Logo

Description automatically generated

**LAPORAN PROGESS TUGAS INTERKONEKSI SISTEM INSTRUMENTASI – VI231418**

**Sistem Monitoring Suhu dan Kelembaban Buah dalam Kontainer Logistik**

Nama Kelompok : 2

Anggota Kelompok :

**Muhammad Aufa Affandi (2042231011)**

**Abi Hanifa (2042231021)**

**Egga Terbyd Fabryan (2042231029)**

Dosen Pengampu

**Dwi Oktavianto Wahyu Nugroho, S.T., M.T.**

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

**Program Studi Rekayasa Teknologi Instrumentasi**

Departemen Teknik Instrumentasi

Fakultas Vokasi

Institut Teknologi Sepuluh Nopember

Surabaya

Tahun 2025

Icon

Description automatically generated

**LAPORAN PROGGES TUGAS INTERKONEKSI SISTEM INSTRUMENTASI – VI231418**

**Sistem Monitoring Suhu dan Kelembaban Buah dalam Kontainer Logistik**

Nama Kelompok : 2

Anggota Kelompok :

**Muhammad Aufa Affandi (2042231011)**

**Abi Hanifa (2042231021)**

**Egga Terbyd Fabryan (2042231029)**

Dosen Pengampu

**Dwi Oktavianto Wahyu Nugroho, S.T., M.T.**

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

**Program Studi Rekayasa Teknologi Instrumentasi**

Departemen Teknik Instrumentasi

Fakultas Vokasi

Institut Teknologi Sepuluh Nopember

Surabaya

Tahun 2025

**ABSTRAK**

**Sistem Monitoring Suhu dan Kelembaban Buah dalam Kontainer Logistik**

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa / NRP** | **:** | **Muhammad Aufa Affandi / 2042231011**  **Abi Hanifa / 2042231021**  **Egga Terbyd Fabryan / 2042231029** |
| **Departemen** | **:** | **Teknik Instrumentasi VOKASI - ITS** |
| **Dosen Pengampu** | **:** | **Ahmad Radhy, S.Si., M.Si.** |

Distribusi buah-buahan segar sangat rentan terhadap perubahan suhu dan kelembaban yang dapat menurunkan kualitas produk. Untuk mengatasi hal tersebut, dikembangkan sistem monitoring suhu dan kelembaban real-time berbasis sensor SHT20 yang terintegrasi dengan teknologi Internet of Things (IoT) dan blockchain. Data dari sensor dibaca menggunakan Modbus RTU melalui program Rust Modbus Client, dikonversi menjadi format JSON, dan dikirimkan secara periodik ke server TCP. Server memproses data dan menyimpannya dalam InfluxDB sebagai basis data time-series.

Data yang tersimpan divisualisasikan melalui Grafana untuk memudahkan pemantauan dalam bentuk grafik suhu dan kelembaban terhadap waktu. Selain itu, data juga dicatat ke dalam smart contract Ethereum menggunakan pustaka ethers-rs agar tidak dapat dimanipulasi dan dapat diverifikasi oleh semua pihak. Pengguna dapat mengakses data tersebut melalui aplikasi desentralisasi (DApp) berbasis Web3 yang dibangun menggunakan HTML, CSS, JavaScript, dan Ethers.js serta terhubung ke MetaMask.

Hasil implementasi menunjukkan sistem mampu membaca, mengirim, menyimpan, dan menampilkan data suhu dan kelembaban secara real-time serta mencatat data penting ke dalam blockchain secara otomatis. Sistem ini memberikan solusi yang transparan dan aman untuk pemantauan kualitas produk hortikultura selama proses distribusi. .

**Kata kunci: sensor suhu dan kelembaban, Rust, Modbus RTU, InfluxDB, Grafana, Blockchain, Smart Contract, Web3, DApp.**

# DAFTAR ISI

ABSTRAK....................................................................................................i

DAFTAR ISI................................................................................................ii

* 1. [Latar Belakang 3](#_TOC_250014)
  2. [Rumusan Masalah 4](#_TOC_250013)
  3. [Tujuan Proyek 4](#_bookmark2)

[BAB II TINJAUAN PUSTAKA 5](#_TOC_250012)

2.1 [Metodologi dan Arsitektur Sistem 5](#_TOC_250011)

2.1.1 Desain Arsitektur Sistem 5

2.2 Deskripsi Komponen 6

2.2.1 Sensor: SHT20 (industrial) dengan komunikasi Modbus RTU ..........................................................................................................6

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

dari sensor..............................................................................................7

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

ke InfluxDB...........................................................................................7

2.2.4 InfluxDB: Menyimpan data time-series.......................................7

2.2.5 Grafana: Menampilkan dashboard suhu dan kelembaban............7

2.3 Format Payload ..................................................................................8

[BAB III HASIL PERCOBAAN 9](#_TOC_250008)

* 1. Implementasi dan Kode Program 9

3.1.1 [Kode Rust Modbus Client 9](#_TOC_250007)

3.1.2 Kode Rust TCP Server [10](#_TOC_250006)

3.1.3 Konfigurasi influxDB dan Integrasi 11

3.1.4 Dashboard Grafana 12

3.1.5 Integrasi Blockchain dan Web3 14

3.1.6 Hasil yang dicapai 16

BAB IV [Pengujian dan Hasil 17](#_bookmark3)

