



PROJECT INTERKONEKSI SISTEM INSTRUMENTASI - VI231418

“KakaoTrace: Integrasi IoT dan Web3 untuk Monitoring Fermentasi dan Jual Beli Kakao di Platform Digital”

M SALMAN ALFARISYI
NRP 2042231006

REVA AULIA RAHMAN
NRP 2042231024

TANGKAS BARA PERMATA
NRP 2042231072

Dosen Pembimbing
Ahmad Radhy, S.Si., M.Si
NPP 2022198911049

Program Studi Teknologi Rekayasa Instrumentasi
Departemen Teknik Instrumentasi
Fakultas Vokasi
Institut Teknologi Sepuluh Nopember
Surabaya
2025

I. LATAR BELAKANG

Fermentasi merupakan suatu proses biokimia yang terjadi dalam sistem biologis, di mana berlangsung reaksi oksidasi-reduksi yang menghasilkan energi. Dalam proses ini, baik donor maupun akseptor elektronnya adalah senyawa organik, sehingga berbeda dengan respirasi seluler yang melibatkan oksigen sebagai akseptor elektron. Umumnya, senyawa organik yang digunakan sebagai substrat dalam fermentasi adalah karbohidrat, khususnya gula seperti glukosa. Gula tersebut akan mengalami serangkaian reaksi biokimia yang dikatalisis oleh enzim tertentu, sehingga mengalami perubahan struktur kimia menjadi senyawa organik lain seperti alkohol, asam laktat, atau asam asetat, tergantung pada jenis mikroorganisme dan kondisi fermentasinya. Proses ini memungkinkan mikroorganisme memperoleh energi dalam kondisi anaerob, yaitu tanpa memerlukan oksigen bebas. (Endang Kwartiningsih, 2005)

Fermentasi kakao adalah metode biologis untuk menurunkan kadar senyawa tertentu dan meningkatkan mutu. Dalam penelitian ini, bakteri asam laktat (BAL) diisolasi dari sumber organik, menghasilkan 27 isolat awal, lalu diseleksi melalui uji biokimia hingga diperoleh 13 isolat BAL. Dari karakterisasi lanjutan, dipilih tiga isolat unggul dengan aktivitas enzimatik berbeda. Fermentasi dilakukan menggunakan campuran ketiga isolat tersebut dengan variasi waktu inkubasi 8, 16, dan 24 jam dalam tiga ulangan. Sampel tanpa fermentasi digunakan sebagai kontrol pembanding. (Doni Usman, 2015)

Fermentasi kakao yang dilakukan secara tepat akan memicu transformasi penting pada biji kakao, seperti peningkatan kadar karbon dioksida, penurunan kadar air dan oksigen, serta perubahan tingkat pH, yang secara keseluruhan dapat meningkatkan kualitas cita. Namun, beberapa hasil fermentasi pada berbagai daerah belum konsisten di setiap proses pengolahan akibat tidak adanya pemantauan suhu dan pH menggunakan alat ukur yang memadai. Hal ini menyebabkan kualitas biji kakao yang dihasilkan kurang optimal dan berdampak pada nilai jualnya. Suhu sendiri memiliki peran penting dalam menentukan laju fermentasi kakao. Untuk mengatasi kendala tersebut, digunakan pendekatan teknologi berbasis software Ubuntu dengan dukungan seperti InfluxDB dan Grafana, yang memungkinkan proses pemantauan suhu dan pH secara real-time dan akurat, sehingga mempermudah pengawasan fermentasi dan meningkatkan konsistensi mutu produk. (Bagus Irfanzah Arda Nugraha, 2024)

II. RUMUSAN MASALAH

1. Tidak adanya sistem monitoring real-time berbasis sensor di gudang fermentasi.
2. Tidak adanya penyimpanan data historis suhu dan kelembaban.
3. Perlunya visualisasi data untuk pengambilan keputusan oleh petani.

III. TUJUAN

1. Membuat sistem monitoring suhu dan kelembaban berbasis sensor SHT20.
2. Mengirimkan data secara real-time ke TCP Server dan menyimpannya di InfluxDB.
3. Menyediakan dashboard real-time menggunakan Grafana.

IV. METODOLOGI

4.1 Design Arsitektur

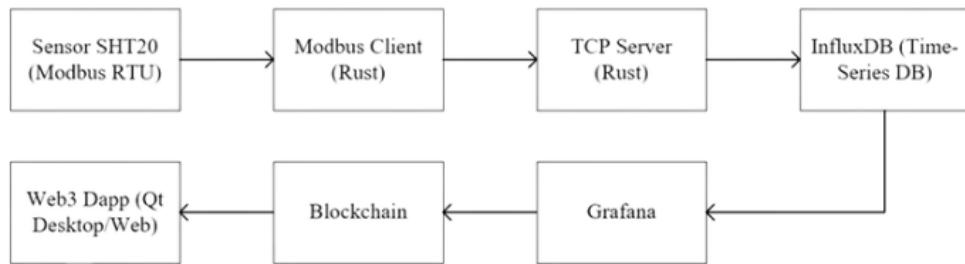


Diagram tersebut menggambarkan arsitektur sistem monitoring suhu dan kelembapan berbasis sensor SHT20 yang terintegrasi dengan teknologi time-series database, visualisasi real-time, dan blockchain/Web3. Berikut penjelasannya:

Sensor SHT20 berfungsi sebagai perangkat pengukur suhu dan kelembapan di lingkungan fermentasi. Sensor ini berkomunikasi melalui protokol Modbus RTU, yang dibaca oleh Modbus Client yang dibangun menggunakan bahasa pemrograman Rust. Data hasil pembacaan dari sensor kemudian dikirimkan ke TCP Server, juga dikembangkan dalam Rust, yang berperan menerima dan memproses data sensor secara jaringan.

Data yang diterima oleh TCP Server kemudian disimpan dalam InfluxDB, yaitu database time-series yang dirancang untuk menyimpan data berdasarkan waktu, sangat cocok untuk data lingkungan yang berubah secara periodik. Data ini selanjutnya divisualisasikan menggunakan Grafana, yang terhubung langsung ke InfluxDB dan menampilkan grafik suhu dan kelembapan secara real-time.

Selanjutnya, data dari Grafana (yang sudah tervisualisasikan) dapat dicatat ke dalam sistem blockchain, dalam bentuk hash sebagai bukti otentikasi bahwa proses fermentasi telah berlangsung sesuai standar. Hash ini dicatat ke blockchain untuk menjamin transparansi dan keaslian data. Terakhir, Web3 DApp berbasis Qt Desktop/Web berfungsi sebagai antarmuka pengguna untuk melihat hasil monitoring dan verifikasi hash data dari blockchain.

4.2 Deskripsi Komponen

a. Sensor SHT20



Gambar 4.1 Sensor SHT20

Sensor suhu dan kelembapan SHT20 buatan Sensirion telah menjadi acuan dalam industri berkat desainnya yang ringkas dan tingkat integrasi yang tinggi. Sensor ini menggabungkan sensor kelembapan kapasitif, sensor suhu tipe band-gap, serta sirkuit analog dan digital khusus dalam satu chip berbasis teknologi CMOSens®. SHT20 mampu mengukur kelembapan relatif dalam rentang 0 hingga 100 %RH dengan akurasi ± 3.0 %RH, serta suhu pada kisaran -40 hingga 125 °C dengan akurasi ± 0.3 °C (Anggara Trisna Nugraha, 2022)

b. Modbus Client (Rust)

MODBUS/TCP adalah protokol komunikasi berbasis TCP/IP yang merupakan varian dari MODBUS, digunakan untuk pengawasan dan kontrol perangkat otomasi melalui jaringan Ethernet. Protokol ini mendukung konektivitas antar perangkat seperti PLC, modul I/O, dan gateway, serta telah menjadi standar de facto di industri otomasi. Selain fungsinya yang mendukung interoperabilitas antar perangkat, sebagian fitur MODBUS merupakan peninggalan dari penggunaannya sebagai protokol pemrograman PLC.

c. TCP Server (Rust)

TCP Server dirancang untuk mengalihkan beban pemrosesan jaringan dari host yang menjalankan aplikasi server internet. Konsep utamanya adalah menjalankan proses TCP/IP pada prosesor, node, atau perangkat khusus (disebut TCP server), dengan komunikasi yang ringan dan tidak mengganggu antara perangkat tersebut dan host aplikasi. Pendekatan ini bertujuan meningkatkan efisiensi sistem secara keseluruhan. (Rangarajan, 2002)

d. InfluxDB



Gambar 4.2 InfluxDB

InfluxDB merupakan sistem basis data deret waktu (time series database) bersifat open-source tanpa skema tetap (schemaless) yang dikembangkan oleh InfluxData. Sistem ini ditulis menggunakan bahasa pemrograman Go dan dirancang secara khusus untuk mengelola data deret waktu secara efisien. InfluxDB menyediakan bahasa kueri yang menyerupai SQL untuk mempermudah proses pengambilan dan pengolahan data. Versi open-source dari InfluxDB dikenal dengan sebutan TICK Stack, yang mencakup berbagai layanan pendukung dan dapat dijalankan baik pada lingkungan cloud maupun secara lokal dalam satu node. Sementara itu, versi komersial seperti InfluxEnterprise (IE) dan InfluxCloud (IC) menawarkan fitur tambahan seperti ketersediaan tinggi (high availability), kemampuan skalabilitas, pencadangan dan pemulihan data, serta opsi penerapan baik di lingkungan lokal (IE) maupun cloud (IC). (Syehda Noor Zehra Naqvi, 2017)

e. Grafana



Gambar 4.3 Grafana

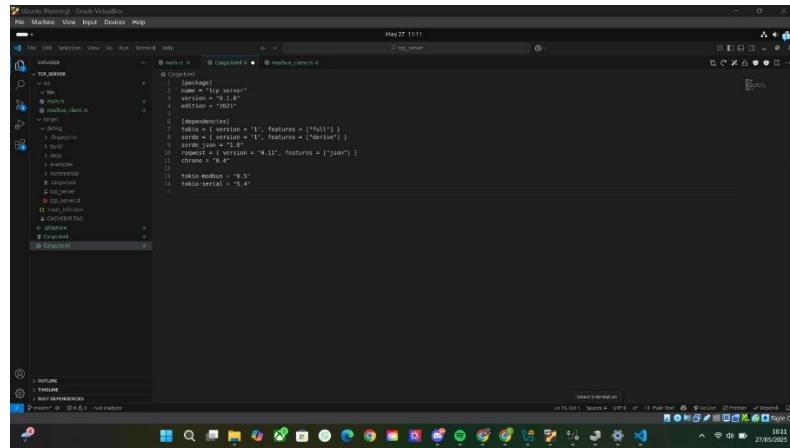
Grafana merupakan alat open-source yang digunakan untuk analisis data, pengambilan metrik, serta pemantauan aplikasi melalui dasbor yang dapat dikonfigurasi sesuai kebutuhan. Karena sifatnya yang terbuka, Grafana memungkinkan pengembangan plugin khusus untuk integrasi dengan berbagai sumber data.

