untuk menampilkan data lingkungan secara real-time dari sebuah rumah kaca (greenhouse), khususnya untuk tanaman cabai. Data yang ditampilkan pada dashboard ini meliputi timestamp (waktu

pencatatan), Sensor ID, lokasi, tahap monitoring, suhu (Temperature) dalam °C, kelembaban (Humidity) dalam %, serta perhitungan energi (kWh) dan biaya (Rp).

Seluruh data berasal dari sensor yang sama, yaitu SHT20-Monitoring-001, yang terpasang di lokasi Green House Cabai dengan status tahap Monitoring. Dari data yang ditampilkan, terlihat bahwa suhu di dalam rumah kaca cukup stabil pada angka 28.40°C, sedangkan kelembaban mengalami fluktuasi kecil, berkisar antara 72.19% hingga 72.90%. Konsumsi energi yang tercatat secara konsisten adalah 1.70 kWh, dengan biaya yang sama pada setiap baris yaitu Rp 2.456. Ini menunjukkan bahwa sistem tidak hanya memantau kondisi lingkungan, tetapi juga mengintegrasikan data konsumsi energi dan biaya operasional, yang sangat berguna untuk manajemen efisiensi dalam pengoperasian rumah kaca.

#### 4.2 Perhitungan Konsumsi Energi dan Estimasi Biaya

Pada sistem monitoring greenhouse cabai ini, dilakukan proses perhitungan konsumsi energi listrik berdasarkan data temperatur dan kelembapan yang diperoleh dari sensor DHT22. Data ini kemudian digunakan untuk menghitung estimasi biaya penggunaan listrik dalam operasional rumah kaca, terutama dalam pengendalian suhu dan kelembapan lingkungan.

Perhitungan konsumsi energi didasarkan pada skenario idealisasi pengoperasian perangkat pendingin, pemanas, serta pengontrol kelembapan seperti humidifier dan dehumidifier. Nilai tarif dasar listrik yang digunakan mengacu pada tarif industri PLN, yaitu sebesar Rp 1.444,70 per kWh.

Untuk parameter temperatur, ditetapkan rentang ideal antara 20°C hingga 27°C. Apabila suhu yang terbaca melebihi 27°C, sistem diasumsikan mengaktifkan perangkat pendingin (kipas) dengan tambahan konsumsi sebesar 0,5 kWh per derajat Celsius di atas ambang batas. Sebaliknya, jika suhu di bawah 20°C, diasumsikan sistem mengaktifkan pemanas dengan tambahan konsumsi sebesar 0,3 kWh per derajat di bawah ambang bawah.

Sementara itu, untuk parameter kelembapan, ditetapkan rentang ideal antara 50% hingga 85%. Apabila kelembapan berada di bawah 50%, sistem akan mengaktifkan humidifier dengan konsumsi tambahan sebesar 0,75 kWh. Jika kelembapan berada di atas 85%, sistem mengaktifkan dehumidifier dengan konsumsi tambahan sebesar 0,75 kWh pula.

Rumus umum perhitungan konsumsi energi total (dalam kWh) adalah sebagai berikut:

$$\text{Energi Total (kWh)} = 1 + \text{Energi Tambahan Temperatur} + \text{Energi Tambahan Kelembapan}$$

Energi tambahan ini kemudian dikalikan dengan tarif listrik untuk memperoleh estimasi biaya listrik:

$$\text{Biaya (Rp)} = \text{Energi Total (kWh)} \times 1.444,70$$

Contoh

kasus:

Jika suhu terbaca 30°C dan kelembapan 45%, maka:

- Tambahan temperatur:  $(30 - 27) \times 0,5 = 1,5 \text{ kWh}$

- Tambahan kelembapan: **0,75 kWh** (karena kelembapan < 50%)
- Total konsumsi:  $1 + 1,5 + 0,75 = \mathbf{3,25 \text{ kWh}}$
- Biaya listrik:  $3,25 \times 1.444,70 = \mathbf{\text{Rp } 4.695}$

Perhitungan ini dilakukan secara otomatis oleh sistem dan ditampilkan pada dashboard dalam bentuk tabel dan grafik untuk memudahkan pemantauan energi dan estimasi biaya yang dikeluarkan pada setiap interval pembacaan sensor.

## **BAB V** **KESIMPULAN DAN SARAN**

### **4.1 Kesimpulan**

Berdasarkan hasil perancangan dan implementasi yang telah dilakukan dalam proyek ini, dapat disimpulkan beberapa hal sebagai berikut:

- Sistem monitoring suhu dan kelembaban berbasis sensor SHT20 dengan komunikasi Modbus RTU berhasil dirancang dan diimplementasikan secara fungsional. Sistem ini mampu membaca data suhu dan kelembaban secara berkala dan mengirimkannya ke server melalui komunikasi TCP.
- Penerapan arsitektur client-server menggunakan bahasa pemrograman Rust memungkinkan proses komunikasi dan pengiriman data berjalan efisien dan cepat. Modbus Client membaca data sensor, kemudian mengirimkan data ke TCP Server untuk diproses lebih lanjut.
- InfluxDB terbukti efektif sebagai media penyimpanan data time-series, karena dapat mencatat data suhu dan kelembaban secara historis dengan presisi waktu tinggi dan terstruktur. Data kemudian divisualisasikan melalui dashboard Grafana secara real-time, sehingga memudahkan pemantauan kondisi lingkungan greenhouse.
- Integrasi teknologi blockchain menambahkan nilai transparansi dan akuntabilitas pada sistem. Hash dari setiap data sensor yang tercatat dikirimkan ke jaringan blockchain melalui smart contract, memungkinkan proses verifikasi keaslian data melalui DApp berbasis Qt dan [Web3.js](#).
- Sistem ini telah membuktikan bahwa penggabungan teknologi embedded, database, visualisasi, dan blockchain dapat menghasilkan solusi monitoring yang adaptif, cerdas, dan dapat diandalkan untuk sektor pertanian, khususnya pada lingkungan greenhouse.

### **4.2 Saran**

Untuk pengembangan sistem kedepannya, beberapa saran yang dapat dipertimbangkan antara lain:

- Penambahan jumlah sensor dan node monitoring agar mencakup area greenhouse yang lebih luas dan mampu memberikan analisis mikroklimat secara lebih detail.
- Penggunaan sensor industri dengan kalibrasi resmi dan proteksi IP agar dapat diterapkan pada kondisi lapangan yang lebih ekstrem, seperti paparan sinar matahari langsung, kelembaban tinggi, atau cipratan air.

