

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

Dosen Mata Kuliah: Ahmad Radhy, S.Si., M.Si

**“Smartine : *Smart Incubator Egg Instrument* Sistem Monitoring Suhu dan Kelembaban untuk Inkubator Telur Berbasis IoT dan Visualisasi Realtime”**



Disusun Oleh:

Raffi Fitra Akbar	[2042231018]
Rafi Muhammad Zhafir	[2042231038]
Rany Surya Oktavia	[2042231060]

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

## DAFTAR ISI

<b>DAFTAR ISI .....</b>	i
<b>DAFTAR GAMBAR .....</b>	ii
<b>DAFTAR TABEL .....</b>	ii
<b>BAB 1. PENDAHULUAN .....</b>	3
1.1    Latar Belakang.....	3
1.2    Rumusan Masalah .....	4
1.3    Tujuan .....	4
1.4    Manfaat.....	4
<b>BAB 2. TINJAUAN PUSTAKA .....</b>	6
2.1 Sensor SHT20 .....	6
2.2 Modbus Client .....	7
2.3 TCP Server .....	8
2.4 InfluxDB.....	9
2.5 Grafana .....	10
2.6 Blokchain .....	11
2.7 Web3.....	12
<b>BAB 3. METODOLOGI DAN ARSITEKTUR SISTEM.....</b>	13
3.1 Prosedur Operasional Komunikasi Sistem .....	14
<b>BAB 4. IMPLEMENTASI DAN KODE PROGRAM .....</b>	19
4.1 Rust Modbus Client .....	19
4.2 Rust TCP Server .....	22
4.3 Konfigurasi InfluxDB dan Integrasi .....	25
4.4 Dashboard Grafana .....	26
<b>BAB 5. PENGUJIAN DAN HASIL.....</b>	28
<b>BAB 6. KESIMPULAN DAN REKOMENDASI.....</b>	30
6.1    Kesimpulan .....	30
6.2    Rekomendasi .....	30

<b>DAFTAR PUSTAKA .....</b>	<b>31</b>
<b>LAMPIRAN.....</b>	<b>32</b>

## **DAFTAR GAMBAR**

Gambar 1. Sensor SHT20 .....	6
Gambar 2. Modbust Sensor SHT20 .....	7
Gambar 3. Serial Komunikasi Modbus.....	7
Gambar 4. Frame data pada Modbus TCP .....	9
Gambar 5. Sistem InfluxDB .....	10
Gambar 6. Sistem Grafana.....	10
Gambar 7. blokchain.....	11
Gambar 8. Web3 .....	12
Gambar 9. Desain Aritektur Sistem .....	13
Gambar 10. <i>Flowchart</i> Sistem .....	13
Gambar 11. Dashboard Grafana.....	28
Gambar 12. Dashboard influxdb.....	28
Gambar 13. Grafik pembacaan Qt .....	29
Gambar 14. Halaman Web3 Smartine.....	29

## **DAFTAR TABEL**

Tabel 1. <i>State of The Art</i> .....	6
----------------------------------------	---

## BAB 1. PENDAHULUAN

### 1.1 Latar Belakang

Saat ini sektor peternakan menjadi sektor yang cukup penting untuk memenuhi kebutuhan pangan masyarakat Indonesia, karena produk peternak merupakan sumber protein hewani. Permintaan terhadap produk peternakan khususnya daging ayam terus meningkat setiap tahunnya. Oleh karena itu para peternak juga membutuhkan sebuah teknologi untuk menghasilkan bibit yang berkualitas dari telur yang dipanen. Jika sebelumnya para peternak melakukan proses panen telur secara manual atau konvensional dengan jarak panen sekitar 21 sampai 30 hari masa panen untuk satu induk ayam. Hal ini akan menjadi permasalahan bagi para peternak ayam yang hanya memiliki induk siap panen yang sedikit.

Selain itu kondisi lingkungan yang kurang optimal juga akan mempengaruhi proses penetasan telur. Untuk mengatasi permasalahan ini telah diteliti suatu alat berupa mesin penetas telur untuk meningkatkan produktifitas dan daya telur yang cukup besar sehingga penetasan menjadi lebih efisien dan banyak. Keberhasilan pada mesin penetas ini dipengaruhi beberapa faktor diantaranya yaitu suhu yang ideal sekitar  $38^{\circ}$ - $39^{\circ}$  Celcius (Nisa & Andreansyah, 2024). Apabila telur ayam menetas di hari ke 20-21, maka telur menetas pada waktu yang sesuai dengan suhu yang stabil. Namun apabila telur menetas pada hari ke 18-19 berarti suhu yang digunakan terlalu tinggi dan akan menyebabkan kematian bagi anak ayam. Faktor lain yang mempengaruhi adalah kelembapan, untuk menetas telur kelembapan yang harus diperhatikan yaitu sekitar 35%-60% (Nisa & Andreansyah, 2024).

Dengan mesin ini kelembapan dan suhu akan tetap stabil karena para peternak dapat melakukan monitoring dan mendapatkan informasi mengenai perkembangan dan kondisi saat proses penetasan sehingga mampu mengurangi terjadinya kematian pada anak ayam. Hal ini terjadi karena mesin ini sudah dilengkapi dengan teknologi berbasis *Internet of Things*. *Internet of Things* (IoT) adalah sistem komputerisasi yang dapat terhubung atau berkomunikasi dengan mesin elektronik serta dapat melakukan pertukaran data melalui jaringan internet sehingga dapat mempermudah pekerjaan manusia(Venty et al., 2020). Dengan menggunakan konsep *Internet of Things* (IoT) akan sangat memudahkan para peternak karena para peternak dapat melakukan pemantauan secara real-time melalui sensor yang terhubung ke jaringan mengenai kondisi di dalam inkubator seperti suhu, kelembapan dan rotasi telur dimanapun dan kapanpun.

Dengan menggunakan konsep IOT sistem monitoring suhu dan pencahayaan akan lebih mudah dan tidak perlu memonitor langsung ke kandang, tinggal kita koneksi alat dan memonitor nya langsung melalui aplikasi berbasis mobile, memonitoring lebih efektif dan membantu peningkatan masa panen dan menekan tingkat quantity dari telur hasil babit unggul induk ayam yang menetas. Dalam penulisan ini penulis tertarik membuat sistem monitoring suhu dan pencahayaan pada inkubator melalui aplikasi, sehingga membantu peternak dalam memonitoring ruang inkubator memalui gadget tanpa harus memonitor langsung ke kandang dan meningkatkan masa panen. Diharapkan dengan adanya inovasi ini dapat meningkatkan quantity pada penetasan telur ayam serta mempercepat waktu panen telur ayam.

## 1.2 Rumusan Masalah

Rumusan masalah dari inovasi ini adalah sebagai berikut :

1. Bagaimana sistem monitoring suhu dan kelembapan pada alat penetasan telur secara real-time dalam industri perunggasan rumahan?
2. Bagaimana inovasi sistem alat penetas telur ayam dapat meningkatkan efisiensi dan nilai ekonomi dalam industri perunggasan rumahan?

## 1.3 Tujuan

Adapun tujuan inovasi ini sebagai berikut :

1. Untuk mengetahui sistem monitoring suhu dan kelembapan pada alat penetasan telur secara real-time dalam industri perunggasan rumahan.
2. Untuk mengetahui inovasi sistem alat penetas telur ayam dapat meningkatkan efisiensi dan nilai ekonomi dalam industri perunggasan rumahan.