Secara teknis, Grafana mendukung analisis deret waktu (time series analytics), yang berguna dalam mempelajari, menganalisis, dan memantau data dalam rentang waktu tertentu. Teknologi ini memberikan informasi kontekstual yang relevan, seperti perilaku pengguna, kinerja aplikasi, frekuensi kesalahan yang terjadi selama eksekusi atau dalam lingkungan pra-produksi, jenis permasalahan yang muncul, serta kondisi yang

menyertainya. (Levente Manases, 2022)

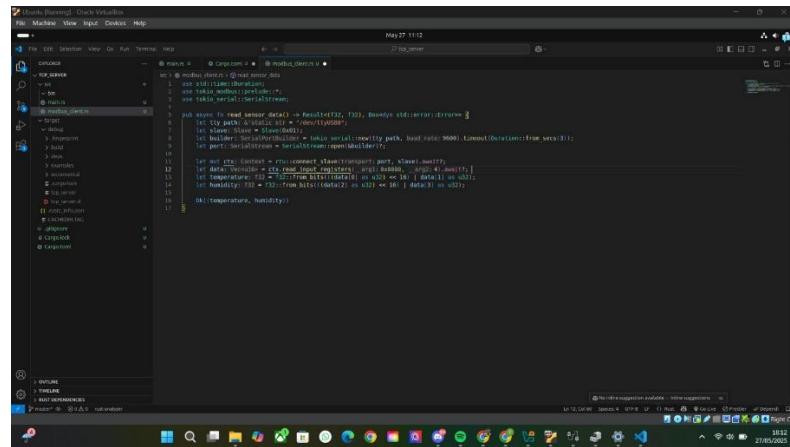
V. IMPLEMENTASI DAN KODE PROGRAM

1. Kode Rust Modbus Client



```
modbus_client$ cd modbus_client
modbus_client$ tree
modbus_client/
├──Cargo.toml
├──Cargo.lock
└──src
    └──main.rs
modbus_client$ cat src/main.rs
use std::io;
use modbus_client::read_sensor;
use tokio::serial::SerialPort;
use tokio::time::Duration;
use tokio::net::SerialStream;
modbus_client$
```

Gambar 5.1 Kode Rust Modbus Client



```
modbus_client$ cd modbus_client
modbus_client$ tree
modbus_client/
├──Cargo.toml
├──Cargo.lock
└──src
    └──modbus_client.rs
modbus_client$ cat src/modbus_client.rs
pub async fn read_sensor(data) -> Result<(f32, f32), Box = city.read_register(Register::Humi, 0x00..0x01).await?;
    let humidity: f32 = ((data[0] as u16) + (data[1] as u16) * 256.0) / 100.0;
    let data: Vec = city.read_register(Register::Temp, 0x00..0x01).await?;
    let temperature: f32 = ((data[0] as u16) + (data[1] as u16) * 256.0) / 100.0;
    Ok((temperature, humidity))
}
modbus_client$
```

Gambar 5.2 Kode Rust Modbus Client

Kedua gambar di atas menunjukkan implementasi perangkat lunak menggunakan bahasa pemrograman Rust yang bertujuan untuk membaca data sensor suhu dan kelembapan melalui protokol komunikasi Modbus RTU. Gambar pertama menampilkan bagian kode sumber dalam berkas main.rs pada modul *modbus_client*, yang berisi fungsi *read_sensor*. Fungsi ini bertugas melakukan inisialisasi koneksi serial menggunakan pustaka *tokio-serial*, kemudian mengirim permintaan pembacaan data register ke perangkat sensor dengan memanfaatkan pustaka *tokio-modbus*.

Data yang diterima dari sensor kemudian diolah untuk memperoleh nilai suhu dan kelembapan dalam satuan yang sesuai. Gambar kedua merupakan tampilan berkas *Cargo.toml* yang berfungsi untuk mendeklarasikan metadata proyek serta dependensi pustaka eksternal seperti *tokio*, *serde*, *tokio-modbus*, *tokio-serial*, dan *chrono*, yang seluruhnya diperlukan dalam pengembangan aplikasi asinkron dan pengolahan data serial dalam proyek ini. Implementasi ini mendemonstrasikan pendekatan terstruktur dalam pengembangan sistem akuisisi data berbasis protokol Modbus pada perangkat tertanam.

2. Kode Rust TCP Server

```

modbus::Client::read_sensor_data().await;
ok((temperature, humidity) => {
    println!("Suhu: ({}, {} °C, Kelembaban: ({}-%)", temperature, humidity);
    let influx_payload = format!(
        "suhu,range:temperature[{}],humidity[{}] ({}",
        temperature,
        humidity,
        chrono::Utc::now().timestamp_nano() / 1_000_000
    );
    let response: Response = client.post("http://localhost:8080/api/v2/writer/http/bucket-server/monitor/precision") RequestBuilder
        .header("Authorization", value::Token("9e3d5f8fb4a9a26f0c95103400a0280a"))
        .body(influx_payload).RequestBuilder
        .send().map_err(|_| error::RequestError::NetworkError).await;
    if response.status().is_success() {
        println!("Data berhasil dikirim ke InfluxDB");
    } else {
        eprintln!("X Gagal Kirim ke InfluxDB: ({})", response.text().await);
    }
}).err().map(|_| error::ReadModbusError::InfluxDbError).await;
}

```

Gambar 5.3 Kode Rust TCP Server 1

```

modbus::Client::read_sensor_data().await;
ok((temperature, humidity) => {
    println!("Suhu: ({}, {} °C, Kelembaban: ({}-%)", temperature, humidity);
    let influx_payload = format!(
        "suhu,range:temperature[{}],humidity[{}] ({}",
        temperature,
        humidity,
        chrono::Utc::now().timestamp_nano() / 1_000_000
    );
    let response: Response = client.post("http://localhost:8080/api/v2/writer/http/bucket-server/monitor/precision") RequestBuilder
        .header("Authorization", value::Token("9e3d5f8fb4a9a26f0c95103400a0280a"))
        .body(influx_payload).RequestBuilder
        .send().map_err(|_| error::RequestError::NetworkError).await;
    if response.status().is_success() {
        println!("Data berhasil dikirim ke InfluxDB");
    } else {
        eprintln!("X Gagal Kirim ke InfluxDB: ({})", response.text().await);
    }
}).err().map(|_| error::ReadModbusError::InfluxDbError).await;
}

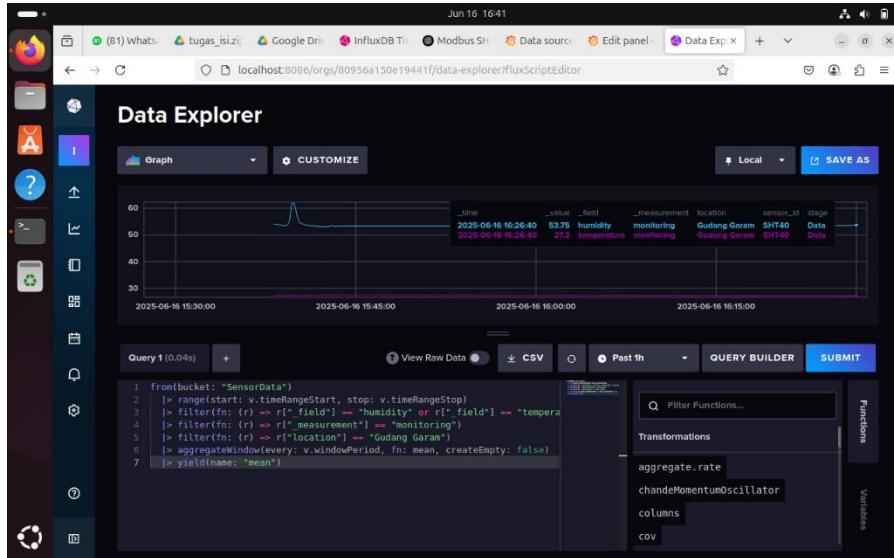
```

Gambar 5.4 Kode Rust TCP Server 2

Kedua gambar tersebut menampilkan struktur program utama yang berfungsi sebagai penghubung antara sensor Modbus RTU dan platform IoT ThingsBoard. Program ditulis dalam bahasa Rust dan memanfaatkan paradigma pemrograman asinkron menggunakan pustaka tokio. Secara berkala, program membaca data suhu dan kelembapan dari sensor melalui fungsi *read_sensor*, lalu memformat hasil pembacaan tersebut ke dalam bentuk data JSON. Data tersebut selanjutnya dikirimkan ke ThingsBoard melalui protokol HTTP POST menggunakan pustaka reqwest, yang ditujukan pada endpoint tertentu dengan autentikasi menggunakan token.