- Penggunaan mikrokontroler atau edge device (seperti Raspberry Pi/ESP32) sebagai Modbus Client, agar sistem lebih portabel dan tidak tergantung pada komputer.
- Integrasi sistem kontrol otomatis berbasis aktuator, misalnya kipas, pompa, atau irigasi otomatis yang dapat diaktifkan berdasarkan ambang suhu/kelembaban tertentu.
- Penerapan sistem keamanan data yang lebih ketat, seperti enkripsi end-to-end, untuk menjaga privasi dan mencegah manipulasi data sensor sebelum masuk ke blockchain.
- Pengembangan antarmuka DApp berbasis web responsif, agar pengguna dapat mengakses data dan melakukan verifikasi dari berbagai perangkat tanpa perlu instalasi lokal.

## DAFTAR PUSTAKA

- Badan Pusat Statistik. (2024). *Produksi Hortikultura Nasional 2023*. <https://www.bps.go.id>
- Bhutia, K., Khanna, V., Meetei, T., & Bhutia, N. (2018). Effects of climate change on growth and development of chilli. *Agrotechnology*, 7(2), 1-4.
- Ikram, M., Ameer, S., Kulsoom, F., Sher, M., Ahmad, A., Zahid, A., & Chang, Y. (2024). Flexible temperature and humidity sensors of plants for precision agriculture: Current challenges and future roadmap. *Computers and Electronics in Agriculture*, 226, 109449.
- Jiang, L., Zeng, Z., & Liu, Y. (2021, November). Design and implementation of smart home control system based on FreeRTOS and QT graphics terminal. In 2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC) (pp. 579-583). IEEE.
- Li, R. A., Sha, X., & Lin, K. (2014, August). Smart greenhouse: A real-time mobile intelligent monitoring system based on WSN. In 2014 international wireless communications and mobile computing conference (IWCMC) (pp. 1152-1156). IEEE.
- Marchesi, L., Marchesi, M., & Tonelli, R. (2022). Automatic generation of Ethereum-based smart contracts for agri-food traceability system. *IEEE Access*, 10, 50363–50383. <https://doi.org/10.1109/ACCESS.2022.3172847>
- Pranto, T. H., Rahman, M. A., & Alam, M. M. (2021). *Blockchain and smart contract for IoT enabled smart agriculture*. arXiv preprint arXiv:2104.00632. <https://doi.org/10.48550/arXiv.2104.00632>
- Rajora, G. L. (2020). Development of a wireless sensor network for agricultural monitoring for Internet of Things (IoT) (Master's thesis, Universitat Politècnica de Catalunya).
- Reddy, C. K., Manojkumar, R., Rao, B. S., & Jena, P. K. (2024, December). Modbus Communication Based Data Extraction Energy Monitoring System. In 2024 International Conference on Smart Electronics and Communication Systems (ISENSE) (pp. 1-5). IEEE.
- Salah, K., Rehman, M. H. U., Nizamuddin, N., & Al-Fuqaha, A. (2019). *Blockchain-based soybean traceability in agricultural supply chain*. *IEEE Access*, 7, 73295–73305. <https://doi.org/10.1109/ACCESS.2019.2918000>
- Wang, L., Zhang, Y., Liu, Y., & Liu, X. (2021). *Smart contract-based agricultural food supply chain traceability*. *IEEE Access*, 9, 9296–9307. <https://doi.org/10.1109/ACCESS.2021.3050294>

## LAMPIRAN

### Lampiran Codingan

#### 1. Codingan Cargo.lock

```
# This file is automatically @generated by Cargo.  
# It is not intended for manual editing.  
version = 4  
  
[[package]]  
name = "addr2line"  
version = "0.24.2"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum =  
"dfbe277e56a376000877090da837660b4427aad530e3028d44e0bffe4f89a1c1  
dependencies = [  
    "gimli",  
]  
  
[[package]]  
name = "adler2"  
version = "2.0.0"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum =  
"512761e0bb2578dd7380c6baaa0f4ce03e84f95e960231d1dec8bf4d7d6e262  
7"  
  
[[package]]  
name = "android-tzdata"  
version = "0.1.1"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum =  
"e999941b234f3131b00bc13c22d06e8c5ff726d1b6318ac7eb276997bbb4fef0"  
  
[[package]]  
name = "android_system_properties"  
version = "0.1.5"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum =  
"819e7219dbd41043ac279b19830f2efc897156490d7fd6ea916720117ee6631  
1"  
dependencies = [  
    "libc",  
]  
  
[[package]]  
name = "async-trait"  
version = "0.1.88"  
source = "registry+https://github.com/rust-lang/crates.io-index"
```

```

checksum = "e539d3fca749fcee5236ab05e93a52867dd549cc157c8cb7f99595f3cedffdb5"
dependencies = [
    "proc-macro2",
    "quote",
    "syn",
]
[[package]]
name = "autocfg"
version = "1.4.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"ace50bade8e6234aa140d9a2f552bbe1db4d353f69b8217bc503490fc1a9f26"
"
[[package]]
name = "backtrace"
version = "0.3.75"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"6806a6321ec58106fea15becdad98371e28d92ccbc7c8f1b3b6dd724fe8f1002"
dependencies = [
    "addr2line",
    "cfg-if",
    "libc",
    "miniz_oxide",
    "object",
    "rustc-demangle",
    "windows-targets",
]
[[package]]
name = "bitflags"
version = "1.3.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"bef38d45163c2f1dde094a7dfd33ccf595c92905c8f8f4fdc18d06fb1037718a"
[[package]]
name = "bitflags"
version = "2.9.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1b8e56985ec62d17e9c1001dc89c88ecd7dc08e47eba5ec7c29c7b5eeecde96
7"
[[package]]
name = "bumpalo"