## 1.4 Manfaat

Dalam inovasi “Smartine : *Smart Incubator Egg Instrument Sistem Monitoring Suhu dan Kelembaban untuk Inkubator Telur Berbasis IoT dan Visualisasi Realtime*” ini memiliki manfaat dalam berbagai aspek, yaitu:

1. Bagi Mahasiswa: Manfaat yang didapatkan oleh mahasiswa adalah sebuah inisiatif yang dirancang untuk mendorong mahasiswa yang mengembangkan ide-ide di bidang teknologi yang kreatif dan inovatif serta mengimplementasikannya ke dalam bentuk proyek.

2. Bagi Peternak : Manfaat yang didapatkan adalah memberikan solusi yang terjangkau dan efektif bagi peternak. Hal ini dapat meningkatkan daya saing mereka di pasar dan mendukung ketahanan pangan lokal.
3. Bagi Industri Peternakan : Dengan memantau suhu dan kelembaban secara real-time, inovasi ini meningkatkan keberhasilan penetasan telur dan menekan angka kegagalan embrio. Alat ini dapat menjadi standar tambahan dalam manajemen inkubasi modern, mendorong peternakan yang lebih efisien, produktif, dan mendukung ketahanan pangan nasional.

## BAB 2. TINJAUAN PUSTAKA

Topik, Penulis dan Tahun	Teknologi yang digunakan	Hasil
Sistem Monitoring Suhu Pada Inkubator Penetas Telur Berbasis IoT. Yunus et al. (2024)	Sensor DHT11, NodeMCU ESP8266, platform Blynk	Sistem efektif dalam meningkatkan kualitas penetasan telur dan efisiensi pemantauan
Monitoring Inkubator Telur Menggunakan Protokol ESP-MESH. Asyam & Purwoto (2024)	ESP32 & ESP8266, sensor SHTC3, protokol ESP-MESH, platform Thinger.io	Akurasi tinggi (error suhu 0,79%, kelembapan 7,69%), sistem efisien untuk banyak inkubator
Sistem Monitoring Suhu dan Kelembaban Berbasis IoT pada Ruang Data Center. Kusumah et al. (2023)	Sensor DHT11, NodeMCU ESP8266, OLED I2C, MQTT, dashboard web	Error suhu 1,7%, kelembapan 2,1%, sistem stabil dan efisien

**Tabel 1.** *State of The Art*

### 2.1 Sensor SHT20

Sensor SHT20 merupakan sensor suhu dan kelembaban udara yang memiliki presisi tinggi dan dilengkapi dengan fitur waterproof atau tahan air serta dilengkapi dengan probe tambahan

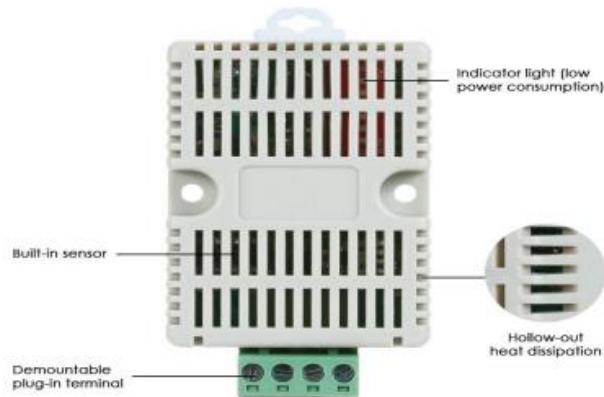


**Gambar 1.** Sensor SHT20

Sensor SHT diatas menggunakan komunikasi Modbus adalah protokol komunikasi serial yang diterbitkan oleh Modicon pada 1979 untuk diaplikasikan pada programmable logic controller (PLC). Kemudian protokol ini telah menjadi standar de facto protokol komunikasi di industri, dan sekarang Modbus merupakan protokol komunikasi dua-arrah yang paling umum digunakan sebagai media penghubung dengan perangkat industri atau media elektronik lainnya dengan komputer.

Paket sensor modbus SHT 20 yang berfungsi untuk mengukur suhu dan kelembaban udara (manual) ,sekaligus yang dialamnya terdapat thermistor tipe NTC (Negative Temperature Coefficient) untuk mengukur suhu, sebuah sensor kelembaban dengan karakteristik resistif terhadap perubahan kadar air di udara serta terdapat chip yang di dalamnya melakukan beberapa konversi

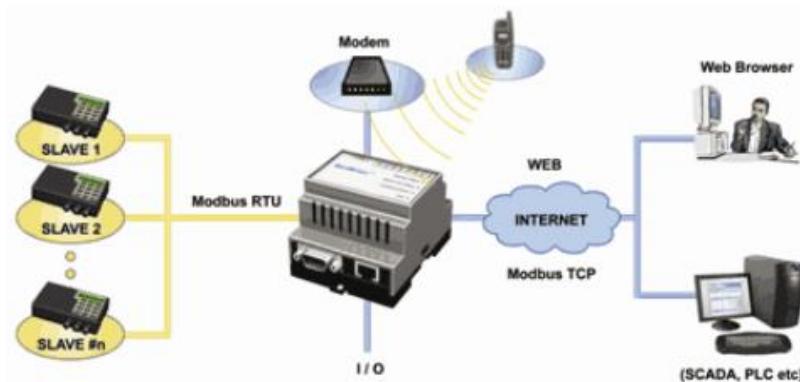
analog ke digital dan mengeluarkan output dengan format single-wire bidirectional (kabel tunggal dua arah). Seperti ditunjukkan pada Gambar berikut.



**Gambar 2.** Modbus Sensor SHT20

## 2.2 Modbus Client

Modbus adalah sebuah protokol terbuka, yang berarti bahwa itu gratis bagi produsen untuk membangun ke dalam peralatan mereka tanpa harus membayar royalti. Hal ini telah menjadi protokol komunikasi standar dalam industri, dan sekarang cara yang paling umum tersedia untuk menghubungkan perangkat elektronik industri (Chandra, 2016). Modbus biasanya digunakan untuk mengirimkan sinyal dari perangkat instrumentasi dan kontrol kembali ke controller utama atau sistem pengumpulan data, misalnya sistem yang mengukur suhu dan kelembaban dan mengkomunikasikan hasilnya ke komputer.



**Gambar 3.** Serial Komunikasi Modbus

Modbus sering digunakan untuk menghubungkan komputer pengawasan dengan unit terminal remote (RTU) kontrol pengawasan dan akuisisi data (SCADA) sistem. Versi protokol Modbus ada untuk baris serial (Modbus RTU dan Modbus ASCII) dan untuk Ethernet (Modbus TCP). Alasan utama penggunaan Modbus secara ekstensif sebagai protokol komunikasi adalah :

- a. Modbus diterbitkan sebagai open protocol dan bebas royalti
- b. Modbus relatif mudah untuk digabungkan dengan jaringan industri
- c. Modbus melakukan transfer data raw bits atau words tanpa membatasi jenis vendor atau jenis merk pabrikan perangkat industri yang digunakan.

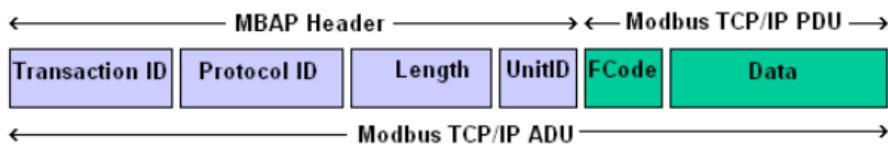
Modbus memungkinkan adanya komunikasi dua-jalur antar perangkat yang terhubung ke jaringan yang sama, misalnya suatu sistem yang mengukur suhu dan kelembaban dan mengkomunikasikan hasilnya ke komputer (HMI). Modbus sering digunakan untuk menghubungkan supervisory computer dengan remote terminal unit (RTU) supervisory control dan sistem akuisisi data (SCADA).