BAB V Kesimpulan dan Rekomendasi 20

[DAFTAR PUSTAKA 21](#_TOC_250000)

LAMPIRAN................................................................................................22

# BAB I PENDAHULUAN

## Latar Belakang

Buah-buahan merupakan komoditas hortikultura yang sangat rentan terhadap perubahan suhu dan kelembaban selama proses distribusi. Ketidakstabilan kondisi lingkungan dapat mempercepat proses respirasi, pematangan dini, dan pembusukan, yang berdampak langsung terhadap kualitas dan nilai jual produk (Fadeyibi et al., 2021; Oliveira et al., 2020). Kontrol suhu dan kelembaban yang optimal terbukti secara signifikan dapat memperpanjang umur simpan dan menjaga mutu produk hortikultura (Rahman et al., 2023).

Sensor SHT20 yang berbasis teknologi kapasitif dan termistor menawarkan pengukuran suhu dan kelembaban dengan akurasi tinggi, yaitu ±0.3°C untuk suhu dan ±2% RH untuk kelembaban relatif. Rentang operasi sensor ini yang luas (−40°C hingga 125°C dan 0–100% RH) menjadikannya cocok untuk monitoring lingkungan selama distribusi produk segar (Sensirion, 2021; Chen et al., 2021).

Data sensor dikirim melalui protokol Modbus RTU dan dikodekan dalam format JSON menggunakan sistem tersemat berbasis Rust, yang dirancang untuk efisiensi dan keamanan dalam pemrosesan data (Rahman et al., 2023). Data yang diterima server selanjutnya disimpan dalam database time-series InfluxDB v2, yang sangat sesuai untuk menyimpan data sensor berkala secara efisien dan mendukung analisis berbasis waktu (Wang et al., 2021). Visualisasi dilakukan menggunakan Grafana, yang terintegrasi langsung dengan InfluxDB untuk menyajikan grafik time-series suhu dan kelembaban secara interaktif dan real-time (Ghosh et al., 2022). Antarmuka lokal tambahan dapat dibuat menggunakan Python dan pustaka PyQt5, yang mampu menampilkan grafik pemantauan menggunakan PyQt Chart serta dilengkapi fitur alarm visual untuk nilai suhu/kelembaban di luar ambang batas (Singh & Bhargava, 2020).

Aplikasi desktop tersebut juga dapat diperluas dengan modul pencatatan data historis dan ekspor ke format Excel, serta menyediakan fungsi pencarian berdasarkan rentang waktu untuk mempermudah proses audit dan pelacakan distribusi (Wang et al., 2021). Untuk meningkatkan kepercayaan dan transparansi dalam rantai pasok, sistem monitoring ini diintegrasikan dengan teknologi blockchain. Dengan memanfaatkan Ethereum dan smart contract berbasis Solidity, setiap data suhu dan kelembaban dapat dicatat secara immutable ke dalam blockchain, memastikan integritas dan keterlacakan data (Park & Lee, 2021; Al-Bassam, 2022). Data dikirim dari sistem Rust ke smart contract melalui pustaka ethers-rs secara aman, menghindari manipulasi atau kehilangan data selama transmisi (Buterin, 2014).

Frontend dari sistem ini dibangun sebagai DApp menggunakan HTML, CSS, dan JavaScript (dengan Ethers.js), terhubung ke dompet MetaMask, dan mampu menampilkan data terverifikasi dalam bentuk tabel serta grafik interaktif dengan Chart.js. Hal ini memberikan nilai tambah bagi pemangku kepentingan dalam memastikan kualitas produk selama proses logistik (Al-Bassam, 2022). Dengan pendekatan sistem monitoring terintegrasi berbasis sensor, InfluxDB, visualisasi real-time, dan pencatatan ke blockchain, kualitas serta keamanan distribusi buah dapat ditingkatkan melalui pengawasan data yang akurat, transparan, dan dapat diaudit.

## Rumusan Masalah

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

* + 1. Tidak adanya sistem monitoring real-time berbasis sensor di container cargo logistik
    2. Tidak adanya penyimpanan data historis suhu dan kelembaban.
    3. Perlunya visualisasi data untuk pengambilan keputusan oleh pihak logistik.

## Tujuan Proyek

Adapun tujuan dari proyek ini adalah sebagai berikut:

* + 1. Mengembangkan sistem monitoring suhu dan kelembaban real-time berbasis sensor SHT20 untuk mendukung distribusi buah-buahan.
    2. Mengirim dan menyimpan data secara efisien menggunakan protokol Modbus RTU, sistem tersemat berbasis Rust, dan database InfluxDB.
    3. Menyediakan visualisasi data secara real-time melalui Grafana dan aplikasi desktop lokal berbasis PyQt5 yang dilengkapi alarm serta pencatatan historis.
    4. Meningkatkan transparansi data dengan mencatat suhu dan kelembaban ke dalam blockchain menggunakan smart contract Ethereum dan menampilkannya melalui DApp interaktif.