```

```
version = "3.17.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1628fb46dfa0b37568d12e5edd512553eccf6a22a78e8bde00bb4aed84d5bdbf
"

[[package]]
name = "byteorder"
version = "1.5.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1fd0f2584146f6f2ef48085050886acf353beff7305ebd1ae69500e27c67f64b"

[[package]]
name = "bytes"
version = "1.10.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"d71b6127be86fdcfddb610f7182ac57211d4b18a3e9c82eb2d17662f2227ad6a
"

[[package]]
name = "cc"
version = "1.2.24"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"16595d3be041c03b09d08d0858631facccee9221e579704070e6e9e4915d3bc
7"
dependencies = [
    "shlex",
]

[[package]]
name = "cfg-if"
version = "1.0.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"baf1de4339761588bc0619e3cbc0120ee582ebb74b53b4efbf79117bd2da40fd
"

[[package]]
name = "cfg_aliases"
version = "0.2.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"613afe47fcd5fac7ccf1db93babcb082c5994d996f20b8b159f2ad1658eb5724"

[[package]]
name = "chrono"
```

```

version = "0.4.41"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"c469d952047f47f91b68d1cba3f10d63c11d73e4636f24f08daf0278abf01c4d"
dependencies = [
    "android-tzdata",
    "iana-time-zone",
    "js-sys",
    "num-traits",
    "serde",
    "wasm-bindgen",
    "windows-link",
]
[[package]]
name = "core-foundation"
version = "0.10.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"b55271e5c8c478ad3f38ad24ef34923091e0548492a266d19b3c0b4d82574c6
3"
dependencies = [
    "core-foundation-sys",
    "libc",
]
[[package]]
name = "core-foundation-sys"
version = "0.8.7"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"773648b94d0e5d620f64f280777445740e61fe701025087ec8b57f45c791888b
"
[[package]]
name = "futures"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"65bc07b1a8bc7c85c5f2e110c476c7389b4554ba72af57d8445ea63a576b087
6"
dependencies = [
    "futures-channel",
    "futures-core",
    "futures-executor",
    "futures-io",
    "futures-sink",
    "futures-task",
    "futures-util",
]

```

```
]

[[package]]
name = "futures-channel"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"2dff15bf788c671c1934e366d07e30c1814a8ef514e1af724a602e8a2fbe1b10"
dependencies =
["futures-core",
"futures-sink",
]
=
```

```
[[package]]
name = "futures-core"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"05f29059c0c2090612e8d742178b0580d2dc940c837851ad723096f87af6663e"
"
```

```
[[package]]
name = "futures-executor"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1e28d1d997f585e54aebc3f97d39e72338912123a67330d723fdbb564d646c9f"
"
dependencies =
["futures-core",
"futures-task",
"futures-util",
]
=
```

```
[[package]]
name = "futures-io"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"9e5c1b78ca4aae1ac06c48a526a655760685149f0d465d21f37abfe57ce075c6"
"
```

```
[[package]]
name = "futures-macro"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"162ee34ebcb7c64a8abebc059ce0fee27c2262618d7b60ed8faf72fef13c3650"
dependencies = [
```

```

"proc-macro2",
"quote",
"syn",
]

[[package]]
name = "futures-sink"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"e575fab7d1e0dcb8d0c7bcf9a63ee213816ab51902e6d244a95819acacf1d4f7"

[[package]]
name = "futures-task"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"f90f7dce0722e95104fcb095585910c0977252f286e354b5e3bd38902cd99988"
"

[[package]]
name = "futures-util"
version = "0.3.31"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"9fa08315bb612088cc391249efdc3bc77536f16c91f6cf495e6fbe85b20a4a81"
dependencies = [
    "futures-channel",
    "futures-core",
    "futures-io",
    "futures-macro",
    "futures-sink",
    "futures-task",
    "memchr",
    "pin-project-lite",
    "pin-utils",
    "slab",
]
[[package]]
name = "gimli"
version = "0.31.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"07e28edb80900c19c28f1072f2e8aec7fa06b23cd4169cef1af5aa3260783f"

[[package]]
name = "iana-time-zone"
version = "0.1.63"

```

```

source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"b0c919e5debc312ad217002b8048a17b7d83f80703865bbfcfebb0458b0b27d
8"
dependencies = [
    "android_system_properties",
    "core-foundation-sys",
    "iana-time-zone-haiku",
    "js-sys",
    "log",
    "wasm-bindgen",
    "windows-core",
]
[[package]]
name = "iana-time-zone-haiku"
version = "0.1.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"f31827a206f56af32e590ba56d5d2d085f558508192593743f16b2306495269f"
dependencies = [
    "cc",
]
[[package]]
name = "io-kit-sys"
version = "0.4.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"617ee6cf8e3f66f3b4ea67a4058564628cde41901316e19f559e14c7c72c5e7b"
dependencies = [
    "core-foundation-sys",
    "mach2",
]
[[package]]
name = "itoa"
version = "1.0.15"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"4a5f13b858c8d314ee3e8f639011f7ccefe71f97f96e50151fb991f267928e2c"
[[package]]
name = "js-sys"
version = "0.3.77"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1cfaf33c695fc6e08064efbc1f72ec937429614f25eef83af942d0e227c3a28f"
dependencies =

```

```
"once_cell",
"wasm-bindgen",
]

[[package]]
name = "libc"
version = "0.2.172"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"d750af042f7ef4f724306de029d18836c26c1765a54a6a3f094cbd23a7267ffa"

[[package]]
name = "lock_api"
version = "0.4.12"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"07af8b9cdd281b7915f413fa73f29ebd5d55d0d3f0155584dade1ff18cea1b17"
dependencies = [
    "autocfg",
    "scopeguard",
]
[[package]]
name = "log"
version = "0.4.27"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"13dc2df351e3202783a1fe0d44375f7295ffb4049267b0f3018346dc122a1d94"

[[package]]
name = "mach2"
version = "0.4.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"19b955cdeb2a02b9117f121ce63aa52d08ade45de53e48fe6a38b39c10f6f709"
"
dependencies = [
    "libc",
]
[[package]]
name = "memchr"
version = "2.7.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"78ca9ab1a0babb1e7d5695e3530886289c18cf2f87ec19a575a0abdce112e3a
3"
[[package]]
```

```

name = "miniz_oxide"
version = "0.8.8"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"3be647b768db090acb35d5ec5db2b0e1f1de11133ca123b9eacf5137868f892a"
"
dependencies = [
"adler2",
]
[[package]]
name = "mio"
version = "1.0.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"78bed444cc8a2160f01cbcf811ef18cac863ad68ae8ca62092e8db51d51c761c"
"
dependencies = [
"libc",
"log",
"wasi",
"windows-sys 0.59.0",
]
[[package]]
name = "mio-serial"
version = "5.0.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"029e1f407e261176a983a6599c084efd322d9301028055c87174beac71397ba
3"
dependencies = [
"log",
"mio",
"nix 0.29.0",
"serialport",
"winapi",
]
[[package]]
name = "nix"
version = "0.26.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"598beaf3cc6fdd9a5dfb1630c2800c7acd31df7aaaf0f565796fba2b53ca1af1b"
dependencies = [
"bitflags 1.3.2",
"cfg-if",
"libc",
]