Setiap perangkat yang diinginkan untuk berkomunikasi via protokol Modbus harus diberi alamat yang unik atau tidak boleh sama dengan alamat perangkat lainnya. Dalam komunikasi serial dan jaringan MB+ hanya node yang ditugaskan sebagai Master saja yang dapat memulai perintah, berbeda halnya dengan Ethernet, perangkat manapun dapat mengirimkan perintah Modbus, walaupun biasanya hanya satu perangkat master yang melakukannya. Perintah Modbus berisi alamat Modbus perangkat yang ingin dituju atau yang ingin diminta berkomunikasi. Hanya perangkat yang dimaksudkan akan bertindak atas perintah, meskipun perangkat lain mungkin juga menerima pesan/perintah tersebut (pengecualian adalah perintah broadcastable khusus dikirim ke node 0 yang bertindak tapi tidak diakui). Semua perintah pada Modbus mengandung pemeriksaan informasi, untuk memastikan bahwa perintah yang datang tidak rusak atau error (Agustian, 2019). The Perintah dasar pada Modbus dapat memerintahkan sebuah RTU untuk mengubah nilai salah satu kontrol, register atau membaca sebuah port Input/Output, serta sekaligus memerintahkan perangkat untuk mengirimkan kembali satu atau lebih nilai yang terkandung dalam register yang diakses atau dirubah tersebut.

### 2.3 TCP Server

Transmission Control Protocol (TCP) dan Internet Protocol (IP) merupakan protokol yang digunakan secara bersamaan dan digunakan sebagai protokol transport untuk internet. Ketika

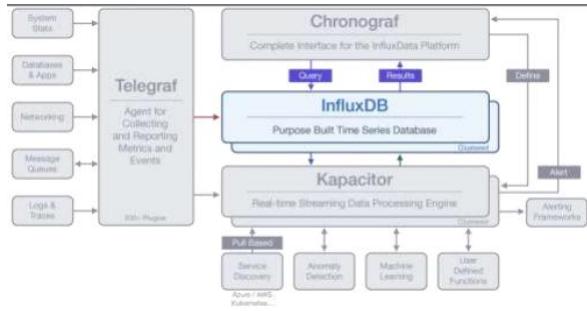
informasi modbus dikirim menggunakan protokol ini, data yang akan diteruskan ke TCP mana informasi tambahan terpasang dan diberikan kepada IP. IP kemudian menempatkan data dalam paket (atau datagram) dan kemudian dikirim. TCP harus membuat sambungan sebelum mentransfer data, karena merupakan protokol berbasis koneksi. Master (atau Client di Modbus TCP) menetapkan koneksi dengan Slave (atau Server). Server menunggu untuk koneksi masuk dari klien. Setelah sambungan dibuat, Server kemudian merespon permintaan dari klien sampai klien menutup koneksi. Modbus TCP/IP lebih cepat dalam melakukan transfer data dibanding dengan Modbus RTU. Pada aplikasi sistem SCADA atau pun Automation yang kompleks dimana digunakan perangkat IED dalam jumlah yang banyak dan beraneka ragam atau dimana tingkat traffic transfer data yang padat, lebih disarankan menggunakan Modbus TCP/IP untuk mencapai tingkat real-time yang lebih tinggi. Namun tentu saja perangkat IED dengan Port TCP/IP itu sendiri harganya relatif lebih mahal dibanding dengan Modbus RTU yang hanya menggunakan komunikasi Port RS-485 (Ariwibisono et al., 2023). Perbedaan mendasar Modbus TCP dengan modbus RTU salah satunya adalah pada frame data yang dikirimkan. Berikut ini adalah frame data pada Modbus TCP :



**Gambar 4.** Frame data pada Modbus TCP

## 2.4 InfluxDB

InfluxDB adalah sebuah database time series open-source yang dikembangkan oleh InfluxData. InfluxDB didukung oleh Bahasa Go (Titin Nurfadila Sudirman, 2019). InfluxDB digunakan sebagai penyimpanan data untuk setiap kasus yang melibatkan sejumlah besar data time-stamped, termasuk pemantauan DevOps, data log, metrik 8 aplikasi, data sensor IoT, dan analisis real-time. Konfigurasi InfluxDB dapat menghemat ruang untuk menyimpan data dalam jangka waktu tertentu, secara otomatis berakhir dan menghapus semua data yang tidak diperlukan dari sistem. InfluxDB juga menawarkan bahasa kueri seperti SQL untuk berinteraksi dengan data (Titin Nurfadila Sudirman, 2019).



**Gambar 5.** Sistem InfluxDB

## 2.5 Grafana

Grafana adalah perangkat open source untuk analisis dan visualisasi metrik. Grafana paling sering digunakan untuk memvisualisasikan data deret waktu untuk infrastruktur dan analitik aplikasi dan juga banyak digunakan di domain lain termasuk sensor industri, otomatisasi rumah, cuaca, dan kontrol proses. Grafana mendukung banyak storage backends yang berbeda untuk data time series (Source Data). Setiap sumber data memiliki kueri editor tertentu yang disesuaikan untuk fitur dan kemampuan tertentu (Grafana, 2019).



**Gambar 6.** Sistem Grafana

## 2.6 Blokchain



**Gambar 7.** blockchain

Blockchain adalah basisdata yang tersebar/terdesentralisasi (decentralized database) yang menggunakan node independen untuk menyimpan dan mengambil data (LaFountain, 2021). Teknologi blockchain menghubungkan blok data secara berurutan dalam buku besar yang didistribusikan. Setiap blok menyimpan berbagai konten, termasuk “hash”, yaitu pengidentifikasi unik (unique identifier) dari blok itu sendiri. Hash melakukan identifikasi dan menautkan blok ini ke semua blok, baik blok sebelumnya dan juga blok setelahnya (Meth, 2019). Jadi bisa disimpulkan bahwa Blockchain merupakan kumpulan dari blok-blok (block) yang berisi data transaksi yang ditautkan/dihubungkan (chain = rantai) dan diurutkan satu sama lain. Blockchain bisa dianggap sebagai sebuah sistem penyimpanan data digital di mana setiap blok yang paling baru atau blok yang paling terakhir dihubungkan, pasti memiliki informasi hash (hash = kode alfanumerik yang mewakili kata, pesan, atau data) dari blok sebelumnya. Setiap blok akan mengacu kepada blok sebelumnya dan seterusnya sehingga membentuk rantai (chain).