# BAB II TINJAUAN PUSTAKA

## Metodologi dan Arsitektur Sistem

## 2.1.1 Desain Arsitektur Sistem

## Sebuah gambar berisi teks, cuplikan layar, Font, diagram Konten yang dihasilkan AI mungkin salah.

## Sebuah gambar berisi teks, cuplikan layar, Font, struk Konten yang dihasilkan AI mungkin salah.

## Sistem monitoring ini menggunakan sensor SHT20 untuk mengukur suhu dan kelembaban dalam kontainer logistik buah. Sensor terhubung ke komputer melalui protokol Modbus RTU, dan datanya dibaca menggunakan program Rust Modbus Client. Hasil pembacaan dikonversi ke format JSON yang memuat informasi waktu pengukuran (timestamp), ID pengiriman (shipment\_id), suhu (temperature\_celsius), dan kelembaban (humidity\_percent).

## Data JSON dikirim ke Rust TCP Server, lalu disimpan ke dalam InfluxDB sebagai basis data time-series. Data yang tersimpan divisualisasikan menggunakan Grafana Dashboard, sehingga pengguna dapat memantau suhu dan kelembaban kontainer secara real-time melalui grafik yang interaktif dan informatif.

## Deskripsi Komponen

## 2.2.1 Sensor: SHT20 (industrial) dengan komunikasi Modbus RTU

## Sebuah gambar berisi stopkontak Konten yang dihasilkan AI mungkin salah.

## Sensor SHT20 adalah sensor digital yang dirancang untuk mengukur suhu dan kelembaban dengan akurasi tinggi. Rentang suhu operasionalnya berkisar antara -40 °C hingga +125 °C, dengan akurasi ±0.3 °C pada suhu 25 °C. Untuk kelembaban, sensor ini dapat mengukur dari 0% hingga 100% RH dengan akurasi ±2% RH pada suhu 25 °C. SHT20 memiliki resolusi 0.01 °C untuk suhu dan 0.03% RH untuk kelembaban, menjadikannya pilihan yang tepat untuk aplikasi yang memerlukan pengukuran yang presisi. Sensor ini beroperasi pada tegangan antara 2.1 V hingga 3.6 V dan memiliki konsumsi daya yang rendah, yaitu 0.5 mA maksimum dalam mode aktif dan kurang dari 1 µA dalam mode tidur. Komunikasi dengan sensor dapat dilakukan melalui protokol I2C atau Modbus RTU, dengan kecepatan komunikasi hingga 1 MHz untuk I2C. Dimensi sensor ini adalah 3.6 mm x 3.6 mm x 1.0 mm tanpa casing, dan dapat beroperasi dalam suhu penyimpanan antara -40 °C hingga +125 °C serta kelembaban penyimpanan dari 0% hingga 100% RH. Dengan spesifikasi ini, SHT20 sangat cocok untuk berbagai aplikasi, termasuk pemantauan lingkungan, HVAC, dan aplikasi industri lainnya.

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

Modbus Client yang ditulis dalam bahasa pemrograman Rust bertugas untuk membaca data dari sensor SHT20. Mengimplementasikan protokol Modbus RTU, client ini dapat mengakses dan mengambil data suhu dan kelembaban yang dikirimkan oleh sensor secara efektif.

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

TCP Server yang juga dikembangkan menggunakan Rust berfungsi untuk menerima data dalam format JSON. Setelah menerima data, server ini melakukan parsing untuk mengekstrak informasi yang relevan dan menyimpannya ke dalam InfluxDB.

## InfluxDB: Menyimpan data time-series

## Sebuah gambar berisi Grafis, Font, logo, simbol Konten yang dihasilkan AI mungkin salah.

## InfluxDB adalah database yang dioptimalkan untuk menyimpan data time-series, sangat cocok untuk aplikasi pemantauan seperti ini. Data suhu dan kelembaban yang diterima dari TCP Server disimpan dengan efisien, memungkinkan akses dan analisis data yang cepat.

## Grafana: Menampilkan dashboard suhu dan kelembaban

## Sebuah gambar berisi Font, Grafis, logo, papan klip Konten yang dihasilkan AI mungkin salah.

## Grafana berfungsi sebagai platform visualisasi data yang memungkinkan pengguna untuk membuat dashboard interaktif. Dengan Grafana, pengguna dapat memantau data suhu dan kelembaban secara real-time dan melakukan analisis mendalam terhadap data yang telah dikumpulkan.

## 2.3 Format Payload

## 

## Data suhu dan kelembaban yang terbaca dari sensor dikirimkan ke server dalam bentuk payload berformat JSON. Payload ini berisi beberapa elemen penting, yaitu timestamp sebagai penanda waktu pengambilan data, shipment\_id sebagai identitas kontainer, temperature\_celsius sebagai nilai suhu dalam derajat Celsius, dan humidity\_percent sebagai nilai kelembaban dalam persen. Contoh payload yang dikirimkan adalah sebagai berikut:

|  |
| --- |
| {"timestamp": "2025-06-18T14:50:36.078688657+00:00","shipment\_id": "KONTAINER-001","temperature\_celsius": 31.7,"humidity\_percent": 64.7} |

## Format tersebut memungkinkan data disimpan secara efisien dalam database time-series dan dianalisis lebih lanjut, serta mendukung integrasi dengan sistem blockchain untuk menjamin transparansi dan integritas data. Shipment.id

# BAB III HASIL PERCOBAAN

## 3.1 Implementasi dan Kode Program

## 3.1.1 Kode Rust Modbus Client

## Untuk membaca data dari sensor suhu dan kelembaban, kelompok kami menggunakan bahasa Rust dengan pendekatan asynchronous. Sensor SHT20 dihubungkan ke sistem melalui antarmuka RS-485 ke USB, dan dikenali sebagai /dev/ttyUSB0 di sistem operasi Linux. Komunikasi dilakukan menggunakan protokol Modbus RTU, dengan alamat slave sensor sebesar 0x01.