```

```
]

[[package]]
name = "nix"
version = "0.29.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"71e2746dc3a24dd78b3cfcb7be93368c6de9963d30f43a6a73998a9cf4b17b46"
"
dependencies = [
"bitflags 2.9.1",
"cfg-if",
"cfg_aliases",
"libc",
]
}

[[package]]
name = "num-traits"
version = "0.2.19"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"071dfc062690e90b734c0b2273ce72ad0ffa95f0c74596bc250dcfd960262841"
dependencies = [
"autocfg",
]
}

[[package]]
name = "object"
version = "0.36.7"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"62948e14d923ea95ea2c7c86c71013138b66525b86bdc08d2dcc262bdb497b
87"
dependencies = [
"memchr",
]
}

[[package]]
name = "once_cell"
version = "1.21.3"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"42f5e15c9953c5e4ccceeb2e7382a716482c34515315f7b03532b8b4e8393d2
d"
}

[[package]]
name = "parking_lot"
version = "0.12.3"
source = "registry+https://github.com/rust-lang/crates.io-index"
```

```

checksum = "f1bf18183cf54e8d6059647fc3063646a1801cf30896933ec2311622cc4b9a27"
dependencies = [
    "lock_api",
    "parking_lot_core",
]

[[package]]
name = "parking_lot_core"
version = "0.9.10"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1e401f977ab385c9e4e3ab30627d6f26d00e2c73eef317493c4ec6d468726cf8"
dependencies = [
    "cfg-if",
    "libc",
    "redox_syscall",
    "smallvec",
    "windows-targets",
]
[[package]]
name = "pin-project-lite"
version = "0.2.16"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"3b3cff922bd51709b605d9ead9aa71031d81447142d828eb4a6eba76fe619f9b"
"
[[package]]
name = "pin-utils"
version = "0.1.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"8b870d8c151b6f2fb93e84a13146138f05d02ed11c7e7c54f8826aaaf7c9f184"
[[package]]
name = "proc-macro2"
version = "1.0.95"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"02b3e5e68a3a1a02aad3ec490a98007cbc13c37cbe84a3cd7b8e406d76e7f77
8"
dependencies = [
    "unicode-ident",
]
[[package]]
name = "quote"

```

```
version = "1.0.40"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1885c039570dc00dcb4ff087a89e185fd56bae234ddc7f056a945bf36467248d"
dependencies =
["proc-macro2",
]
[[package]]
name = "redox_syscall"
version = "0.5.12"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"928fcfa9cf2aa042393a8325b9ead81d2f0df4cb12e1e24cef072922ccd99c5af"
dependencies =
["bitflags 2.9.1",
]
[[package]]
name = "rustc-demangle"
version = "0.1.24"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"719b953e2095829ee67db738b3bfa9fa368c94900df327b3f07fe6e794d2fe1f"
[[package]]
name = "rustversion"
version = "1.0.21"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"8a0d197bd2c9dc6e53b84da9556a69ba4cdfab8619eb41a8bd1cc2027a0f6b1
d"
[[package]]
name = "ryu"
version = "1.0.20"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"28d3b2b1366ec20994f1fd18c3c594f05c5dd4bc44d8bb0c1c632c8d6829481f"
[[package]]
name = "scopeguard"
version = "1.2.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"94143f37725109f92c262ed2cf5e59bce7498c01bcc1502d7b9afe439a4e9f49"
[[package]]
name = "sensor_client"
```

```

version = "0.1.0"
dependencies = [
    "chrono",
    "serde",
    "serde_json",
    "tokio",
    "tokio-modbus",
    "tokio-serial",
]
[[package]]
name = "serde"
version = "1.0.219"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"5f0e2c6ed6606019b4e29e69dbaba95b11854410e5347d525002456dbbb786
b6"
dependencies = [
    "serde_derive",
]
[[package]]
name = "serde_derive"
version = "1.0.219"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"5b0276cf7f2c73365f7157c8123c21cd9a50fbcd844757af28ca1f5925fc2a00"
dependencies = [
    "proc-macro2",
    "quote",
    "syn",
]
[[package]]
name = "serde_json"
version = "1.0.140"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"20068b6e96dc6c9bd23e01df8827e6c7e1f2fddd43c21810382803c136b99373
"
dependencies = [
    "itoa",
    "memchr",
    "ryu",
    "serde",
]
[[package]]
name = "serialport"

```

```

version = "4.7.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"cdb0bc984f6af6ef8bab54e6cf2071579ee75b9286aa9f2319a0d220c28b0a2b"
dependencies = [
"bitflags 2.9.1",
"cfg-if",
"core-foundation",
"core-foundation-sys",
"io-kit-sys",
"mach2",
"nix 0.26.4",
"scopeguard",
"unescaper",
"winapi",
]
[[package]]
name = "shlex"
version = "1.3.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"0fdaf2ff0d084019ba4d7c6f371c95d8fd75ce3524c3cb8fb653a3023f6323e64"
[[package]]
name = "signal-hook-registry"
version = "1.4.5"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"9203b8055f63a2a00e2f593bb0510367fe707d7ff1e5c872de2f537b339e5410"
dependencies = [
"libc",
]
[[package]]
name = "slab"
version = "0.4.9"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"8f92a496fb766b417c996b9c5e57daf2f7ad3b0bebe1ccfca4856390e3d3bb67"
dependencies = [
"autocfg",
]
[[package]]
name = "smallvec"
version = "1.15.0"
source = "registry+https://github.com/rust-lang/crates.io-index"

```

```

checksum = "8917285742e9f3e1683f0a9c4e6b57960b7314d0b08d30d1ecd426713ee2eee9"

[[package]]
name = "socket2"
version = "0.5.9"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "4f5fd57c80058a56cf5c777ab8a126398ece8e442983605d280a44ce79d0edef"
dependencies = [
    "libc",
    "windows-sys 0.52.0",
]

[[package]]
name = "syn"
version = "2.0.101"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "8ce2b7fc941b3a24138a0a7cf8e858bfc6a992e7978a068a5c760deb0ed43caf"
dependencies = [
    "proc-macro2",
    "quote",
    "unicode-ident",
]

```

```

[[package]]
name = "thiserror"
version = "2.0.12"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "567b8a2dae586314f7be2a752ec7474332959c6460e02bde30d702a66d488708"
dependencies = [
    "thiserror-impl",
]