Alih-alih bergantung pada entitas pusat, basisdata Blockchain ini justru bekerja pada jaringan global dari banyak node sukarelawan (volunteer nodes). Hal ini juga berarti bahwa tidak ada satu individu pun yang mengontrol data atau jaringan. Semua transaksi dicatat, dapat diakses dan transparan (LaFountain, 2021). Hal ini karena secara prinsip, teknologi Blockchain bisa dianalogikan seperti sebuah buku induk/buku besar. Akan tetapi berbeda dengan buku besar tradisional yang pecatatan dari setiap jumlah transaksi (amounts), pihak-pihak yang terlibat (parties involved), waktu transaksi (time) serta informasi-informasi terkait lainnya hanya dilakukan satu orang atau satu pihak (terpusat). Hal tersebut tidak berlaku dalam prinsip teknologi Blockchain. Blockchain yang

dianalogikan sebagai buku besar terdistribusi mengambil semua informasi tersebut (amounts, parties involved, time of transaction, dan informasi-informasi yang terkait lainnya) untuk kemudian menempatkannya secara daring (online) dan kemudian mendistribusikan salinan informasi tersebut secara indentik kepada semua komputer yang tergabung di dalam sistem. Sehingga salinan identik dari informasi tersebut berada di banyak tempat. Hal ini dilakukan agar informasi ini nantinya bisa divalidasi oleh setiap pihak yang tergabung di dalam sistem. Tentu saja hal ini bertujuan untuk menjamin keamanan dan keaslian informasi tersebut dan hal ini lah yang menjadi keunggulan dari teknologi blockchain ini. Jika kita ingat, blockchain ini memiliki kesamaan dengan konsep yang sudah lama kita kenal di dalam dunia perpustakaan : LOCKSS (Lots of Copies Keep Stuff Save) (Utomo, 2021).

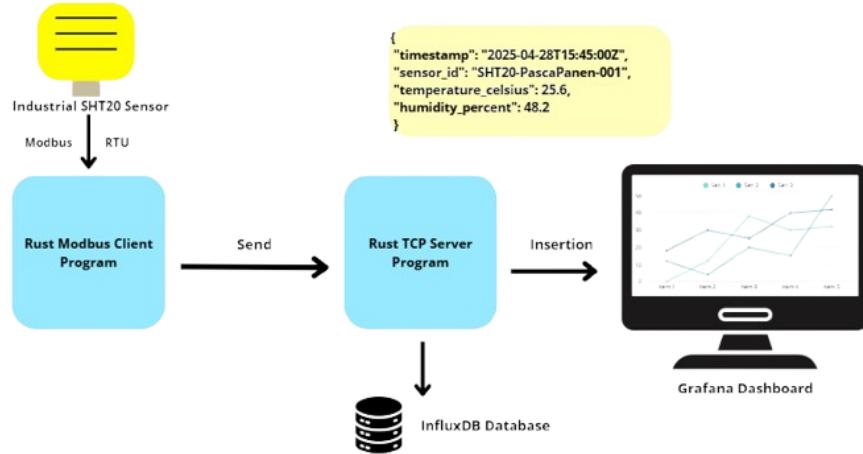
## 2.7 Web3



**Gambar 8.** Web3

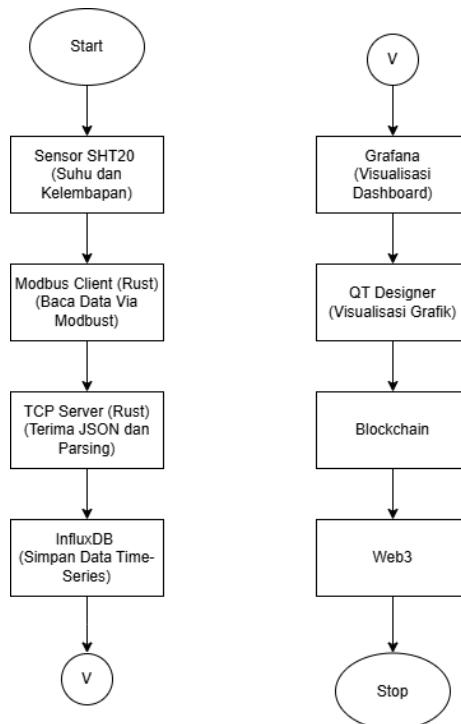
Web3 mengacu pada generasi berikutnya dari teknologi dan protokol internet sumber terbuka terdesentralisasi yang memungkinkan pengguna untuk berinteraksi dengan aplikasi terdesentralisasi (dApps) dan aset digital dengan cara yang tidak dapat dipercaya, aman, dan transparan. Konsep inti dibalik Web3 adalah teknologi blockchain, jaringan peer-to-peer, dan smart contract, yang bertujuan memberi pengguna lebih banyak kendali atas data dan aset online mereka, serta distribusi kekuatan yang lebih adil di internet (Cao, 2022).

### BAB 3. METODOLOGI DAN ARSITEKTUR SISTEM



**Gambar 9.** Desain Aritektur Sistem

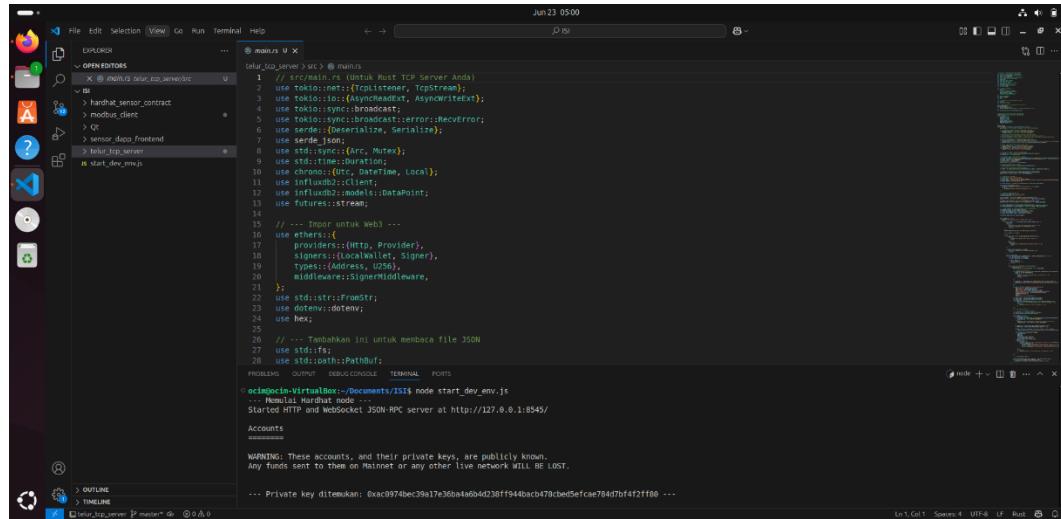
Gambar diatas merupakan desain arsitektur sistem yang digunakan pada inovasi kami ”Smartine : *Smart Incubator Egg Instrument* Sistem Monitoring Suhu dan Kelembaban untuk Inkubator Telur Berbasis IoT dan Visualisasi Realtime”. Mulai dari sensor SHT20 kemudian ke modbus client yang hasilnya akan dikirim ke rust TCP server program. Pada rust TCP server program akan menyimpan data di influxDB database dan akan diteruskan ke grafana dashboard. Lebih detailnya ada dalam flowchart dibawah ini.