Program juga dilengkapi dengan penanganan kondisi kesalahan seperti kegagalan pembacaan sensor maupun kegagalan pengiriman data, sehingga mendukung kestabilan dan keandalan sistem. Dengan pendekatan ini, integrasi antara perangkat fisik dan sistem pemantauan daring dapat dilakukan secara efisien dan terstruktur.

3. Hasil Query InfluxDB



Gambar 5.5 Hasil Query InfluxDB

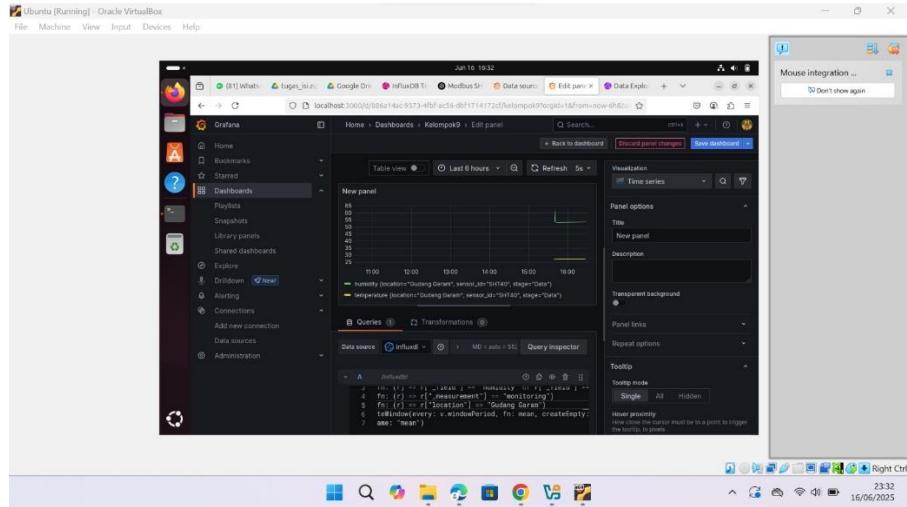
Antarmuka *Data Explorer* pada InfluxDB digunakan untuk menampilkan dan menganalisis data deret waktu yang diperoleh dari hasil akuisisi sensor suhu dan kelembapan. Query yang ditulis menggunakan bahasa Flux melakukan pemanggilan data dari *bucket* bernama *SensorData*, dengan penerapan beberapa tahap penyaringan terhadap *field* suhu dan kelembapan, *measurement* bernama *monitoring*, serta lokasi pengukuran yang ditentukan, yaitu "*Gudang Garam*".

Selanjutnya, data yang telah difilter dianalisis menggunakan fungsi *aggregateWindow* untuk menghitung nilai rata-rata dalam interval waktu tertentu. Hasil pemrosesan ditampilkan dalam bentuk grafik yang merepresentasikan tren perubahan nilai suhu dan kelembapan terhadap waktu. Visualisasi ini memungkinkan proses pemantauan kondisi lingkungan dilakukan secara real-time maupun historis, sehingga mendukung pengambilan keputusan berbasis data secara tepat dan efisien.

Query ini terdiri dari beberapa tahap penyaringan (*filtering*) untuk memfokuskan data:

- Mengambil data dalam rentang waktu tertentu menggunakan *range()*.
- Menyaring data berdasarkan *field* yang hanya berisi *humidity* dan *temperature*.
- Menyaring berdasarkan *measurement* yang diberi nama *monitoring*.
- Menyaring data berdasarkan *location* yaitu "*Gudang Garam*".

4. Grafik Time Series Grafana



Gambar tersebut menampilkan antarmuka *dashboard* Grafana yang digunakan untuk memvisualisasikan data deret waktu (*time-series*) yang bersumber dari InfluxDB. Visualisasi ini menyajikan grafik parameter kelembapan yang diperoleh dari sensor SHT40, yang merupakan bagian dari sistem pemantauan lingkungan berbasis IoT di lokasi “Gudang Garam”.

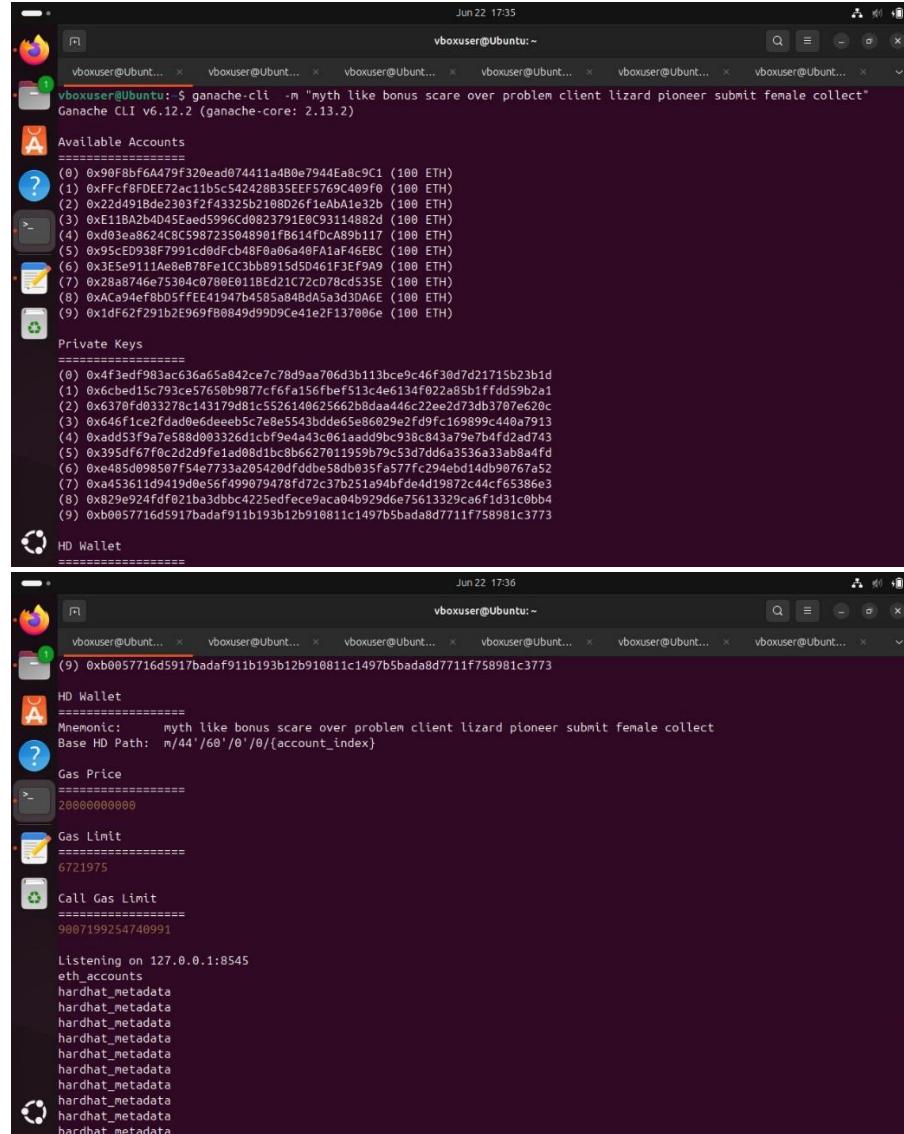
Grafana dalam hal ini menjalankan query menggunakan bahasa Flux, dengan struktur query sebagai berikut:

- | |
|--|
| • Mengambil data berdasarkan rentang waktu tertentu melalui fungsi <code>range()</code> . |
| • Melakukan penyaringan (<i>filtering</i>) terhadap field yang hanya berisi <code>humidity</code> . |
| • Menyaring data berdasarkan measurement bernama <code>monitoring</code> . |
| • Menyaring data berdasarkan lokasi (location) yaitu “Gudang Garam”. |
| • Melakukan agregasi data menggunakan fungsi <code>aggregateWindow()</code> dengan metode <code>mean</code> untuk menghitung rata-rata kelembapan dalam suatu interval waktu tertentu, serta menetapkan <code>createEmpty: false</code> untuk menghindari visualisasi data kosong. |
| • Menampilkan hasil akhir dengan menggunakan fungsi <code>yield()</code> dan memberi label “ <code>mean</code> ”. |

Grafik yang ditampilkan pada panel Grafana menunjukkan dinamika perubahan nilai kelembapan dalam kurun waktu yang ditentukan. Visualisasi ini memungkinkan pengguna untuk mengamati tren lingkungan secara aktual maupun historis secara interaktif, mendukung pengambilan keputusan teknis yang lebih cepat dan tepat berdasarkan data yang tervalidasi dari InfluxDB.