```

```

[[package]]
name = "thiserror-impl"
version = "2.0.12"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "7f7cf42b4507d8ea322120659672cf1b9dbb93f8f2d4ecfd6e51350ff5b17a1d"
dependencies = [
    "proc-macro2",
    "quote",
    "syn",
]

```

```
[[package]]
name = "tokio"
version = "1.45.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"75ef51a33ef1da925cea3e4eb122833cb377c61439ca401b770f54902b806779"
"
dependencies = [
"backtrace",
"bytes",
"libc",
"mio",
"parking_lot",
"pin-project-lite",
"signal-hook-registry",
"socket2",
"tokio-macros",
"windows-sys 0.52.0",
]

```

```
[[package]]
name = "tokio-macros"
version = "2.5.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"6e06d43f1345a3bcd39f6a56dbb7dcab2ba47e68e8ac134855e7e2bdbaf8cab8"
"
dependencies = [
"proc-macro2",
"quote",
"syn",
]

```

```
[[package]]
name = "tokio-modbus"
version = "0.5.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"e757b218fb8634aced3517ed0092da74d56f0db1be9e9f0429d520e611ef72c2"
"
dependencies = [
"async-trait",
"byteorder",
"bytes",
"futures-util",
"log",
"smallvec",
"tokio",
]
```

```

"tokio-serial",
"tokio-util",
]

[[package]]
name = "tokio-serial"
version = "5.4.5"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"aa1d5427f11ba7c5e6384521cf76f2d64572ff29f3f4f7aa0f496282923fdc8"
dependencies = [
"cfg-if",
"futures",
"log",
"mio-serial",
"serialport",
"tokio",
]
=
```

```

[[package]]
name = "tokio-util"
version = "0.7.15"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"66a539a9ad6d5d281510d5bd368c973d636c02dbf8a67300bfb6b950696ad7d
f"
dependencies = [
"bytes",
"futures-core",
"futures-sink",
"pin-project-lite",
"tokio",
]
=
```

```

[[package]]
name = "unescaper"
version = "0.1.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"c01d12e3a56a4432a8b436f293c25f4808bdf9e9f9f98f9260bba1f1bc5a1f26"
dependencies = [
>thiserror",
]
=
```

```

[[package]]
name = "unicode-ident"
version = "1.0.18"
source = "registry+https://github.com/rust-lang/crates.io-index"
```

```
checksum = "5a5f39404a5da50712a4c1eecf25e90dd62b613502b7e925fd4e4d19b5c9651  
2"  
  
[[package]]  
name = "wasi"  
version = "0.11.0+was-snapshot-preview1"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum = "9c8d87e72b64a3b4db28d11ce29237c246188f4f51057d65a7eab63b7987e42  
3"  
  
[[package]]  
name = "wasm-bindgen"  
version = "0.2.100"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum = "1edc8929d7499fc4e8f0be2262a241556fc54a0bea223790e71446f2aab1ef5"  
dependencies = [  
    "cfg-if",  
    "once_cell",  
    "rustversion",  
    "wasm-bindgen-macro",  
]  
  
[[package]]  
name = "wasm-bindgen-backend"  
version = "0.2.100"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum = "2f0a0651a5c2bc21487bde11ee802ccaf4c51935d0d3d42a6101f98161700bc6  
dependencies = [  
    "bumpalo",  
    "log",  
    "proc-macro2",  
    "quote",  
    "syn",  
    "wasm-bindgen-shared",  
]  
  
[[package]]  
name = "wasm-bindgen-macro"  
version = "0.2.100"  
source = "registry+https://github.com/rust-lang/crates.io-index"  
checksum = "7fe63fc6d09ed3792bd0897b314f53de8e16568c2b3f7982f468c0bf9bd0b407"  
dependencies = [  
    "quote",
```

```

    "wasm-bindgen-macro-support",
]

[[package]]
name = "wasm-bindgen-macro-support"
version = "0.2.100"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"8ae87ea40c9f689fc23f209965b6fb8a99ad69aeeb0231408be24920604395de"
"
dependencies = [
"proc-macro2",
"quote",
"syn",
"wasm-bindgen-backend",
"wasm-bindgen-shared",
]
=
```

```

[[package]]
name = "wasm-bindgen-shared"
version = "0.2.100"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1a05d73b933a847d6ccdda8f838a22ff101ad9bf93e33684f39c1f5f0eece3d"
dependencies = [
"unicode-ident",
]
=
```

```

[[package]]
name = "winapi"
version = "0.3.9"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"5c839a674fc7a98952e593242ea400abe93992746761e38641405d28b00f41
9"
dependencies = [
"winapi-i686-pc-windows-gnu",
"winapi-x86_64-pc-windows-gnu",
]
=
```

```

[[package]]
name = "winapi-i686-pc-windows-gnu"
version = "0.4.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"ac3b87c63620426dd9b991e5ce0329eff545bccbbb34f3be09ff6fb6ab51b7b6"
=
```

```

[[package]]
name = "winapi-x86_64-pc-windows-gnu"
```

```
version = "0.4.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"712e227841d057c1ee1cd2fb22fa7e5a5461ae8e48fa2ca79ec42fcf1931183f" =  
  
[[package]]
name = "windows-core"
version = "0.61.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"c0fdd3ddb90610c7638aa2b3a3ab2904fb9e5cdbbecc643ddb3647212781c4ae3" =  
dependencies = [
    "windows-implement",
    "windows-interface",
    "windows-link",
    "windows-result",
    "windows-strings",
]
  
  
[[package]]
name = "windows-implement"
version = "0.60.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"a47fddd13af08290e67f4acabf4b459f647552718f683a7b415d290ac744a836" =  
dependencies = [
    "proc-macro2",
    "quote",
    "syn",
]
  
  
[[package]]
name = "windows-interface"
version = "0.59.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"bd9211b69f8dcdfa817bfd14bf1c97c9188afa36f4750130fcdf3f400eca9fa8" =  
dependencies = [
    "proc-macro2",
    "quote",
    "syn",
]
  
  
[[package]]
name = "windows-link"
version = "0.1.1"
source = "registry+https://github.com/rust-lang/crates.io-index"
```

```
checksum = "76840935b766e1b0a05c0066835fb9ec80071d4c09a16f6bd5f7e655e3c14c38"
"

[[package]]
name = "windows-result"
version = "0.3.4"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"56f42bd332cc6c8eac5af113fc0c1fd6a8fd2aa08a0119358686e5160d0586c6"
dependencies = [
    "windows-link",
]

[[package]]
name = "windows-strings"
version = "0.4.2"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"56e6c93f3a0c3b36176cb1327a4958a0353d5d166c2a35cb268ace15e91d3b5
7"
dependencies = [
    "windows-link",
]

[[package]]
name = "windows-sys"
version = "0.52.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"282be5f36a8ce781fad8c8ae18fa3f9beff57ec1b52cb3de0789201425d9a33d"
dependencies = [
    "windows-targets",
]

[[package]]
name = "windows-sys"
version = "0.59.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"1e38bc4d79ed67fd075bcc251a1c39b32a1776bbe92e5bef1f0bf1f8c531853b"
dependencies = [
    "windows-targets",
]

[[package]]
name = "windows-targets"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
```

```

checksum = "9b724f72796e036ab90c1021d4780d4d3d648aca59e491e6b98e725b84e999
73"
dependencies = [
    "windows_aarch64_gnullvm",
    "windows_aarch64_msvc",
    "windows_i686_gnu",
    "windows_i686_gnullvm",
    "windows_i686_msvc",
    "windows_x86_64_gnu",
    "windows_x86_64_gnullvm",
    "windows_x86_64_msvc",
]
[[package]]
name = "windows_aarch64_gnullvm"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "32a4622180e7a0ec044bb555404c800bc9fd9ec262ec147edd5989ccd0c02cd
3"
[[package]]
name = "windows_aarch64_msvc"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "09ec2a7bb152e2252b53fa7803150007879548bc709c039df7627cabbd05d46
9"
[[package]]
name = "windows_i686_gnu"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "8e9b5ad5ab802e97eb8e295ac6720e509ee4c243f69d781394014ebfe8bbfa0
b"
[[package]]
name = "windows_i686_gnullvm"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "0eeee52d38c090b3caa76c563b86c3a4bd71ef1a819287c19d586d7334ae8ed6
6"
[[package]]
name = "windows_i686_msvc"
version = "0.52.6"