**Gambar 10.** Flowchart Sistem

### 3.1 Prosedur Operasional Komunikasi Sistem

1. Langkah pertama adalah menjalankan layanan berbasis Node.js.
2. Buka terminal atau command prompt, lalu arahkan direktori kerja ke folder project Node menggunakan perintah `cd Dokuments/ISI`.



```
Jun 23 05:00
@ manus 0 x
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
x @ manus/telur_tcp_server/src
  1 // Import Tokio (Rust TCP Server Andi)
  2 use tokio::net::TcpListener, TcpStream;
  3 use tokio::io::AsyncReadExt, AsyncWriteExt;
  4 use tokio::sync::broadcast;
  5 use tokio::sync::broadcast::error::RecvError;
  6 use serde::Deserialize, Serialize;
  7 use serde_json;
  8 use std::sync::Arc, Mutex;
  9 use chrono::Utc, DateTime, Local;
10 use influxdb::client;
11 use influxdb::models::DataPoint;
12 use futures::stream;
13
14 // --- Import Untuk Web3 ---
15 use ethers::prelude::*;
16 providers::HttpProvider,
17 signers::LocalWallet, Signer,
18 types::Address, U256,
19 middleware::SignerMiddleware,
20 );
21 use std::str::FromStr;
22 use dotenv::dotenv;
23 use hex;
24
25 // --- Tambahkan Ini Untuk Membaca File JSON
26 use std::fs;
27 use std::path::PathBuf;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

o oclinpc-in-virtualBox-Dokuments/ISI\$ node start\_dev\_env.js

Hardhat Hardhat node -

Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

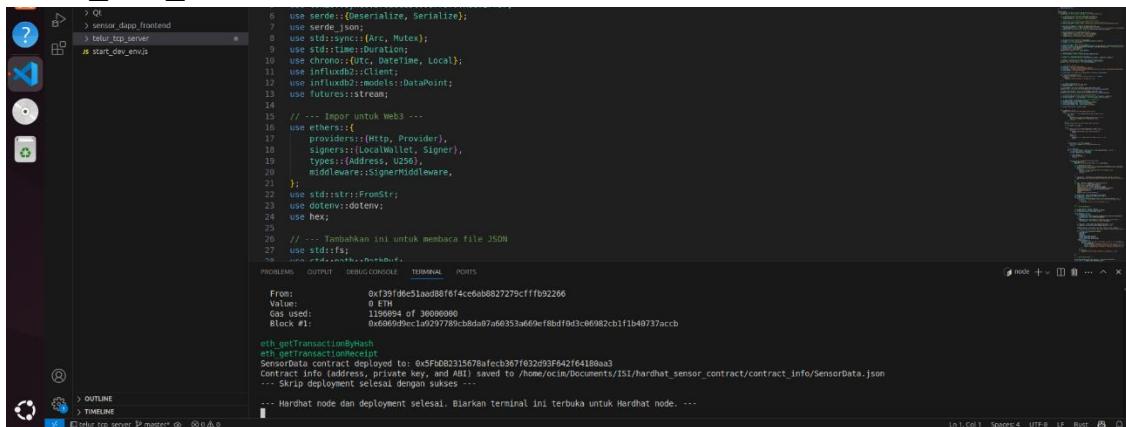
Accounts

WARNING: These accounts, and their private keys, are publicly known.  
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

... Private key ditemukan: 0xacc974dec3ec9a7e3b5aa6a4d30f194abcb79cbed5efcae704d7bf42f2ff00 ...

Un 1 Col 1 Spaces 4 UTF-8 LF Read ⌂

3. Setelah itu, jalankan program dengan perintah `node index.js` atau gunakan `node start_dev_env` jika script sudah didefinisikan dalam file package.json.



```
Jun 23 05:00
@ manus 0 x
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
x @ manus/telur_tcp_server/src
  1 use serde::Deserialize, Serialize;
  2 use serde_json;
  3 use std::sync::Arc, Mutex;
  4 use std::time::Duration;
  5 use chrono::Utc, DateTime, Local;
  6 use influxdb::client;
  7 use influxdb::models::DataPoint;
  8 use futures::stream;
  9
10 // --- Import Untuk Web3 ---
11 use ethers::prelude::*;
12 providers::HttpProvider,
13 signers::LocalWallet, Signer,
14 types::Address, U256,
15 middleware::SignerMiddleware,
16 );
17 use std::str::FromStr;
18 use dotenv::dotenv;
19 use hex;
20
21 // --- Tambahkan Ini Untuk Membaca File JSON
22 use std::fs;
23 use std::path::PathBuf;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

o oclinpc-in-virtualBox-Dokuments/ISI\$ node start\_dev\_env.js

From: 0x1391dd51aadd8f614cc6abb827279cfffb2266  
Value: 0 ETH  
Gas used: 1199694 of 36000000  
Block #1: 0xd6869de9c1a929778c08da07a60353a6d0ef78bd1fd3c806982cb1f1b40737accb  
eth\_getTransactionByHash  
getTransactionByHash  
SensorData contract deployed to: 0x5F1d0B2315678afeCB367f032d93F642f6410aa3  
Contract info (address, private key, and ABI) saved to /home/oclin/Documents/ISI/hardhat\_sensor.contract/contract\_info/sensorData.json  
-- skip deployment selesai dengan sukses ...

-- Hardhat node dan deployment selesai. Biarkan terminal ini terbuka untuk Hardhat node. --

Un 1 Col 1 Spaces 4 UTF-8 LF Read ⌂

4. Jika sudah maka akan tertulis Hardhat node dan deploy selesai. Biarkan terminal ini terbuka untuk Hardhat node
5. Selanjutnya, jalankan program TCP server yang ditulis menggunakan bahasa pemrograman Rust.
6. Buka terminal baru agar proses sebelumnya tetap berjalan
7. Kemudian arahkan direktori ke folder proyek TCP server, `cd telur_tcp_server/`

```

// src/main.rs (Untuk Rust TCP Server Anda)
use tokio::net::TcpListener, TcpStream;
use tokio::io::AsyncReadExt, AsyncWriteExt;
use tokio::sync::broadcast;
use tokio::sync::broadcast::error::RecvError;
use serde::Deserialize, Serialize;
use serde_json;
use std::sync::Arc, Mutex;
use std::time::Duration;
use chrono::Utc, DateTime, Local;
use influxdb2::client;
use influxdb2::models::dataPoint;
use futures::stream;
use ethers::types::Address;
use ethers::signers::LocalWallet;
use ethers::providers::HttpProvider;
use ethers::signers::InMemorySigner;
use hex;
// ... Tambahan ini untuk membaca file JSON
use serde::de::DeserializeOwned;

```

```

ocinocim@VirtualBox:~/Documents/ISI$ cd telur_tcp_server/
ocinocim@VirtualBox:~/Documents/ISI$ telur_tcp_server$ cargo run
Compiling tcp server v0.1.0 (/home/ocin/Documents/ISI/telur_tcp_server)
Finished dev profile in 0.00s
Running `target/debug/telur_tcp_server` [unoptimized + debuginfo] (target(s) in mem)
[...]
wallet address: 0xf39fd6e1a0d96f4ce0a08827279cffff92206
smart contract address: 0x5fbdb23156878eb36703205f642f54100ea3
Health check healthy: HealthCheck { name: "influxdb", message: Some("ready for queries and writes"), checks: [], status: Pass, version: Some("v2.7.11"), commit: Some("1fb545a5") }
Server (untuk klien GUI) mendengarkan di 127.0.0.1:7878
Server (untuk Rust Modbus Client) mendengarkan di 127.0.0.1:7877
Task penerima data dari Modbus Client siap.