VI. STEP BY STEP

1) Connect to Wallet



```
vboxuser@Ubuntu: ~$ ganache-cli -m "myth like bonus scare over problem client lizard pioneer submit female collect"
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
=====
(0) 0x90f8bf64a79f329ead074411a4B0e7944Ea8c9C1 (100 ETH)
(1) 0xFcf8f5FDEE72ac11b5c542428835EEF5769c489f0 (100 ETH)
(2) 0x2d4918de2303ff3325b218026feab1e32b (100 ETH)
(3) 0x11BA2b4045Eaed5996Cd08237910C93114982d (100 ETH)
(4) 0xd03ea8624C8C987235048901fb614fDcA89b117 (100 ETH)
(5) 0x95cED938F7991cd0Fcba8f0a06a0fAlaf46EB (100 ETH)
(6) 0x1E5e9111Ae0eB78Fe1C3bb8915d5D46f3E9f9A9 (100 ETH)
(7) 0x28a8746e5304c7880E011Ed21C72cD78cd535E (100 ETH)
(8) 0xAc094eF8b05fEE41947b4585a04Bd5a3d30A6E (100 ETH)
(9) 0x1df62f291b2e969FB0849d99dCe41e2f13700e (100 ETH)

Private Keys
=====
(0) 0x4f3edf983ac636a65a842ce7c78d9aa706d3b113bc9c46f30d7d21715b23b1d
(1) 0x6cbed15c793ce57650b9877cf6fa156fbeF513c4e6134f022a85b1ffdd59b2a1
(2) 0x370fd033278c143179d81c5526140625662b8daa446c22ee2d73db3797e620c
(3) 0x46f1ce2fdad0e6deeeb5c7ebe5543dd6e5e86029e2fd9fc169899c440a7913
(4) 0xadd53f9a7e588d093326d1cb9e4a43c061aad9hc938c843a79e7b4fd2ad743
(5) 0x395d67ffcc2d2d9fe1ad0801bc8b627011959b79c53d7dd6a3536a33ab8a4fd
(6) 0xe485d0985807f54e7733a205420dfddde58db035fa577fc294ebd14db98767a52
(7) 0xa453611d941960e56f499079478fd72c37b251a94bfde4d199872c44c6538663
(8) 0xb29e924fdf821ba3dbc4225edfce9aca04b929d6e75613329ca6f1d31c0bb4
(9) 0xb0057716d5917badaf911b193b12b910811c1497b5badabd7711f758981c3773

HD Wallet
=====
```

```
vboxuser@Ubuntu: ~$ mnemonics: myth like bonus scare over problem client lizard pioneer submit female collect
Base HD Path: m/44'/60'/0' /account_index

Gas Price
=====
200000000000

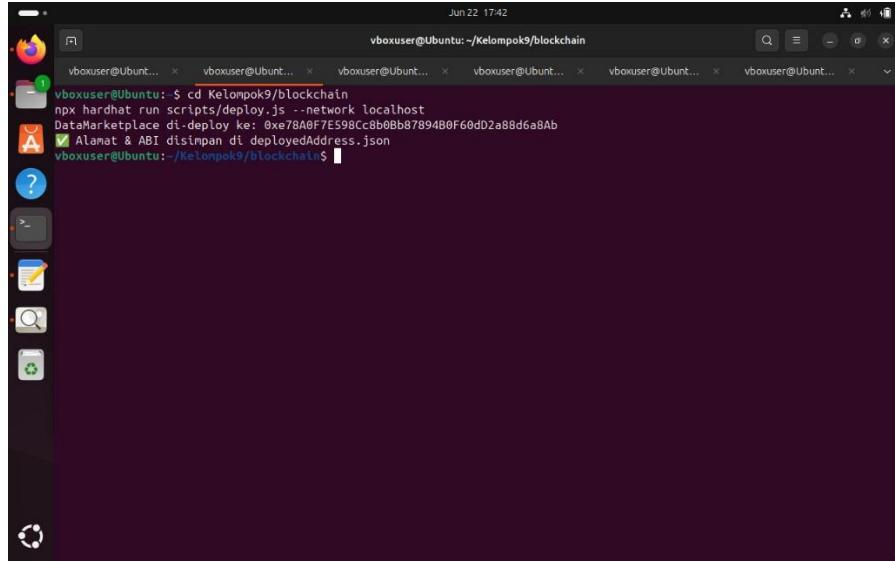
Gas Limit
=====
6721975

Call Gas Limit
=====
9007199254740991

Listening on 127.0.0.1:8545
eth_accounts
hardhat_metadata
```

Tahapan awal menampilkan proses koneksi sistem ke dompet digital (wallet), yang digunakan untuk autentikasi pengguna di jaringan Web3. Wallet ini memungkinkan interaksi dengan blockchain, seperti menandatangani transaksi dan mencatat hash data sensor secara aman dan terverifikasi. Koneksi ini menjadi dasar untuk memastikan akses ke fitur Web3 hanya dilakukan oleh pengguna yang sah.

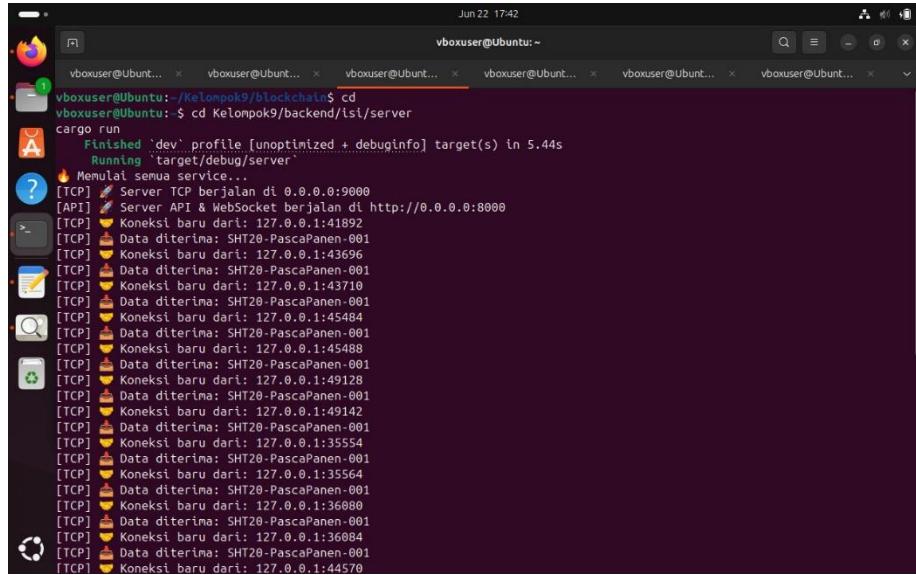
2) Connect to Blockchain



```
vboxuser@Ubuntu: ~$ cd Kelompok9/blockchain
vboxuser@Ubuntu: ~/Kelompok9/blockchain$ npx hardhat run scripts/deploy.js --network localhost
DataMarketplace deployed ke: 0xe78A0f7E598Ccb6Bb87894B0F60d2a88d6a8Ab
Alamat & ABI disimpan di deployedAddress.json
vboxuser@Ubuntu: ~/Kelompok9/blockchain$
```

Selanjutnya ditampilkan proses koneksi ke jaringan blockchain setelah wallet terhubung. Sistem memastikan bahwa node blockchain aktif dan siap menerima transaksi dari aplikasi. Koneksi ini penting untuk menjamin bahwa data sensor yang dikirim nantinya dapat disimpan dalam bentuk hash secara permanen dan tidak dapat dimanipulasi, mendukung prinsip transparansi dalam sistem.

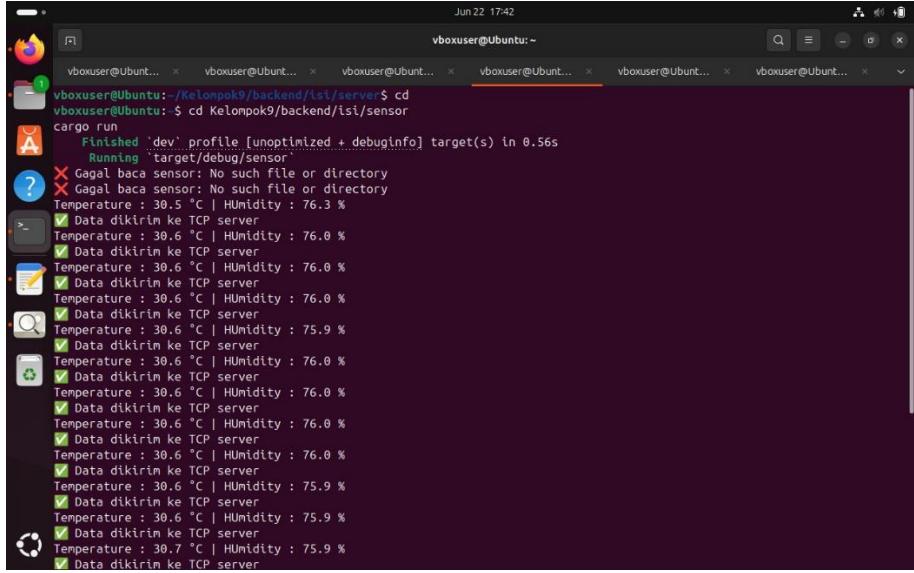
3) Connect to TCP Server dan data diterima



```
vboxuser@Ubuntu: ~/Kelompok9/blockchain$ cd
vboxuser@Ubuntu: ~$ cd Kelompok9/backend/lst/server
Cargo run
  Finished `dev` profile [unoptimized + debuginfo] target(s) in 5.44s
    Running `target/debug/server`
🔥  Mulai servis...
[TCP] 🌐 Server TCP berjalan di 0.0.0.0:9000
[API] 🌐 Server API & WebSocket berjalan di http://0.0.0.0:8000
[TCP] 💫 Koneksi baru dari: 127.0.0.1:41892
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:43696
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:43710
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:45484
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:45488
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:49128
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:49142
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:35554
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:35564
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:36080
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:36084
[TCP] 🎨 Data diterima: SHT20-PascaPanen-001
[TCP] 💫 Koneksi baru dari: 127.0.0.1:44570
```

Berikutnya adalah tahap di mana sistem telah berhasil terhubung ke TCP Server dan mulai menerima data sensor. Data yang diterima biasanya berbentuk JSON dan berisi informasi penting seperti suhu, kelembaban, waktu pencatatan, serta ID sensor. Ini merupakan bukti bahwa komunikasi antara sensor dan server berjalan baik dan siap untuk proses pencatatan lebih lanjut.