```

```

source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"240948bc05c5e7c6dabba28bf89d89ffce3e303022809e73deaefe4f6ec56c66" =


[[package]]
name = "windows_x86_64_gnu"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"147a5c80aabfbf0c7d901cb5895d1de30ef2907eb21fbab29ca94c5b08b1a78" =


[[package]]
name = "windows_x86_64_gnullvm"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"24d5b23dc417412679681396f2b49f3de8c1473deb516bd34410872eff51ed0d" =


[[package]]
name = "windows_x86_64_msvc"
version = "0.52.6"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"589f6da84c646204747d1270a2a5661ea66ed1cced2631d546fdfb155959f9ec" =
"
```

## 2. Codingan Cargo.toml

```

[package]
name = "sensor_client"
version = "0.1.0"
edition = "2021"

[dependencies]
tokio = { version = "1.32", features = ["full"] }
tokio-serial = "5.4.0"
tokio-modbus = "0.5.0"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
chrono = { version = "0.4", features = ["serde"] }
```

## 3. Codingan main.rs

```

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-Monitoring-001".into(),
                    location: "Green House Cabai".into(),
                    process_stage: "Monitoring".into(),
                    temperature_celsius: temp,
                    humidity_percent: rh,
                };

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

                match TcpStream::connect("172.20.10.4:9000").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 koneksi ke TCP server: {}", e);
    }
},
Ok(other) => {
    println!(" ⚠️ Data tidak lengkap: {:?}", other);
},
Err(e) => {
    println!(" ❌ Gagal baca sensor: {}", e);
}
}

sleep(Duration::from_secs(5)).await;
}
}

```

#### 4. Codingan

/home/alimakki/Unduhan/TUBES\ GREENHOUSE/Tubes\  
 ISI/sensor/target/debug/sensor\_client: /home/alimakki/Unduhan/TUBES\  
 GREENHOUSE/Tubes\ ISI/sensor/src/[main.rs](#)

#### 5. Sensor storage

```
[{"anonymous":false,"inputs":[{"indexed":false,"internalType":"uint256","name":"timestamp","type":"uint256"}, {"indexed":false,"internalType":"string","name":"sensorId","type":"string"}, {"indexed":false,"internalType":"string","name":"location","type":"string"}, {"indexed":false,"internalType":"string","name":"stage","type":"string"}, {"indexed":false,"internalType":"int256","name":"temperature","type":"int256"}, {"indexed":false,"internalType":"int256","name":"humidity","type":"int256"}], "name": "DataStored", "type": "event"}, {"inputs": [{"internalType": "uint256", "name": "timestamp", "type": "uint256"}, {"internalType": "string", "name": "sensorId", "type": "string"}, {"internalType": "string", "name": "location", "type": "string"}, {"internalType": "string", "name": "stage", "type": "string"}, {"internalType": "int256", "name": "temperature", "type": "int256"}, {"internalType": "int256", "name": "humidity", "type": "int256"}], "name": "storeData", "outputs": [], "stateMutability": "nonpayable", "type": "function"}]
```

#### 6. sensor storage.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);
    }
}

```

## 7. style.css web3

```

/* Reset */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #e0f7fa, #ffffff);
    color: #333;
    padding: 20px;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
}

header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
}

h1 {

```

```
font-size: 2rem;
color: #00796b;
}

button {
background-color: #00796b;
color: white;
border: none;
padding: 12px 18px;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
transition: background 0.3s ease;
}

button:hover {
background-color: #004d40;
}

.table-section {
overflow-x: auto;
margin-top: 20px;
}

table {
width: 100%;
border-collapse: collapse;
background: white;
border-radius: 10px;
overflow: hidden;
box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}

th {
background-color: #004d40;
color: white;
padding: 12px;
text-align: center;
}

td {
padding: 10px;
border: 1px solid #ddd;
text-align: center;
}

tr:nth-child(even) {
background-color: #f1f1f1;
}
```

```

.chart-section {
  margin-top: 40px;
  background: #fff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}

8. web3 script.js
const contractAddress = "0xac9c6dab81ea26869da00c9c1944a1175eef5171";
// Ganti jika berbeda
const abiPath = "abi/SensorStorage.abi";
const rpcURL = "http://172.20.10.4:8545"; // ← IP VPS Ganache kamu

let chart;

async function loadSensorData() {
  const abiRes = await fetch(abiPath);
  const abi = await abiRes.json();

  const provider = new ethers.JsonRpcProvider(rpcURL);
  const contract = new ethers.Contract(contractAddress, abi, provider);