```

8. Setelah berada di direktori yang benar, jalankan program dengan perintah cargo run. Program ini akan memulai server TCP untuk menerima dan mengelola koneksi dari client.
9. Setelah TCP server berjalan, lanjutkan dengan menjalankan program Modbus yang juga dibuat dengan Rust.

```

// src/main.rs (Untuk Rust TCP Server Anda)
use tokio::net::TcpListener, TcpStream;
use tokio::io::AsyncReadExt, AsyncWriteExt;
use tokio::sync::broadcast;
use tokio::sync::broadcast::error::RecvError;
use serde::Deserialize, Serialize;
use serde_json;
use std::sync::Arc, Mutex;
use std::time::Duration;
use chrono::Utc, DateTime, Local;
use influxdb2::client;
use influxdb2::models::dataPoint;
use futures::stream;
use ethers::types::Address;
use ethers::signers::LocalWallet;
use ethers::providers::HttpProvider;
use ethers::signers::InMemorySigner;

```

```

ocinocim@VirtualBox:~/Documents/ISI$ cd modbus_client/
ocinocim@VirtualBox:~/Documents/ISI$ modbus_client$ cargo run
warning: unused import: `self`
--> src/main.rs:16:15
36 use std::io::(Self, Read, Write); // Mengimport Read dan Write untuk operasi file
                                         ^~~~~
note: #[warn(unused_imports)] on by default
warning: modbus_client [bin "modbus_client"] generated 1 warning (run 'cargo fix --bin "modbus_client" to apply 1 suggestion)
Finished dev profile in 0.00s
Running `target/debug/modbus_client`
InfluxDB dimulai (dari file): 2025-06-23T05:25:00Z
Hari ke-4, Temp=30.2°C, RH=71.6%
Sending JSON to 127.0.0.1:7877: {"humidity_percent":71.59999874742111, "location": "Inkubator 1", "process_stage": "Hari ke-4", "sensor_id": "SHT20-Smartline", "temperature_celsius": 30.200000762939453, "timestamp": "2025-06-23T05:25:00Z", "unit": "%RH", "unit_label": "Percent (%)", "value": 71.59999874742111}
Sending JSON to 127.0.0.1:7877: {"humidity_percent": 67.99999815258789, "location": "Inkubator 1", "process_stage": "Hari ke-4", "sensor_id": "SHT20-Smartline", "temperature_celsius": 31.200000762939453, "timestamp": "2025-06-23T05:25:07:00Z"}

```

10. Buka terminal baru kembali, lalu pindah ke folder project Modbus dengan perintah `cd modbus_client/`. Jalankan program ini menggunakan perintah cargo run, yang akan menginisialisasi komunikasi protokol Modbus.
11. Berikutnya, jalankan antarmuka pengguna (GUI) yang dikembangkan menggunakan Python dan framework Qt.
12. Buka terminal baru, arahkan direktori kerja ke folder GUI, `cd Qt/`.

```

Jun 23 05:15
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
main.py
hardhat_sensor_contract
modbus_client
venv_coffee
configuration
sensor_dsp_frontend
telur_tcp_server
start_dev_envs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● ocmocain-VirtualBox:~/Documents/ISI/qt$ cd QT/
● ocmocain-VirtualBox:~/Documents/ISI/qt$ pip install influxdb-client
● ocmocain-VirtualBox:~/Documents/ISI/qt$ venv_coffee/bin/activate
(venv_coffee) ocmocain-VirtualBox:~/Documents/ISI/qt$ pip install influxdb-client
Collecting influxdb-client==1.49.0-py3-none-any.whl.metadata (0.8 kB)
  Using cached influxdb-client-1.49.0-py3-none-any.whl.metadata (0.8 kB)
Collecting reactive==0.4 (from influxdb-client)
  Using cached reactive-0.4-py3-none-any.whl.metadata (5.5 kB)
Collecting certifi==2025.6.15-py3-none-any.whl (2.4 kB)
  Using cached certifi-2025.6.15-py3-none-any.whl.metadata (2.4 kB)
Collecting python-dateutil==2.5.3 (from influxdb-client)
  Using cached python-dateutil-2.5.3-py3-none-any.whl.metadata (8.4 kB)
Collecting setuputils==21.0.0 (from influxdb-client)
  Using cached setuputils-20.0.0-py3-none-any.whl.metadata (6.6 kB)
Collecting urllib3==1.26.8 (from influxdb-client)
  Using cached urllib3-1.26.8-py3-none-any.whl.metadata (6.5 kB)
Collecting idna==3.4 (from influxdb-client)
  Using cached idna-3.4-py3-none-any.whl (11 kB)
Collecting six==1.5 (from python-dateutil==2.5.3-influxdb-client)
  Using cached six-1.5-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting certifi==2025.6.15-py3-none-any.whl (2.4 kB)
  Using cached certifi-2025.6.15-py3-none-any.whl.metadata (2.4 kB)
Collecting typing_extensions==4.14.0-py3-none-any.whl.metadata (3.0 kB)
  Using cached typing_extensions-4.14.0-py3-none-any.whl (746 kB)
Collecting influxdb-client==1.49.0-py3-none-any.whl (746 kB)
  Using cached influxdb-client-1.49.0-py3-none-any.whl (229 kB)
Using cached reactivex==4.0.4-py3-none-any.whl (217 kB)
Using cached setuputils==20.0.0-py3-none-any.whl (11.2 kB)
Using cached idna==3.4-py3-none-any.whl (11 kB)
Using cached six==1.17.0-py2.py3-none-any.whl (11 kB)
Using cached typing_extensions==4.14.0-py3-none-any.whl (43 kB)

```

13. Sebelum menjalankan program, aktifkan virtual environment dengan perintah `source venv_coffee/bin/activate`, yang akan mengatur lingkungan Python agar sesuai dengan dependensi yang telah disiapkan.
14. Lakukan install influx db client dengan sourse `pip install influxdb-client`

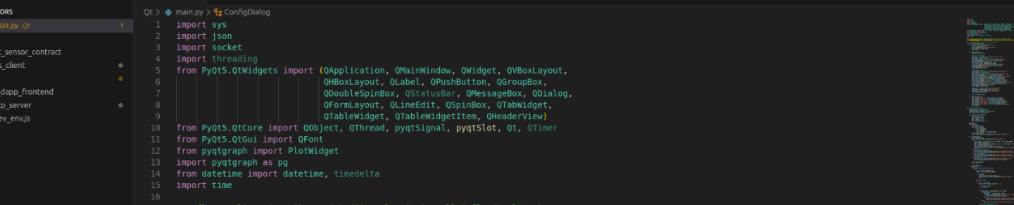
```

Jun 23 05:16
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
main.py
hardhat_sensor_contract
modbus_client
venv_coffee
configuration
sensor_dsp_frontend
telur_tcp_server
start_dev_envs
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● (venv_coffee) ocmocain-VirtualBox:~/Documents/ISI/qt$ pip install PyQt5
Collecting PyQt5
  Using cached PyQt5-5.15.11-cp38-abi3-manylinux2_2_17_x86_64.whl.metadata (2.1 kB)
Collecting PyQt5sip==13.12.15 (from PyQt5)
  Using cached PyQt5sip-13.12.15-py3-cp38-cp38-manylinux2_2_17_x86_64-manylinux1_x86_64.whl.metadata (472 bytes)
Collecting PyQt5sip==13.12.15-py3-cp38-cp38-manylinux2_2_17_x86_64-manylinux1_x86_64.whl (536 bytes)
  Using cached PyQt5sip-13.12.15-py3-cp38-cp38-manylinux2_2_17_x86_64-manylinux1_x86_64.whl (536 bytes)
Using cached PyQt5sip-5.15.11-cp38-abi3-manylinux2_2_17_x86_64.whl (8.2 kB)
Using cached PyQt5_QTS-5.15.17-py3-none-any!linux2014.x86_64.whl (61.1 kB)
Using cached PyQt5_sip-12.17.6-cp38-cp38-manylinux2_2_17_x86_64-manylinux1_x86_64.whl (281 kB)
Installing collected packages: PyQt5sip, PyQt5, PyQt5-sip
Successfully installed PyQt5sip-13.12.15 PyQt5-5.15.11 PyQt5-sip-12.17.6