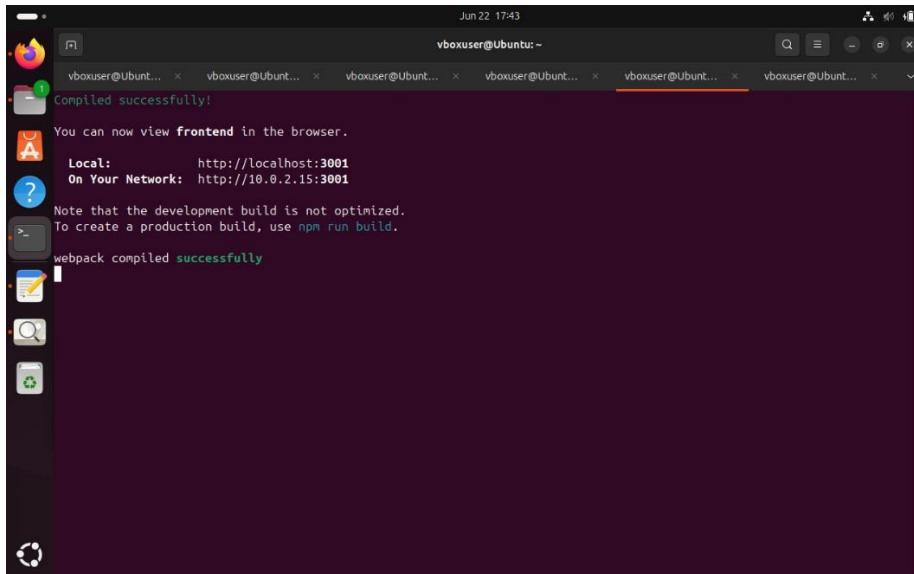
4) Connect to Sensor



```
Jun 22 17:42
vboxuser@Ubuntu:~/Kelompok9/backend/isl/server$ cd Kelompok9/backend/isl/sensor
cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.56s
        Running `target/debug/sensor`
Gagal baca sensor: No such file or directory
Gagal baca sensor: No such file or directory
Temperature : 30.5 °C | HUmidity : 76.3 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 75.9 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 76.0 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 75.9 %
Data dikirim ke TCP server
Temperature : 30.6 °C | HUmidity : 75.9 %
Data dikirim ke TCP server
Temperature : 30.7 °C | HUmidity : 75.9 %
Data dikirim ke TCP server
```

Tahapan berikut menunjukkan bahwa sensor berhasil terhubung dengan sistem. Koneksi ini menandakan bahwa perangkat sensor bekerja secara fungsional dan mampu mengirimkan data secara periodik. Kondisi ini memastikan bahwa semua data yang akan diproses berasal dari sumber valid yang aktif di lapangan.

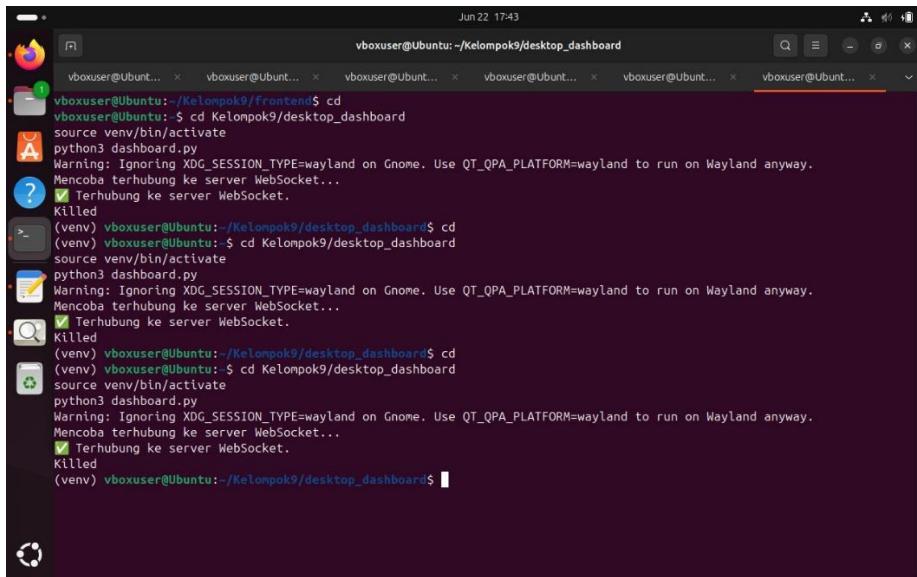
5) Connect to Frontend (lokal)



```
Jun 22 17:43
vboxuser@Ubuntu:~/Kelompok9/frontend$ npm run build
Compiled successfully!
You can now view frontend in the browser.
Local:          http://localhost:3001
Or Your Network: http://10.0.2.15:3001
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```

Selanjutnya ditampilkan antarmuka lokal (frontend) berbasis aplikasi desktop yang berhasil terkoneksi ke sistem. Tampilan ini memperlihatkan data sensor secara real-time dan menjadi alat bantu bagi pengguna di sisi lokal untuk memantau kondisi lingkungan atau proses tanpa perlu akses ke internet atau jaringan blockchain.

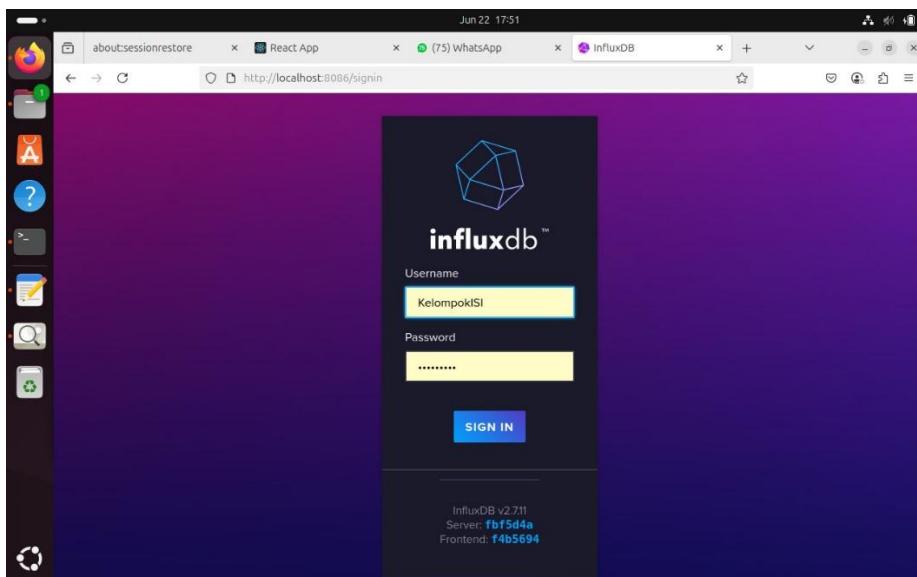
6) Connect to Dashboard PYQT5

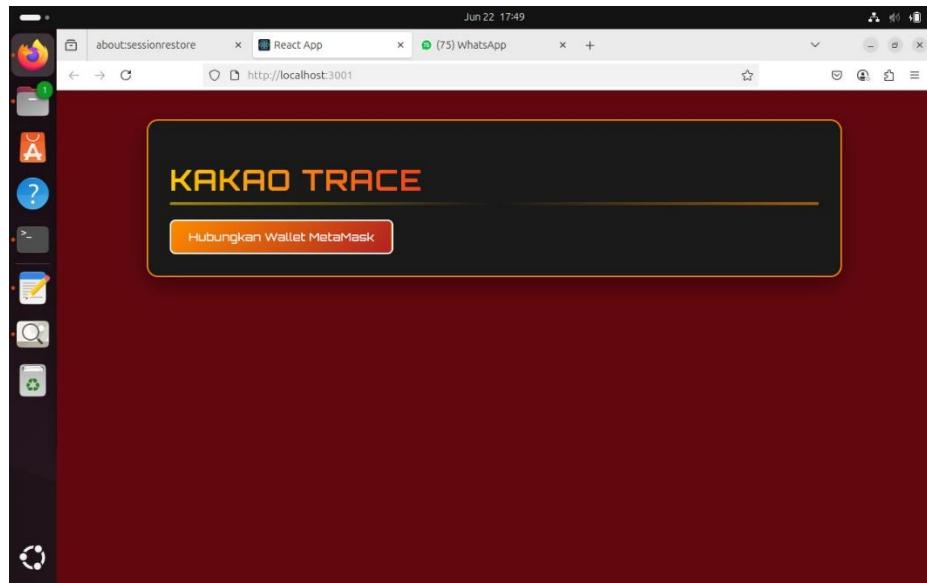