  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 }
    }
  }
});
}

```

## 9. Codingan gui.py

```

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 = "almeki"
BUCKET = "Tugas4"
TOKEN
      =
"ocdloeKU4xYD9Iz4UI2Rcq0NS0YdRM7XN3PEIR7y843m3Zx5Rzf_dkbOMy
H6Aom9SFe0PJTmizk99v3K41LyA=="

# Riwayat data
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:
                pass
    except:
        pass

```

```

        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))
        temp_map = {}
        rh_map = {}

        for row in reader:
            try:
                t = row["_time"]
                field = row["_field"]
                value = float(row["_value"])
                if field == "temperature":
                    temp_map[t] = value
                elif field == "humidity":
                    rh_map[t] = value
            except:
                continue

        sorted_keys = sorted(set(temp_map.keys()) & set(rh_map.keys()))
    
```

```

        temps = [temp_map[t] for t in sorted_keys]
        rhs = [rh_map[t] for t in sorted_keys]
        times = [t[11:19] for t in sorted_keys] # jam:menit:detik

    return temps, rhs, times
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():
    ax1.clear()
    ax2.clear()

    # Set background ke hitam
    fig.patch.set_facecolor('black')
    ax1.set_facecolor('black')
    ax2.set_facecolor('black')

    x = list(range(len(time_history)))
    times = list(time_history)

    # Plot data dengan garis dan warna yang kontras
    ax1.plot(x, list(temp_history), label='Suhu (°C)', color='red', marker='o',
              linestyle='-')



```

```

    ax2.plot(x, list(rh_history), 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')

# Tampilkan hanya label waktu setiap 5 data
interval = 5
tick_positions = x[::interval]
tick_labels = times[::interval]

    ax1.set_xticks(tick_positions)
    ax2.set_xticks(tick_positions)
    ax1.set_xticklabels(tick_labels, rotation=45, ha="right", color='white')
    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()

def show_history():
    start = entry_start.get()
    end = entry_end.get()
    temps, rhs, times = get_data_range(start, end)

    if temps and rhs:
        temp_history.clear()
        rh_history.clear()
        time_history.clear()

        temp_history.extend(temps)
        rh_history.extend(rhs)
        time_history.extend(times)

        label_temp.config(text="(Hist) Suhu: -- °C")
        label_rh.config(text="(Hist) RH: -- %")
        status_label.config(text="  Menampilkan data historis")
        plot_graph()
    else:
        status_label.config(text="  Tidak ada data historis")

```

```

# GUI Setup
root = tk.Tk()
root.title("Monitor SHT20 dari InfluxDB")
root.geometry("800x650")

label_temp = tk.Label(root, text="Suhu: -- °C", font=("Helvetica", 16))
label_temp.pack(pady=5)

label_rh = tk.Label(root, text="Kelembaban: -- %", font=("Helvetica", 16))
label_rh.pack(pady=5)

status_label = tk.Label(root, text="Status: ---", fg="blue")
status_label.pack(pady=5)

frame_input = tk.Frame(root)
frame_input.pack(pady=5)

tk.Label(frame_input, text="Start (RFC3339):").grid(row=0, column=0, padx=5)
entry_start = tk.Entry(frame_input, width=30)
entry_start.grid(row=0, column=1)

tk.Label(frame_input, text="End (RFC3339):").grid(row=1, column=0, padx=5)
entry_end = tk.Entry(frame_input, width=30)
entry_end.grid(row=1, column=1)

btn_show      =      tk.Button(frame_input,      text="Tampilkan Riwayat",
command=show_history)
btn_show.grid(row=0, column=2, rowspan=2, padx=10)

fig = Figure(figsize=(6, 4), dpi=100)
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)

canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(pady=10)

# Mulai update realtime
threading.Thread(target=update_data, daemon=True).start()

root.mainloop()

```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>  Greenhouse Sensor Dashboard</title>

```

```

<!-- Font & Style -->
<link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@500&display=swap" rel="stylesheet">
<style>
body {
  margin: 0;
  font-family: 'Orbitron', sans-serif;
  background: linear-gradient(to right, #0f2027, #203a43, #2c5364);
  color: white;
}
.container {
  padding: 2rem;
  max-width: 1200px;
  margin: auto;
}
header {
  text-align: center;
  margin-bottom: 2rem;
}
h1 {
  font-size: 2.5rem;
  margin-bottom: 1rem;
  text-shadow: 0 0 10px #00f3ff;
}
button {
  background: #00f3ff;
  color: #000;
  border: none;
  padding: 10px 20px;
  font-size: 1rem;
  border-radius: 30px;
  cursor: pointer;
  transition: transform 0.3s, background 0.3s;
}
button:hover {
  background: #00c4d3;
  transform: scale(1.05);
}
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
  background-color: rgba(255, 255, 255, 0.05);
  border-radius: 8px;
  overflow: hidden;
}
th, td {
  padding: 12px;
  text-align: center;
  border-bottom: 1px solid rgba(255, 255, 255, 0.1);
}
th {
  background-color: rgba(0, 243, 255, 0.2);
  color: #00f3ff;
}

```

```

.chart-section {
  margin-top: 40px;
}
canvas {
  background: #ffff08;
  border-radius: 10px;
  padding: 1rem;
}
</style>

<!-- Lib -->
<script
src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.umd.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
<div class="container">
<header>
  <h1>🌱 Greenhouse Sensor Dashboard</h1>
  <button onclick="loadSensorData()">🔄 Load Sensor Data</button>
</header>

<section class="table-section">
  <table id="sensorTable">
    <thead>
      <tr>
        <th>Timestamp</th>
        <th>Sensor ID</th>
        <th>Location</th>
        <th>Stage</th>
        <th>Temperature (°C)</th>
        <th>Humidity (%)</th>
        <th>Energy (kWh)</th>
        <th>Biaya (Rp)</th>
      </tr>
    </thead>
    <tbody>
      <!-- Data akan dimuat di sini -->
    </tbody>
  </table>
</section>

<section class="chart-section">
  <canvas id="chart" height="100"></canvas>
</section>
</div>

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