```

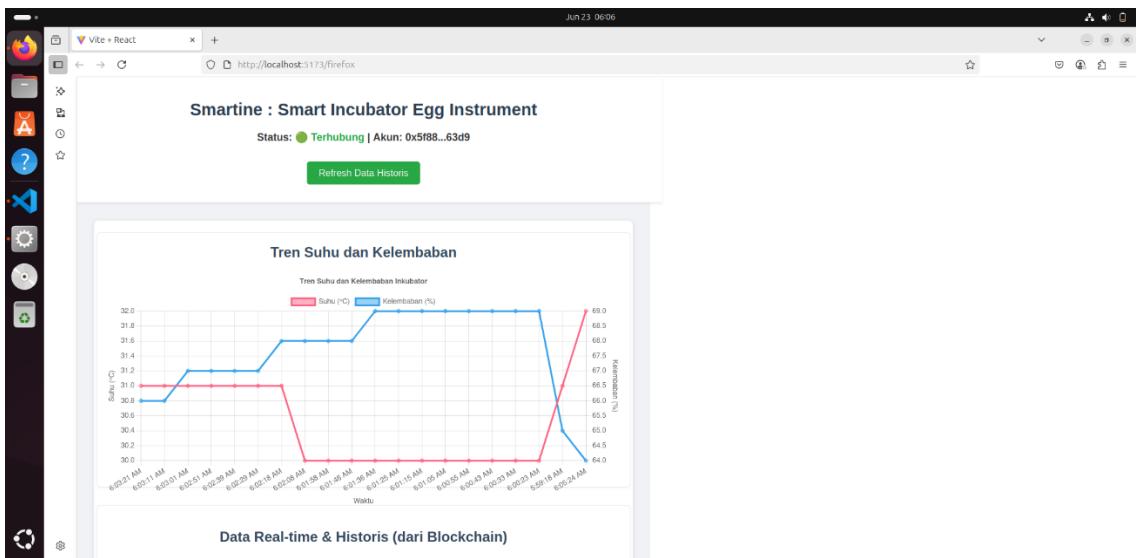
15. Sebelum melakukan perintah ke python, tulis source `pip install PyQt5` untuk menginstall
16. Setelah environment aktif, jalankan file utama GUI dengan perintah `python3 main.py`. sebelum itu install pyqtgraph

17. Terakhir, jalankan tampilan web (frontend) berbasis Node.js
  18. Buka terminal baru lagi, lalu pindah ke direktori project web frontend menggunakan perintah `cd sensor dapp frontend/`



```
Jun 23 05:20
File Edit Selection View Go Run Terminal Help ← → ⌂ ISI ⌂
EXPLORER OPEN EDITORS ... main.py 1
Qt > main.py @ ConfigDialog
  1 import sys
  2 import json
  3 import socket
  4 import threading
  5 from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout,
  6                             QHBoxLayout, QLabel, QPushButton, QGroupBox,
  7                             QFormLayout, QLineEdit, QMessageBox, QDialog,
  8                             QFileDialog, QTableWidget, QTableWidgetItem, QHeaderView
  9
 10 from PyQt5.QtCore import QObject, QThread, pyqtSignal, pyqtSlot, Qt, QTimer
 11 from PyQt5.QtGui import QFont
 12 from pygraphviz import PlotWidget
 13 import requests as rq
 14 from datetime import datetime, timedelta
 15 import time
 16
 17 # InfluxDB Client (Pastikan sudah diinstal: pip install influxdb-client)
 18 from influxdb_client import InfluxDBClient
 19
 20 # --- Dialog Konfigurasi TCP Server dan InfluxDB ---
 21 class ConfigDialog(QDialog):
 22     def __init__(self, config, parent=None):
 23         super().__init__(parent)
 24         self.setWindowTitle("Konfigurasi Sistem")
 25         self.config = config.copy()
 26         self.initUI()
 27
 28     def initUI(self):
 29
  PROBLEMS 0 OUTPUT DEBUG CONSOLE TERMINAL PORTS
 30
 31 o cincinocin-VirtualBox::~/Documents/ISII/sensor_dapp.frontend/
 32 o cincinocin-VirtualBox::~/Documents/ISII/sensor_dapp_frontend$ npm run dev
 33
 34 > sensor_dapp_frontend@0.0.0 dev
 35 > vite
 36
 37 5:18:48 AM [vite] (client) Re-optimizing dependencies because vite config has changed
 38
 39 VITE v6.3.5 ready in 1346 ms
 40
 41 - Local: http://localhost:5173/
 42 - Network: use --host to expose
 43 - press h + enter to show help
```

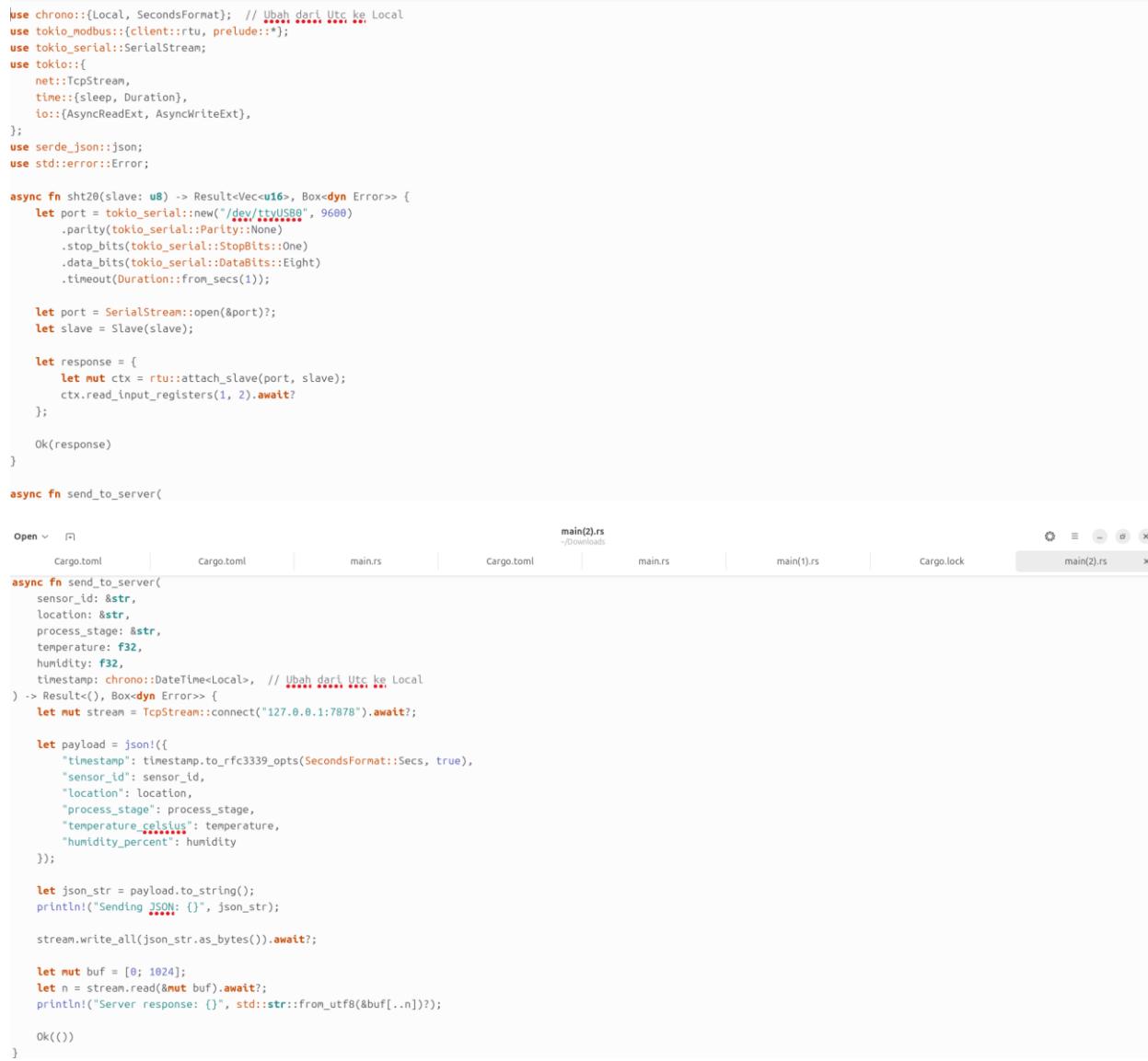
19. Setelah itu, aktifkan mode pengembangan dengan perintah `npm run dev` yang akan memulai server lokal dan menampilkan antarmuka web yang terhubung dengan backend dan sistem lainnya.
  20. Dari local host yang tertera, buka firefox dan search laman <http://localhost:5173/>
  21. Maka akan tertampil sebagai berikut