```
Jun 22 17:43
vboxuser@Ubuntu:~/Kelompok9/Desktop_Dashboard$ cd
vboxuser@Ubuntu: $ cd Kelompok9/Desktop_Dashboard
source venv/bin/activate
python3 dashboard.py
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
Mencoba terhubung ke server WebSocket...
Terhubung ke server WebSocket.
Killed
(venv) vboxuser@Ubuntu:~/Kelompok9/Desktop_Dashboard$ cd
(venv) vboxuser@Ubuntu: $ cd Kelompok9/Desktop_Dashboard
source venv/bin/activate
python3 dashboard.py
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
Mencoba terhubung ke server WebSocket...
Terhubung ke server WebSocket.
Killed
(venv) vboxuser@Ubuntu:~/Kelompok9/Desktop_Dashboard$ cd
(venv) vboxuser@Ubuntu: $ cd Kelompok9/Desktop_Dashboard
source venv/bin/activate
python3 dashboard.py
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
Terhubung ke server WebSocket.
Killed
(venv) vboxuser@Ubuntu:~/Kelompok9/Desktop_Dashboard$
```

Tahap ini memperlihatkan dashboard berbasis pyqt5 yang telah terkoneksi dan dapat digunakan untuk menampilkan data-data hasil monitoring yang telah dicatat di blockchain. Dashboard ini memudahkan konsumen atau pengguna akhir untuk melakukan verifikasi independen terhadap data fermentasi atau proses lainnya yang tercatat di sistem.

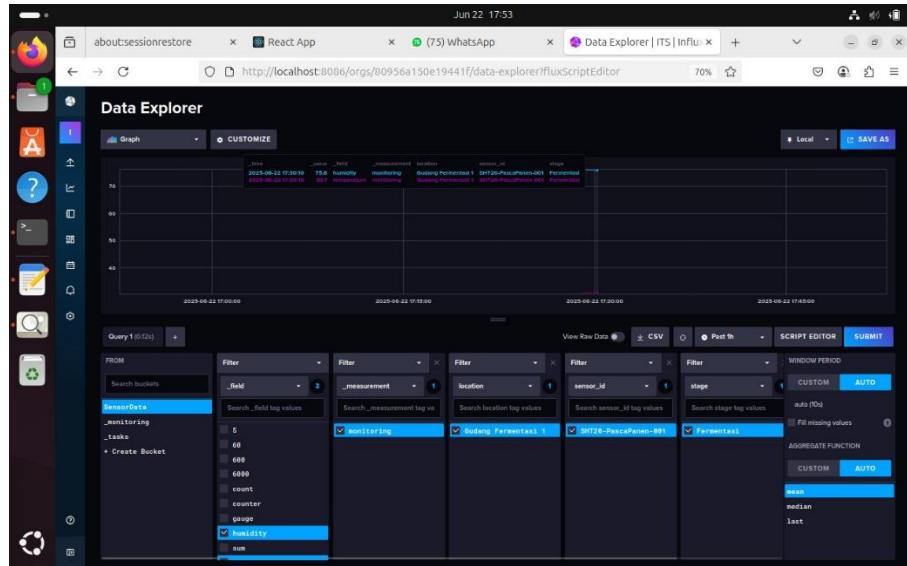
7) Tampilan awal Web3





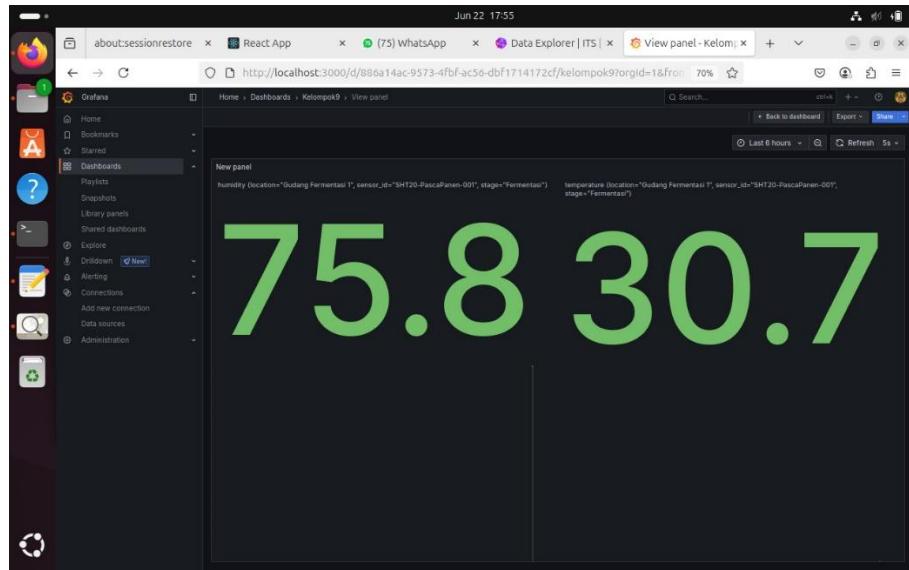
Ditampilkan antarmuka awal dari aplikasi Web3, di mana pengguna dapat melihat informasi seperti identitas wallet, hash data yang tercatat, dan menu navigasi untuk melakukan verifikasi. Tampilan ini menyederhanakan proses audit data oleh pihak eksternal, sehingga setiap data lingkungan yang direkam dapat ditelusuri ke sumber aslinya.

8) Tampilan Awal dan Grafik InfluxDB



Berikutnya adalah tampilan awal grafik dari InfluxDB yang menunjukkan bahwa data suhu dan kelembaban berhasil disimpan sebagai time-series. Grafik ini mencerminkan bagaimana data direkam secara berkala dan membentuk tren, yang berguna untuk analisis performa proses seperti fermentasi atau penyimpanan.

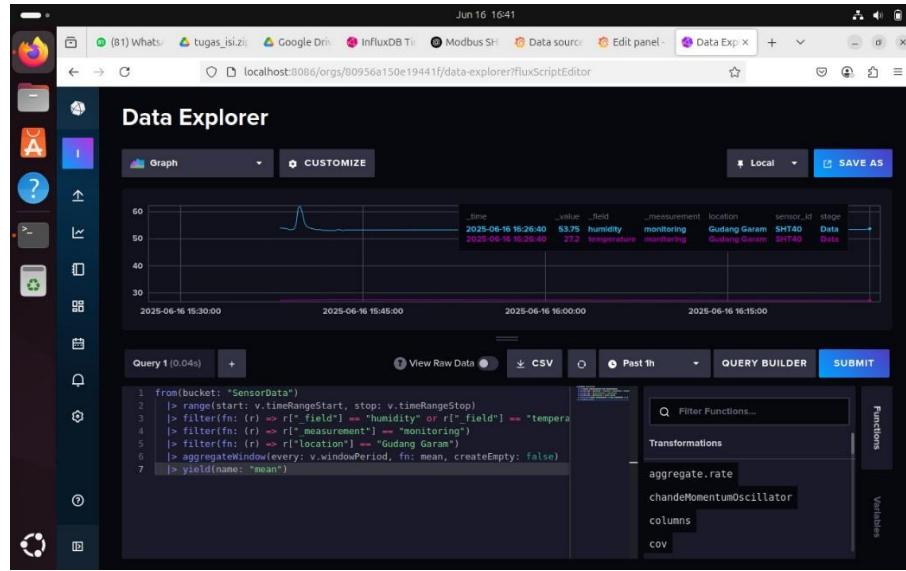
9) Tampilan di Grafana



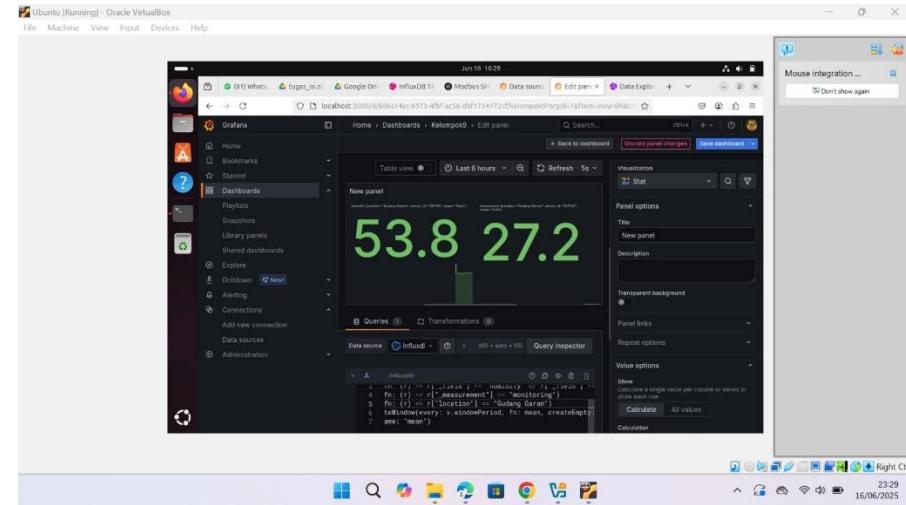
Terakhir, visualisasi dari Grafana menunjukkan data sensor yang telah tersimpan di InfluxDB dalam bentuk grafik interaktif. Panel-panel grafik yang ditampilkan memudahkan pengguna untuk memantau nilai suhu dan kelembaban secara real-time serta mengatur batas peringatan apabila nilai melebihi ambang aman proses.

VII. PENGUJIAN DAN HASIL

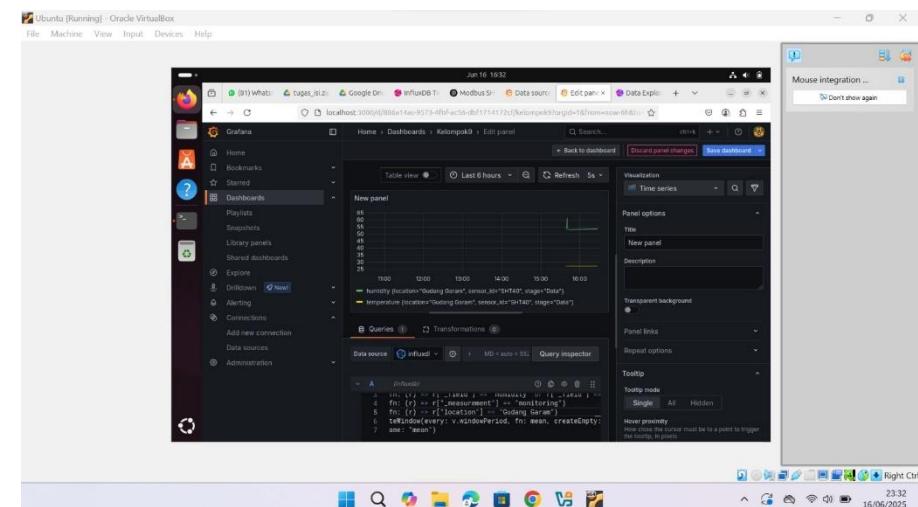
a. Grafik Real Time & Query InfluxDB



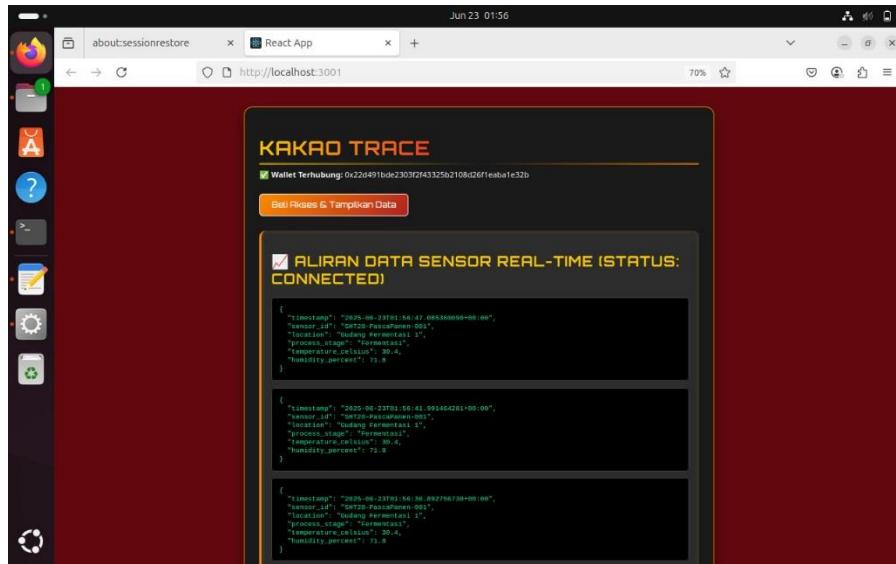
b. Stat Grafana



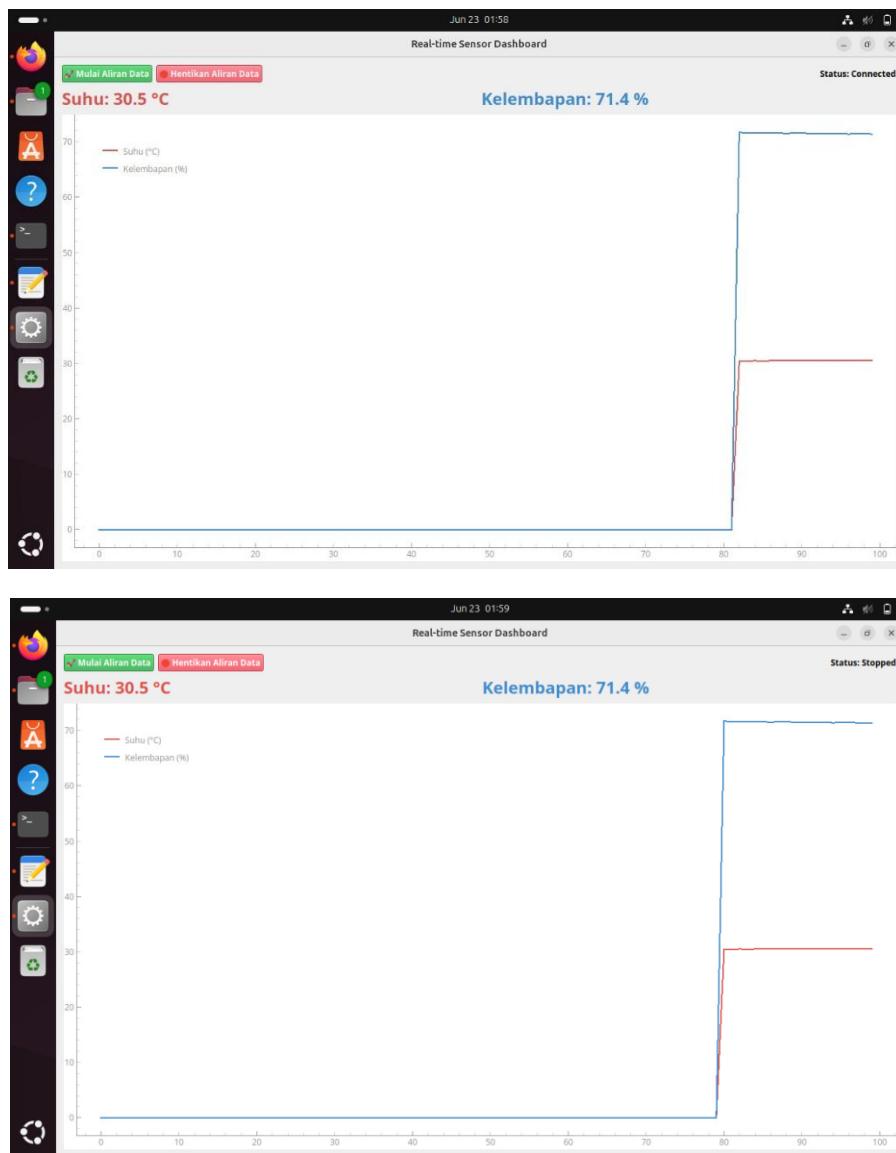
c. Time Series Grafana



d. Data Transaksi



e. Tampilan Qt



VIII. KESIMPULAN DAN REKOMENDASI

Implementasi sistem monitoring suhu dan kelembapan berbasis sensor SHT20 berhasil dilakukan dengan pendekatan terintegrasi menggunakan bahasa pemrograman Rust, protokol Modbus RTU, serta platform InfluxDB dan Grafana. Sistem ini mampu melakukan pembacaan data suhu dan kelembapan secara real-time, mengirimkan data melalui protokol TCP, menyimpannya dalam time-series database (InfluxDB), dan memvisualisasikannya melalui dashboard Grafana. Hasil pengujian menunjukkan bahwa data dari sensor berhasil terbaca dan divisualisasikan dengan baik, memungkinkan pengguna untuk melakukan pemantauan lingkungan fermentasi kopi secara historis maupun real-time secara akurat dan efisien. Sistem ini diharapkan dapat membantu menjaga konsistensi kualitas fermentasi kopi pasca panen melalui pengawasan parameter lingkungan secara terus-menerus.

Untuk pengembangan lebih lanjut, disarankan agar sistem monitoring ini dilengkapi dengan fitur kendali otomatis, misalnya pengaturan suhu atau kelembapan secara adaptif jika terdeteksi berada di luar ambang batas yang ditentukan. Selain itu, dapat ditambahkan alarm peringatan dini (misalnya buzzer atau notifikasi digital) apabila terjadi kondisi ekstrem.

IX. DAFTAR PUSTAKA

- Anggara Trisna Nugraha, R. F. (2022). Rancang Bangun Penstabil Kinerja Panel Hubung Bagi Tegangan Rendah . *Journal of Computer, Electronic, and Telecommunication* .
- Bagus Irfanzah Arda Nugraha, R. S. (2024). Sistem Monitoring pH dan Suhu Pada Fermentasi Kopi Berbasis Internet of Things. *G-Tech, Jurnal Teknologi Terapan*.
- Doni Usman, A. S. (2015). Fermentasi Kopi Robusta (*Coffea canephora*) Menggunakan Isolat Bakteri Asam Laktat Dari Feces Luwak Dengan Perlakuan Lama Waktu Inkubasi. *Jurnal Biologi, Volume 4 No 3*.
- Endang Kwartiningsih, L. N. (2005). FERMENTASI SARI BUAH NANAS MENJADI VINEGAR . *E K U I L I B R I U M*.
- Levente Manases, D. Z. (2022). Automation of Network Traffic Monitoring using Docker images of Snort3, Grafana and a custom API. *RoEduNet Conference: Networking in Education and Research*.
- Rangarajan, M. B. (2002). TCP Servers: Offloading TCP Processing in Internet Servers. Design, Implementation, and Performance . *Rutgers University Libraries*.
- Syehda Noor Zehra Naqvi, S. Y. (2017). Time Series Databases and InfluxDB. *Advanced Databases*.

X. LAMPIRAN

LINK REPOSITORY GITHUB	https://github.com/Tangkasbarap/Interkoneksi_Sistem_Instrumentasi
-------------------------------	---

- **LAMPIRAN LISTING CODE**