## Data diambil dari dua register input yang menyimpan nilai suhu dan kelembaban. Setelah data dibaca dalam bentuk Vec<u16>, masing-masing nilai dikonversi ke format float dengan membaginya dengan 10, karena sensor mengirimkan data dalam format integer ×10 (contoh: 317 berarti 31.7°C). Hasil konversi kemudian dibungkus ke dalam struct SensorData yang berisi timestamp, shipment\_id, temperature\_celsius, dan humidity\_percent.

## Data tersebut dikonversi ke dalam format JSON menggunakan pustaka serde\_json dan dikirim ke server lokal melalui koneksi TCP (port 7878). Proses ini dilakukan setiap 15 detik secara berkala menggunakan fitur tokio::time::sleep, sehingga sistem dapat melakukan pengiriman data sensor secara real-time dan efisien. Contoh payload JSON yang dikirim:

|  |
| --- |
| {"timestamp": "2025-06-18T14:50:36.078688657+00:00","shipment\_id": "KONTAINER-001","temperature\_celsius": 31.7,"humidity\_percent": 64.7} |

# Pengiriman data ke TCP Server:

# Setelah data suhu dan kelembaban berhasil dikonversi, program akan mengemas informasi tersebut ke dalam format JSON yang mencakup timestamp, shipment\_id, temperature\_celsius, dan humidity\_percent. JSON ini kemudian dikirim ke TCP Server menggunakan pustaka tokio::net::TcpStream melalui alamat 127.0.0.1:7878.

# Pengiriman dilakukan secara asynchronous dengan write\_all, dan program juga dapat menerima respon balik dari server. Seluruh proses ini berjalan secara otomatis dan periodik setiap 15 detik, sehingga data sensor selalu diperbarui secara real-time di sisi server. Berikut Contoh cuplikan saat data dikirim ke server:

|  |
| --- |
| Membaca sensor untuk pengiriman: KONTAINER-001Suhu: 31.7 C, Kelembapan: 64.7 %Mengirim data: {"timestamp":"2025-06-18T14:50:36.078688657+00:00","shipment\_id":"KONTAINER-001","temperature\_celsius":31.7,"humidity\_percent":64.7}Data berhasil dikirim ke server. |

## 3.1.2 Kode Rust TCP Server

Pada sisi server, data dari client diterima melalui koneksi TCP menggunakan port 7878. Data yang diterima berupa format JSON yang berisi informasi suhu, kelembaban, timestamp, dan shipment\_id. JSON tersebut di-deserialize menggunakan pustaka serde\_json. Setelah berhasil diproses, data dikirim ke **InfluxDB** menggunakan protokol Line Protocol untuk disimpan sebagai time-series, dengan shipment\_id digunakan sebagai tag.

Selain disimpan ke InfluxDB, data juga dicatat ke dalam **Blockchain Ethereum** melalui smart contract yang ditulis dalam Solidity. Proses ini dilakukan dengan memanggil fungsi add\_data\_point menggunakan pustaka ethers-rs. Dengan demikian, setiap data yang diterima oleh server tidak hanya tersimpan secara lokal, tetapi juga secara permanen dan tidak dapat dimodifikasi di blockchain. Berikut contoh cuplikan saat server menerima dan memproses data JSON dari client:

|  |
| --- |
| async fn handle\_connection(mut stream: TcpStream) {let mut buffer = Vec::new();if let Err(e) = stream.read\_to\_end(&mut buffer).await {eprintln!("Gagal membaca data dari stream: {}", e);return;}match serde\_json::from\_slice::<SensorData>(&buffer) {Ok(data) => {println!("\n--- Menerima Data Baru ---");println!("{:?}", data);if let Err(e) = write\_to\_influxdb(&data).await {eprintln!("Error saat memproses data ke InfluxDB: {}", e);}if let Err(e) = write\_to\_blockchain(&data).await {eprintln!("❌ Error saat menulis data ke Blockchain: {}", e);}}Err(e) => {eprintln!("Gagal mem-parse JSON: {}", e);}}} |

## 3.1.3 Konfigurasi InfluxDB dan Integrasi

Server dibangun untuk berjalan pada port 7878 dan menerima data dari client secara asynchronous. Ketika koneksi masuk diterima, server membaca data dalam format JSON dan memprosesnya menggunakan pustaka serde\_json untuk diubah menjadi objek Rust terstruktur.

Data sensor tersebut kemudian ditulis ke dalam **InfluxDB** dengan struktur sebagai berikut:

* **Measurement**: monitoring  
  Menjadi nama utama pengelompokan data.
* **Tags** (untuk proses filter & identifikasi):
  + sensor\_id: ID unik sensor, contoh SHT20-PascaPanen-001
  + shipment\_id: Identitas kontainer atau lokasi fisik, contoh KONTAINER-001
  + process\_stage: Tahapan proses, contoh Fermentasi
* **Fields** (nilai aktual yang akan dianalisis):
  + temperature\_celsius: Suhu dalam °C
  + humidity\_percent: Kelembaban dalam %
* **Timestamp**:  
  Diambil dari waktu pembacaan sensor dalam format RFC3339, lalu dikonversi menjadi Unix timestamp (dalam detik) untuk memenuhi format precision=s di InfluxDB v2.