```

const	contractAddress	=
-------	-----------------	---

```

"0x9473c354d520e99bb7819d90b6b7a6a062952621";
const abiPath = "abi/SensorStorage.abi";
const rpcURL = "http://127.0.0.1:8545";

let chart;

async function loadSensorData() {
  const abiRes = await fetch(abiPath);
  const abi = await abiRes.json();

  const provider = new ethers.JsonRpcProvider(rpcURL);
  const contract = new ethers.Contract(contractAddress, abi, provider);

  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;

    const energy = calculateEnergyCost(temp, hum);

    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>
        <td>${energy.kWh}</td>
        <td>Rp ${energy.cost.toLocaleString('id-ID')}</td>
      </tr>
    `;

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

  renderChart(labels, temps, hums);
}

function calculateEnergyCost(temp, humidity) {
  const baseTariff = 1444.7; // IDR per kWh (PLN Industrial Rate)
  let kWh = 1; // Base consumption

```

```

let extraCooling = 0;
let extraHumidity = 0;

if (temp > 27) {
  extraCooling = (temp - 27) * 0.5; // 0.5 kWh per °C over 27
}

if (humidity < 50 || humidity > 85) {
  extraHumidity = 0.75; // 0.75 kWh per abnormal humidity
}

const totalKWh = kWh + extraCooling + extraHumidity;
const cost = totalKWh * baseTariff;

return {
  kWh: totalKWh.toFixed(2),
  cost: Math.round(cost)
};
}

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 }
      }
    }
  });
}

```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>➤ Greenhouse Cabai Dashboard</title>

  <!-- Font & Style -->
  <link
    href="https://fonts.googleapis.com/css2?family=Orbitron:wght@500&display=swap" rel="stylesheet">
  <style>
    body {
      margin: 0;
      font-family: 'Orbitron', sans-serif;
      background: linear-gradient(to right, #0f2027, #203a43, #2c5364);
      color: white;
    }
    .container {
      padding: 2rem;
      max-width: 1200px;
      margin: auto;
    }
    header {
      text-align: center;
      margin-bottom: 2rem;
    }
    h1 {
      font-size: 2.5rem;
      margin-bottom: 1rem;
      text-shadow: 0 0 10px #00f3ff;
    }
    button {
      background: #00f3ff;
      color: #000;
      border: none;
      padding: 10px 20px;
      font-size: 1rem;
      border-radius: 30px;
      cursor: pointer;
      transition: transform 0.3s, background 0.3s;
    }
    button:hover {
      background: #00c4d3;
      transform: scale(1.05);
    }
    .info-section {
      background: rgba(0, 243, 255, 0.1);
      padding: 1rem 1.5rem;
      border-left: 5px solid #00f3ff;
      border-radius: 8px;
      margin-bottom: 2rem;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
    }
    .info-section h2 {
      font-size: 1.5rem;
    }
  </style>

```

```

margin-bottom: 0.5rem;
color: #ffffff;
text-shadow: 0 0 6px #00f3ff;
}
.info-section p {
line-height: 1.6;
font-size: 1rem;
color: #e0f7fa;
}
table {
width: 100%;
border-collapse: collapse;
margin-top: 20px;
background-color: rgba(255, 255, 255, 0.05);
border-radius: 8px;
overflow: hidden;
}
th, td {
padding: 12px;
text-align: center;
border-bottom: 1px solid rgba(255, 255, 255, 0.1);
}
th {
background-color: rgba(0, 243, 255, 0.2);
color: #00f3ff;
}
.chart-section {
margin-top: 40px;
}
canvas {
background: #fffff08;
border-radius: 10px;
padding: 1rem;
}
small {
display: block;
margin-top: 4px;
}

```

</style>

```

<!-- Lib -->
<script
src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.umd.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
<div class="container">
<header>
<h1><img alt="Chili" style="vertical-align: middle; margin-right: 10px;">Greenhouse Cabai Smart Monitor</h1>
<button onclick="loadSensorData()">  Load Sensor Data</button>
</header>

<section class="info-section">
<h2>  Informasi Biaya Listrik & Set Point Greenhouse</h2>

```

```

<p>
     Tarif dasar listrik: <strong>Rp 1.444,70 / kWh</strong><br/>
     Suhu ideal: <strong>20°C - 27°C</strong>. Di atas atau di bawah rentang ini, sistem pendingin atau pemanas akan aktif.<br/>
     Kelembapan ideal: <strong>50% - 85%</strong>. Di luar rentang ini, pompa humidifier atau dehumidifier aktif.
</p>
</section>

<section class="table-section">
    <table id="sensorTable">
        <thead>
            <tr>
                <th>Timestamp</th>
                <th>Sensor ID</th>
                <th>Location</th>
                <th>Stage</th>
                <th>Temperature (°C)</th>
                <th>Humidity (%)</th>
                <th>Energy (kWh)</th>
                <th>Biaya (Rp)</th>
            </tr>
        </thead>
        <tbody>
            <!-- Data will be loaded here -->
        </tbody>
    </table>
</section>

<section class="chart-section">
    <canvas id="chart" height="100"></canvas>
</section>
</div>

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

```

```

const contractAddress = "0x9473c354d520e99bb7819d90b6b7a6a062952621";
const abiPath = "abi/SensorStorage.abi";
const rpcURL = "http://127.0.0.1:8545";

let chart;

async function loadSensorData() {
    const abiRes = await fetch(abiPath);
    const abi = await abiRes.json();

    const provider = new ethers.JsonRpcProvider(rpcURL);
    const contract = new ethers.Contract(contractAddress, abi,

```

```

provider);

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;

  const energy = calculateEnergyCost(temp, hum);

  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>
<td>${energy.kWh}</td>
<td>
  Rp ${energy.cost.toLocaleString('id-ID')}
  <small style="color:#00f3ff">${energy.note}</small>
</td>
</tr>
`;

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

renderChart(labels, temps, hums);
}

function calculateEnergyCost(temp, humidity) {
  const baseTariff = 1444.7;

```

```

let kWh = 1;
let details = [];

if (temp > 27) {
    const cooling = (temp - 27) * 0.5;
    kWh += cooling;
    details.push(`+${cooling.toFixed(2)} kWh (Cooling Fan)`);
} else if (temp < 20) {
    const heating = (20 - temp) * 0.3;
    kWh += heating;
    details.push(`+${heating.toFixed(2)} kWh (Heater)`);
}

if (humidity < 50) {
    kWh += 0.75;
    details.push(`+0.75 kWh (Humidifier)`);
} else if (humidity > 85) {
    kWh += 0.75;
    details.push(`+0.75 kWh (Dehumidifier)`);
}

const cost = kWh * baseTariff;

return {
    kWh: kWh.toFixed(2),
    cost: Math.round(cost),
    note: details.join(', ')
};
}

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 }
    }
});
}
```