```
// --- BAGIAN 1: IMPORTS ---
use axum::{
    extract::{ws::{Message, WebSocket}, State, WebSocketUpgrade, Query},
    http::StatusCode,
    response::{IntoResponse, Json},
    routing::get,
    Router,
};
use ethers::prelude::*;
use futures::{sink::SinkExt, stream::StreamExt};
use influxdb2::{models::DataPoint, Client as InfluxClient};
use serde::{Deserialize, Serialize};
use std::{collections::HashMap, env, fs, net::SocketAddr, str::FromStr, sync::Arc};
use tokio::sync::Mutex;
use tokio::net::{TcpListener, TcpStream};
use tokio::io::{AsyncBufReadExt, BufReader};
use tower_http::cors::{Any, CorsLayer};
use uuid::Uuid;

// --- BAGIAN 2: DEFINISI STRUCT & TYPES ---
#[derive(Deserialize, Serialize, Debug, Clone)]
struct SensorData {
    timestamp: String,
    sensor_id: String,
    location: String,
    process_stage: String,
    temperature_celsius: f32,
    humidity_percent: f32,
}

#[derive(Deserialize)]
struct VerifyParams {
    tx_hash: String,
}

#[derive(Deserialize)]
struct ContractInfo {
    address: String,
}

#[derive(Serialize)]
struct ApiResponse {
    message: String,
```

```

}

type WsClients = Arc<Mutex<HashMap<String, tokio::sync::mpsc::Sender<Message>>>;
```

```

#[derive(Clone)]
struct AppState {
    ethers_provider: Arc<Provider<Http>>,
    contract_address: Arc<H160>,
    ws_clients: WsClients,
    influx_client: Arc<InfluxClient>,
    influx_bucket: Arc<String>,
}
```