Contoh hasil query di InfluxDB CLI :

Sebuah gambar berisi cuplikan layar, teks, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

## 3.1.4 Dashboard Grafana

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

Dashboard menampilkan dua grafik utama, yaitu:

* Grafik **suhu terhadap waktu** (temperature\_celsius)
* Grafik **kelembaban terhadap waktu** (humidity\_percent)

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

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

Sebuah gambar berisi teks, cuplikan layar, Software multimedia, software

Konten yang dihasilkan AI mungkin salah.

Sebuah gambar berisi cuplikan layar, teks, Software multimedia, software

Konten yang dihasilkan AI mungkin salah.

## Integrasi Blockchain dan Web3

## Sebuah gambar berisi teks, software, Software multimedia, Ikon komputer Konten yang dihasilkan AI mungkin salah.

* **Persiapan Lingkungan Pengembangan Blockchain**

Sebelum melakukan integrasi dengan blockchain, dilakukan beberapa persiapan:

* Instalasi **Node.js** dan **npm** untuk mengelola dependensi.
* Instalasi dan konfigurasi **Hardhat**, yaitu framework pengembangan smart contract berbasis Ethereum.
* Instalasi **MetaMask**, dompet digital berbasis browser untuk menghubungkan DApp dengan jaringan blockchain lokal.
* **Pengembangan Smart Contract Monitoring.sol**

Smart contract ditulis menggunakan **Solidity versi ^0.8.20** dengan struktur sebagai berikut:

* **Struct DataPoint** menyimpan data sensor berupa timestamp, shipment\_id, temperature (integer ×10), dan humidity (integer ×10).
* **Array allDataPoints** untuk menyimpan seluruh data yang tercatat.
* **Variabel owner** sebagai identitas pemilik kontrak untuk keamanan.
* **Fungsi addDataPoint()** dengan modifier onlyOwner untuk menambahkan data ke blockchain.
* **Event DataPointAdded** sebagai notifikasi setiap ada data baru.
* **Fungsi getDataPointCount()** untuk menghitung jumlah data.  
  Kontrak ini dikompilasi menggunakan perintah:

npx hardhat compile

* **Deployment Smart Contract ke Hardhat Local Node**
* Menjalankan jaringan lokal menggunakan:

npx hardhat node

* Deploy dilakukan dengan membuat skrip scripts/deploy.js menggunakan ethers.js.
* Eksekusi skrip dengan:

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

* Setelah berhasil, alamat kontrak (**CONTRACT\_ADDRESS**) dicatat untuk digunakan di sisi Rust dan DApp.
* **Integrasi Smart Contract ke Server TCP (Rust)**

Untuk mencatat data sensor ke blockchain, dilakukan modifikasi di server TCP:

* Menambahkan dependencies:
  + ethers untuk interaksi blockchain.
  + dotenv untuk membaca variabel lingkungan.
  + hex untuk manipulasi data hexadesimal.
* Membuat file .env berisi:
  + PRIVATE\_KEY dari akun #0 Hardhat.
  + CONTRACT\_ADDRESS dari hasil deploy.
* Salin ABI kontrak Monitoring.json ke dalam tcp\_server/abi/.
* Gunakan macro abigen! untuk membuat binding Rust dari smart contract:

abigen!(Monitoring, "./abi/Monitoring.json");

* Membuat fungsi write\_to\_blockchain di server untuk:
  + Terhubung ke Hardhat node (http://127.0.0.1:8545).
  + Memuat wallet menggunakan private key.
  + Membuat instance kontrak dengan ABI dan alamat.
  + Memanggil fungsi addDataPoint dengan data dari sensor.
* Fungsi ini dipanggil di dalam handle\_connection.
* **Pengembangan DApp Sederhana**

Untuk menampilkan data di browser secara langsung dari blockchain:

* Dibuat dua file:
  + index.html → struktur dasar & tampilan tabel.
  + app.js → logika koneksi ke blockchain.
* Menggunakan **Ethers.js** di sisi klien untuk:
  + Menghubungkan ke MetaMask.
  + Membuat instance kontrak.
  + Memanggil getDataPointCount() dan allDataPoints() dari smart contract.
  + Menampilkan data dalam tabel (waktu, lokasi, suhu, kelembaban).
* MetaMask dikonfigurasi ke jaringan Hardhat lokal:
  + Network Name: Hardhat Local
  + RPC URL: http://127.0.0.1:8545
  + Chain ID: 31337
* **Pengujian End-to-End**

Sistem diuji dengan menjalankan seluruh komponen secara bersamaan:

1. Menjalankan **Hardhat Node**.
2. Menjalankan **Rust TCP Server**.
3. Menjalankan **Rust Modbus Client** untuk baca & kirim data sensor.
4. Membuka **DApp di browser** untuk melihat data blockchain secara langsung.