```
// --- BAGIAN 3: FUNGSI-FUNGSI LOGIKA ---
```

```

async fn write_to_influxdb(client: &InfluxClient, bucket: &str, data: &SensorData) ->
Result<(), Box<dyn std::error::Error + Send + Sync>> {
    let timestamp =
    chrono::DateTime::parse_from_rfc3339(&data.timestamp)? .timestamp_nanos_opt().ok_or("Ti
    mestamp conversion failed")?;
    let point = DataPoint::builder("monitoring")
        .tag("sensor_id", &data.sensor_id)
        .tag("location", &data.location)
        .tag("stage", &data.process_stage)
        .field("temperature", data.temperature_celsius as f64)
        .field("humidity", data.humidity_percent as f64)
        .timestamp(timestamp)
        .build()?;
    client.write(bucket, futures::stream::iter(vec![point])).await?;
    Ok(())
}

async fn process_socket(stream: TcpStream, app_state: AppState) {
    let mut reader = BufReader::new(stream);
    let mut line_buffer = String::new();
    loop {
        if reader.read_line(&mut line_buffer).await.unwrap_or(0) == 0 { break; }
        let trimmed_line = line_buffer.trim();
        if let Ok(data) = serde_json::from_str(<SensorData>(trimmed_line)) {
            println!("[TCP] 📡 Data diterima: {}", data.sensor_id);
            if let Err(e) = write_to_influxdb(&app_state.influx_client, &app_state.influx_bucket,
&data).await {
                eprintln!("[INFLUX] ❌ Gagal tulis ke DB: {}", e);
            }
            let data_json = serde_json::to_string(&data).unwrap();
            let clients = app_state.ws_clients.lock().await;
            for sender in clients.values() {
                if sender.send(Message::Text(data_json.clone())).await.is_err() {
                    // Klien mungkin terputus, akan dihapus saat koneksi benar-benar tertutup
                }
            }
        }
    }
}

```

```

        }
    } else if !trimmed_line.is_empty() {
        eprintln!("[TCP] ⚠ Gagal parse JSON dari klien: {}", trimmed_line);
    }
    line_buffer.clear();
}
}

async fn tcp_listener_task(app_state: AppState) {
    let tcp_addr = env::var("TCP_SERVER_ADDRESS").unwrap();
    let listener = TcpListener::bind(&tcp_addr).await.unwrap();
    println!("[TCP] 🚀 Server TCP berjalan di {}", tcp_addr);
    loop {
        let (socket, addr) = listener.accept().await.unwrap();
        println!("[TCP] 💎 Koneksi baru dari: {}", addr);
        tokio::spawn(process_socket(socket, app_state.clone()));
    }
}

async fn verify_access_handler(State(state): State<AppState>, Query(params): Query<VerifyParams>) -> impl IntoResponse {
    println!("[API] 📲 Permintaan verifikasi diterima: {}", params.tx_hash);
    let tx_hash = match H256::from_str(&params.tx_hash) {
        Ok(h) => h,
        Err(_) => return (StatusCode::BAD_REQUEST, Json(ApiResponse { message: "Format tx_hash tidak valid.".to_string() })).into_response(),
    };
    if let Some(r) =
        state.ethers_provider.get_transaction_receipt(tx_hash).await.unwrap_or(None) {
        if r.status.unwrap_or_default().as_u64() == 1 && r.to.unwrap_or_default() ==
            *state.contract_address {
            println!("[API] ✅ Verifikasi transaksi BERHASIL: {}", params.tx_hash);
            return (StatusCode::OK, Json(ApiResponse { message: "Verifikasi berhasil. Silakan hubungkan WebSocket.".to_string() })).into_response();
        }
    }
    println!("[API] ❌ Verifikasi transaksi GAGAL: {}", params.tx_hash);
    (StatusCode::UNAUTHORIZED, Json(ApiResponse { message: "Bukti pembayaran tidak valid atau tidak ditemukan.".to_string() })).into_response()
}

async fn websocket_handler(ws: WebSocketUpgrade, State(state): State<AppState>) -> impl IntoResponse {
    println!("[API] Menerima permintaan upgrade ke WebSocket... ");
    ws.on_upgrade(move |socket| handle_websocket(socket, state))
}
// Ganti seluruh fungsi handle_websocket dengan ini

async fn handle_websocket(socket: WebSocket, state: AppState) {
    let (mut sender, mut receiver) = socket.split();

```

```

let client_id = Uuid::new_v4().to_string();
let (tx, mut rx) = tokio::sync::mpsc::channel(100);

state.ws_clients.lock().await.insert(client_id.clone(), tx);
println!("[WS] 🚀 Klien WebSocket baru terhubung: {}", client_id);

// --- BAGIAN YANG DIPERBAIKI ---
// 1. Buat klon/duplikat dari client_id sebelum dipindahkan ke dalam task.
let client_id_clone_for_task = client_id.clone();

tokio::spawn(async move {
    while let Some(msg_to_send) = rx.recv().await {
        if sender.send(msg_to_send).await.is_err() {
            // 2. Gunakan klon di dalam task ini.
            println!("[WS] Gagal mengirim pesan ke klien {}, koneksi mungkin sudah terputus.", client_id_clone_for_task);
            break;
        }
    }
});
// --- AKHIR BAGIAN YANG DIPERBAIKI ---

// Loop ini akan berjalan selama klien masih terhubung
while let Some(Ok(_)) = receiver.next().await {}

// Setelah loop di atas selesai (klien terputus), kita hapus dari daftar.
// Kita menggunakan client_id yang asli di sini, yang kepemilikannya tidak pernah pindah.
state.ws_clients.lock().await.remove(&client_id);
println!("[WS] 💤 Klien WebSocket terputus: {}", client_id);
}

async fn api_server_task(app_state: AppState) {
    let api_addr: SocketAddr =
env::var("API_SERVER_ADDRESS").unwrap().parse().unwrap();
    let cors = CorsLayer::new().allow_origin(Any).allow_methods(Any).allow_headers(Any);
    let app = Router::new()
        .route("/verify-access", get(verify_access_handler))
        .route("/ws", get(websocket_handler))
        .with_state(app_state).layer(cors);
    println!("[API] 🚀 Server API & WebSocket berjalan di http://{}", api_addr);
    let listener = tokio::net::TcpListener::bind(api_addr).await.unwrap();
    axum::serve(listener, app).await.unwrap();
}

// --- BAGIAN 4: FUNGSI MAIN ---
#[tokio::main]
async fn main() {
    dotenvy::dotenv().expect("File .env tidak ditemukan");

    let contract_info_str = fs::read_to_string("../../.deployedAddress.json")
        .expect("File deployedAddress.json tidak ditemukan. Pastikan path relatif sudah benar dan");
}

```

```

Anda sudah menjalankan deployment Hardhat.");
let contract_info: ContractInfo = serde_json::from_str(&contract_info_str).expect("Gagal
parse deployedAddress.json");

let app_state = AppState {
    ethers_provider:
Arc::new(Provider::<Http>::try_from(env::var("GANACHE_URL").unwrap().unwrap()),
    contract_address: Arc::new(H160::from_str(&contract_info.address).unwrap()),
    influx_client: Arc::new(InfluxClient::new(env::var("INFLUXDB_URL").unwrap(),
env::var("INFLUXDB_ORG").unwrap(), env::var("INFLUXDB_TOKEN").unwrap())),
    influx_bucket: Arc::new(env::var("INFLUXDB_BUCKET").unwrap()),
    ws_clients: Arc::new(Mutex::new(HashMap::new())),
);
};

println!("🔥 Memulai semua service...");  

tokio::join!(
    tcp_listener_task(app_state.clone()),
    api_server_task(app_state.clone())
);
}

```

```

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

[dependencies]
tokio = { version = "1", features = ["full"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
chrono = { version = "0.4", features = ["serde"] }
dotenvy = "0.15"
axum = { version = "0.7", features = ["ws"] }
ethers = { version = "2.0.14", features = ["default"] }
influxdb2 = "0.3.0"
futures = "0.3"
uuid = { version = "1.8.0", features = ["v4", "serde"] }
tower-http = { version = "0.5", features = ["cors"] }

```

```
{
"address": "0xe78A0F7E598Cc8b0Bb87894B0F60dD2a88d6a8Ab",
"abi": [
{
    "type": "constructor",
    "stateMutability": "undefined",
    "payable": false,
    "inputs": []
},
{
    "type": "error",
}
```

```
"name": "OwnableInvalidOwner",
"inputs": [
{
  "type": "address",
  "name": "owner"
}
],
},
{
  "type": "error",
  "name": "OwnableUnauthorizedAccount",
  "inputs": [
  {
    "type": "address",
    "name": "account"
  }
]
},
{
  "type": "event",
  "anonymous": false,
  "name": "AccessGranted",
  "inputs": [
  {
    "type": "address",
    "name": "buyer",
    "indexed": true
  },
  {
    "type": "uint256",
    "name": "amountPaid",
    "indexed": false
  }
]
},
{
  "type": "event",
  "anonymous": false,
  "name": "OwnershipTransferred",
  "inputs": [
  {
    "type": "address",
    "name": "previousOwner",
    "indexed": true
  },
  {
    "type": "address",
    "name": "newOwner",
    "indexed": true
  }
]
```

```
        ],
    },
    {
        "type": "function",
        "name": "accessPrice",
        "constant": true,
        "stateMutability": "view",
        "payable": false,
        "inputs": [],
        "outputs": [
            {
                "type": "uint256",
                "name": ""
            }
        ],
    },
    {
        "type": "function",
        "name": "hasAccess",
        "constant": true,
        "stateMutability": "view",
        "payable": false,
        "inputs": [
            {
                "type": "address",
                "name": ""
            }
        ],
        "outputs": [
            {
                "type": "bool",
                "name": ""
            }
        ],
    },
    {
        "type": "function",
        "name": "owner",
        "constant": true,
        "stateMutability": "view",
        "payable": false,
        "inputs": [],
        "outputs": [
            {
                "type": "address",
                "name": ""
            }
        ],
    },
    {

```

```
"type": "function",
"name": "purchaseAccess",
"constant": false,
"stateMutability": "payable",
"payable": true,
"inputs": [],
"outputs": []
},
{
"type": "function",
"name": "renounceOwnership",
"constant": false,
"payable": false,
"inputs": [],
"outputs": []
},
{
"type": "function",
"name": "transferOwnership",
"constant": false,
"payable": false,
"inputs": [
{
"type": "address",
"name": "newOwner"
}
],
"outputs": []
},
{
"type": "function",
"name": "withdrawFunds",
"constant": false,
"payable": false,
"inputs": [],
"outputs": []
}
]
```