## 3.1.6 Hasil yang dicapai

* Prototipe sistem monitoring suhu dan kelembaban yang berjalan secara real-time dan terintegrasi dari sensor hingga penyimpanan data terpusat dan terdesentralisasi.
* Visualisasi data yang intuitif melalui Grafana (berdasarkan data InfluxDB) dan aplikasi desktop PyQt (berdasarkan data InfluxDB).
* Smart contract (Monitoring.sol) yang berfungsi untuk mencatat data sensor secara transparan dan tidak dapat diubah di blockchain simulasi.
* DApp sederhana yang menunjukkan fungsi pembacaan data dari smart contract, membuktikan transparansi dan aksesibilitas data yang tercatat di blockchain.
* Konektivitas penuh antara sistem instrumentasi (simulasi sensor), server pengolah data (Rust), penyimpanan data (InfluxDB), dan pencatatan di blockchain (Hardhat/Solidity) serta antarmuka Web3 (DApp).

# BAB IV PENGUJIAN & HASIL

## Hasil Tabel Pengujian

## Sebuah gambar berisi teks, cuplikan layar, nomor, Font Konten yang dihasilkan AI mungkin salah.

## Link SPS:

## <https://drive.google.com/file/d/1yxUvvlzYhXjvAJ6raatwTw_Xyryx4EoO/view?usp=sharing>

## Screenshot hasil penyimpanan di influxDB

## Sebuah gambar berisi cuplikan layar, teks, software, Software multimedia Konten yang dihasilkan AI mungkin salah.

## Screenshot real-time dashboard di Grafana

## Sebuah gambar berisi teks, cuplikan layar, Software multimedia, software Konten yang dihasilkan AI mungkin salah.

## Sebuah gambar berisi cuplikan layar, teks, Software multimedia, software Konten yang dihasilkan AI mungkin salah.

## Tampilan monitoring real-time di QT

## Sebuah gambar berisi teks, cuplikan layar, software, nomor Konten yang dihasilkan AI mungkin salah.

## Tampilan Dashboard DApp dan Visualisasi Data Sensor

## Sebuah gambar berisi teks, software, Font, cuplikan layar Konten yang dihasilkan AI mungkin salah.Sebuah gambar berisi teks, software, Ikon komputer, Laman internet Konten yang dihasilkan AI mungkin salah.Sebuah gambar berisi teks, software, Software multimedia, Ikon komputer Konten yang dihasilkan AI mungkin salah.

## BAB V KESIMPULAN DAN REKOMENDASI

## Kesimpulan dan Rekomendasi

## Sistem monitoring suhu dan kelembaban berbasis sensor SHT20 yang terintegrasi dengan platform modern seperti Rust, InfluxDB, Grafana, PyQt, dan blockchain telah terbukti menjadi solusi yang efektif dan andal dalam menjaga kualitas buah-buahan selama proses distribusi. Pemanfaatan sensor yang presisi memungkinkan pengukuran kondisi lingkungan secara real-time, sedangkan integrasi dengan database time-series InfluxDB dan dashboard Grafana menyediakan visibilitas data yang komprehensif bagi operator lapangan dan manajemen.

## Pengembangan aplikasi desktop berbasis PyQt menambahkan fleksibilitas dalam pemantauan lokal, sementara implementasi fitur alarm dan pencatatan historis memperkuat aspek pengendalian mutu. Lebih jauh, penggunaan teknologi blockchain melalui smart contract pada Ethereum memberikan jaminan integritas, transparansi, dan keterlacakan data yang tidak dapat dimanipulasi, sangat penting bagi rantai pasok produk segar yang sensitif.

## Oleh karena itu, sistem ini direkomendasikan untuk diimplementasikan secara nyata pada rantai distribusi buah-buahan maupun produk hortikultura lainnya. Penggabungan teknologi IoT dan blockchain tidak hanya meningkatkan efisiensi dan ketelitian pemantauan, tetapi juga membuka peluang sertifikasi mutu berbasis data dan peningkatan kepercayaan konsumen. Di masa mendatang, pengembangan lebih lanjut dapat difokuskan pada penerapan teknologi edge computing untuk pengolahan lokal dan integrasi machine learning guna prediksi kerusakan produk secara dini.

# DAFTAR PUSTAKA

Oliveira, D. P. de, et al. (2020). "Postharvest technologies for fruits: a review on recent advances." Scientia Horticulturae, vol. 265.

Fadeyibi, A., et al. (2021). "Postharvest management of perishables using low-cost technologies." Journal of Food Process Engineering, vol. 44, no. 9.

Rahman, M. M., et al. (2023). "A secure and low-cost smart Modbus-based temperature and humidity monitoring system for perishable goods." IEEE Access, vol. 11, pp. 44230–44243.

Sensirion AG. (2021). "SHT2x Humidity and Temperature Sensor Datasheet."

Chen, C., et al. (2021). "Temperature and humidity monitoring system for fresh food logistics based on Modbus protocol." Sensors and Actuators A: Physical, vol. 330.

Wang, Y., et al. (2021). "Application of predictive analytics in food logistics based on historical sensor data." IEEE Transactions on Automation Science and Engineering, vol. 18, no. 2.

Ghosh, R., et al. (2022). "Supply chain optimization through IoT-enabled data-driven models." IEEE Transactions on Industrial Informatics, vol. 18, no. 10.

Singh, A., & Bhargava, S. (2020). "Development of embedded GUI using Qt for smart monitoring systems." IEEE Xplore, pp. 1–5.

Park, C., & Lee, K. (2021). "Blockchain-based secure data integrity for IoT-based food logistics monitoring." IEEE Internet of Things Journal, vol. 8, no. 12, pp. 9632–9643.

Al-Bassam, M. (2022). "Decentralized sensor data management in smart supply chains using Ethereum smart contracts." Journal of Blockchain Research, vol. 3, no. 1.

Buterin, V. (2014). "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform." Ethereum Foundation.

**LAMPIRAN**

* **Foto Kegiatan:**

Sebuah gambar berisi orang, teks, Wajah manusia, dalam ruangan

Konten yang dihasilkan AI mungkin salah.

**Sebuah gambar berisi pakaian, orang, computer, dalam ruangan

Konten yang dihasilkan AI mungkin salah.Sebuah gambar berisi orang, Wajah manusia, pakaian, dalam ruangan

Konten yang dihasilkan AI mungkin salah.**

* **Codingan Modbus Client**

|  |
| --- |
| **use tokio::net::TcpStream;**  **use tokio::io::AsyncWriteExt;**  **use tokio::time::{sleep, Duration};**  **use tokio\_modbus::{client::rtu, prelude::\*};**  **use tokio\_serial::SerialPortBuilderExt;**  **use serde::Serialize;**  **use std::error::Error;**  **use chrono::{DateTime, Utc};**  **// Struct JSON baru, sekarang menyertakan shipment\_id**  **#[derive(Serialize)]**  **struct SensorData {**  **timestamp: String,**  **shipment\_id: String, // ID unik untuk setiap kontainer/pengiriman**  **temperature\_celsius: f32,**  **humidity\_percent: f32,**  **}**  **// Fungsi read\_sht20 tidak berubah**  **async fn read\_sht20(slave\_id: u8) -> Result<Vec<u16>, Box<dyn Error>> {**  **let builder = tokio\_serial::new("/dev/ttyUSB0", 9600)**  **.parity(tokio\_serial::Parity::None)**  **.stop\_bits(tokio\_serial::StopBits::One)**  **.data\_bits(tokio\_serial::DataBits::Eight)**  **.timeout(std::time::Duration::from\_secs(1));**  **let port = builder.open\_native\_async()?;**  **let slave = Slave(slave\_id);**  **let response = {**  **let mut ctx = rtu::connect\_slave(port, slave).await?;**  **ctx.read\_input\_registers(1, 2).await?**  **};**  **Ok(response)**  **}**  **// Fungsi send\_data\_to\_server tidak berubah**  **async fn send\_data\_to\_server(data: &SensorData) -> Result<(), Box<dyn Error>> {**  **const SERVER\_ADDR: &str = "127.0.0.1:7878";**  **let json\_payload = serde\_json::to\_string(&data)?;**  **println!("Mengirim data: {}", json\_payload);**  **match TcpStream::connect(SERVER\_ADDR).await {**  **Ok(mut stream) => {**  **stream.write\_all(json\_payload.as\_bytes()).await?;**  **println!("Data berhasil dikirim ke server.");**  **}**  **Err(e) => eprintln!("Gagal terhubung ke server: {}", e),**  **}**  **Ok(())**  **}**  **#[tokio::main]**  **async fn main() -> Result<(), Box<dyn Error>> {**  **// Untuk tujuan tes, kita tetapkan ID pengiriman di sini.**  **// Nanti, ID ini bisa berasal dari file konfigurasi atau argumen program.**  **let shipment\_id\_tes = "KONTAINER-001".to\_string();**  **loop {**  **println!("\nMembaca sensor untuk pengiriman: {}", shipment\_id\_tes);**  **match read\_sht20(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!("Suhu: {:.1} C, Kelembapan: {:.1} %", temp, rh);**    **let now: DateTime<Utc> = Utc::now();**  **// Membuat struct data baru dengan shipment\_id**  **let sensor\_reading = SensorData {**  **timestamp: now.to\_rfc3339(),**  **shipment\_id: shipment\_id\_tes.clone(),**  **temperature\_celsius: temp,**  **humidity\_percent: rh,**  **};**  **if let Err(e) = send\_data\_to\_server(&sensor\_reading).await {**  **eprintln!("Error saat mengirim data: {}", e);**  **}**  **} else {**  **println!("Response sensor tidak valid: {:?}", response);**  **}**  **}**  **Err(e) => eprintln!("Gagal membaca sensor: {}", e),**  **}**  **sleep(Duration::from\_secs(15)).await; // Kita perlambat interval pengiriman**  **}**  **}** |

* **Codingan TCP Server**

|  |
| --- |
| **use tokio::net::{TcpListener, TcpStream};**  **use tokio::io::AsyncReadExt;**  **use std::error::Error;**  **use serde::Deserialize;**  **use chrono::{DateTime, Utc};**  **// --- Integrasi Blockchain ---**  **use ethers::prelude::\*;**  **use std::env;**  **use std::sync::Arc;**  **use dotenv::dotenv;**  **abigen!(Monitoring, "./abi/Monitoring.json");**  **// Struct JSON baru, disesuaikan dengan yang dikirim client**  **#[derive(Deserialize, Debug, Clone)]**  **struct SensorData {**  **timestamp: String,**  **shipment\_id: String, // Menerima shipment\_id**  **temperature\_celsius: f32,**  **humidity\_percent: f32,**  **}**  **// Fungsi write\_to\_blockchain dimodifikasi untuk menerima shipment\_id**  **async fn write\_to\_blockchain(data: &SensorData) -> Result<TxHash, Box<dyn Error>> {**  **let provider = Provider::<Http>::try\_from("http://127.0.0.1:8545")?;**  **let chain\_id = provider.get\_chainid().await?.as\_u64();**  **let private\_key = env::var("PRIVATE\_KEY")?;**  **let wallet = private\_key.parse::<LocalWallet>()?.with\_chain\_id(chain\_id);**  **let client = Arc::new(SignerMiddleware::new(provider, wallet));**  **let contract\_address: Address = env::var("CONTRACT\_ADDRESS")?.parse()?;**  **let contract = Monitoring::new(contract\_address, client);**  **let timestamp\_unix = data.timestamp.parse::<DateTime<Utc>>()?.timestamp();**  **// Gunakan shipment\_id dari data**  **let shipment\_id = data.shipment\_id.clone();**  **let temp = (data.temperature\_celsius \* 10.0) as i128;**  **let hum = (data.humidity\_percent \* 10.0) as i128;**  **println!("Mengirim transaksi ke blockchain untuk pengiriman: {}", shipment\_id);**  **// Panggil fungsi addDataPoint yang baru**  **let tx\_call = contract.add\_data\_point(**  **shipment\_id,**  **U256::from(timestamp\_unix),**  **I256::from(temp),**  **I256::from(hum)**  **);**  **let pending\_tx = tx\_call.send().await?;**  **let receipt = pending\_tx.await?;**  **let tx\_hash = receipt.unwrap().transaction\_hash;**  **println!("✅ Transaksi ke blockchain berhasil dengan hash: {:?}", tx\_hash);**  **Ok(tx\_hash)**  **}**  **// Fungsi write\_to\_influxdb dimodifikasi untuk menggunakan shipment\_id sebagai tag**  **async fn write\_to\_influxdb(data: &SensorData) -> Result<(), Box<dyn Error>> {**  **const INFLUX\_URL: &str = "http://localhost:8086/api/v2/write?org=instrumentasi&bucket=monitoring&precision=s";**  **const INFLUX\_TOKEN: &str = "t3XfMpAywZYjbc0rQ35w1baYYP8RG2ejOtutZPyQoUrE8oAJoQitAal2gSPc3chRWDNOBJ0C2ltWYIUpStf2GQ==";**  **let timestamp\_dt: DateTime<Utc> = data.timestamp.parse()?;**  **let timestamp\_unix = timestamp\_dt.timestamp();**    **// Gunakan shipment\_id sebagai tag di InfluxDB**  **let line\_protocol = format!(**  **"environment\_monitoring,shipment\_id={} temperature\_celsius={},humidity\_percent={} {}",**  **data.shipment\_id, data.temperature\_celsius, data.humidity\_percent, timestamp\_unix**  **);**    **let client = reqwest::Client::new();**  **let response = client.post(INFLUX\_URL)**  **.header("Authorization", format!("Token {}", INFLUX\_TOKEN))**  **.header("Content-Type", "text/plain").body(line\_protocol).send().await?;**  **if response.status().is\_success() {**  **println!("✅ Data berhasil ditulis ke InfluxDB.");**  **} else {**  **eprintln!("❌ Gagal menulis ke InfluxDB: {}", response.text().await?);**  **}**  **Ok(())**  **}**  **// Fungsi handle\_connection tidak perlu banyak perubahan**  **async fn handle\_connection(mut stream: TcpStream) {**  **let mut buffer = Vec::new();**  **if let Err(e) = stream.read\_to\_end(&mut buffer).await {**  **eprintln!("Gagal membaca data dari stream: {}", e);**  **return;**  **}**  **match serde\_json::from\_slice::<SensorData>(&buffer) {**  **Ok(data) => {**  **println!("\n--- Menerima Data Baru ---");**  **println!("{:?}", data);**  **if let Err(e) = write\_to\_influxdb(&data).await {**  **eprintln!("Error saat memproses data ke InfluxDB: {}", e);**  **}**  **if let Err(e) = write\_to\_blockchain(&data).await {**  **eprintln!("❌ Error saat menulis data ke Blockchain: {}", e);**  **}**  **}**  **Err(e) => {**  **eprintln!("Gagal mem-parse JSON: {}", e);**  **}**  **}**  **}**  **#[tokio::main]**  **async fn main() -> Result<(), Box<dyn Error>> {**  **dotenv().ok();**  **let listener = TcpListener::bind("0.0.0.0:7878").await?;**  **println!("TCP Server listening di 0.0.0.0:7878");**  **println!("Menunggu data sensor untuk dikirim ke InfluxDB dan Blockchain...");**  **loop {**  **let (stream, addr) = listener.accept().await?;**  **println!("\nMenerima koneksi dari: {}", addr);**  **tokio::spawn(handle\_connection(stream));**  **}**  **}** |

* **Link repository GitHub :**

<https://github.com/tadashi-student/ISI_MonitoringKelembapanBuahwithBlockchain>