22. Jika ingin menghubungkan ke metamask, klik pada extension di dalam firefox lalu klik pada Metamask untuk menghubungkan blockchain
23. Setelah itu klik pada tampilan kiri, Lalu klik ‘add a custom network’
24. Lalu sesuaikan dengan localhost dan klik save
25. kemudian klik panah ke bawah di sebelah account, lalu klik ‘add account or hardware wallet, lalu klik private keys, Kemudian isikan Private Keys dan klik import.
26. Kemudian klik ‘muat data sensor’ pada halaman Web3, kemudian akan muncul tampilan dari extension metamask. Kemudian klik connect maka tampilan dari Web3 akan menampilkan tabel dari data yang telah dikirimkan ke TCP Server

## BAB 4. IMPLEMENTASI DAN KODE PROGRAM

### 4.1 Rust Modbus Client



```
use chrono::{Local, SecondsFormat}; // Ubah dari Utc ke Local
use tokio_modbus::{Client, prelude::*};
use tokio_serial::SerialStream;
use tokio::{
    net::TcpStream,
    time::Sleep, Duration,
    io::{AsyncReadExt, AsyncWriteExt},
};
use serde_json::json;
use std::error::Error;

async fn sht20(slave: u8) -> Result<Vec<u16>, Box
```

Program diatas merupakan program dari Modbus RTU, cara membaca sensor menggunakan Modbus RTU dalam program di atas dilakukan melalui komunikasi serial menggunakan protokol Modbus RTU, yang diimplementasikan dengan bantuan library tokio\_modbus dan tokio\_serial. Sensor yang digunakan, seperti SHT20, terhubung ke komputer atau mikrokontroler melalui port serial /dev/ttyUSB0, dengan konfigurasi komunikasi standar

yaitu baudrate 9600, parity none, stop bit satu, dan data bit delapan. Proses pembacaan dimulai dengan :

1. Membuka koneksi ke port serial menggunakan SerialStream
2. Menetapkan alamat slave sensor menggunakan Slave(slave).
3. Setelah itu, koneksi Modbus RTU dibuat menggunakan rtu::attach\_slave, yang memungkinkan pembacaan register input pada alamat tertentu. Dalam program ini, dua register dibaca secara berurutan dari alamat 1, yang masing-masing berisi nilai suhu dan kelembaban dalam bentuk bilangan bulat.
4. Nilai-nilai tersebut kemudian dikonversi menjadi format desimal dengan membaginya dengan 10, sehingga didapatkan suhu dalam satuan derajat Celsius dan kelembaban dalam persen. Seluruh proses ini berjalan secara asinkron dan dilakukan berulang secara periodik setiap 10 detik dalam loop utama program.

```
async fn send_to_server(  
    sensor_id: &str,  
    location: &str,  
    process_stage: &str,  
    temperature: f32,  
    humidity: f32,  
    timestamp: chrono::DateTime<Local>,  
) -> Result<(), Box<dyn Error>> {  
    let mut stream = TcpStream::connect("127.0.0.1:7878").await?;  
  
    let payload = json!({  
        "timestamp": timestamp.to_rfc3339_opts(SecondsFormat::Secs, true),  
        "sensor_id": sensor_id,  
        "location": location,  
        "process_stage": process_stage,  
        "temperature_celsius": temperature,  
        "humidity_percent": humidity  
    });  
  
    let json_str = payload.to_string();  
    println!("Sending JSON: {}", json_str);  
  
    stream.write_all(json_str.as_bytes()).await?;
```

```
let mut buf = [0; 1024];
let n = stream.read(&mut buf).await?;
println!("Server response: {}", std::str::from_utf8(&buf[..n])?);

Ok(())
}
```

Program diatas merupakan program untuk mengirim data ke TCP server, berikut merupakan cara mengirim data ke TCP server dari program di atas :

1. Langkah pertama dilakukan melalui fungsi `send_to_server`, yang bertugas mengirimkan data sensor dalam format JSON ke alamat server lokal (127.0.0.1) pada port 7878.
2. Setelah data sensor berupa suhu dan kelembaban berhasil dibaca, fungsi ini dipanggil dengan membawa informasi tambahan seperti ID sensor, lokasi, tahap proses, dan timestamp lokal sebagai penanda waktu pengambilan data.
3. Di dalam fungsi tersebut, koneksi TCP dibuka menggunakan `TcpStream::connect`, kemudian data disusun ke dalam format JSON menggunakan makro `serde_json::json!`, yang selanjutnya dikonversi menjadi string.
4. Data JSON tersebut dikirim ke server melalui `stream.write_all`, dan program menunggu respons dari server menggunakan `stream.read`.
5. Jika server berhasil menerima data dan mengirimkan respons, pesan tersebut ditampilkan di konsol. Proses ini dilakukan setiap 10 detik dalam loop utama, sehingga pengiriman data berjalan secara periodik dan otomatis sesuai dengan siklus pembacaan sensor.

## 4.2 Rust TCP Server

```
use influxdb2::Client;
use influxdb2::models::DataPoint;
use serde::Deserialize;
use tokio::{
    io::AsyncReadExt, AsyncWriteExt,
    net::TcpListener,
};
use futures::stream;
use chrono::{Utc, DateTime};

#[derive(Debug, Deserialize)]
struct SensorData {
    timestamp: String, // Required timestamp in ISO 8601 format
    sensor_id: String, // Required sensor identifier
    location: String, // Required location
    process_stage: String, // Required process stage
    temperature_celsius: f64, // Temperature field with explicit unit
    humidity_percent: f64, // Humidity field with explicit unit
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    // Configure InfluxDB connection
    let influx_url = "http://localhost:8086";
    let influx_org = "Institut Teknologi Sepuluh Nopember";
    let influx_token = "5kyldp22*****N2*****XN*****Dx*****c1cx*****C06n_nkg_dCTLB3Ik_c761a3Vqgas05687r0andV7hNB0rQ=";
    let influx_bucket = "coffee_monitoring_db";

    let client = Client::new(influx_url, influx_org, influx_token);
}
```

```
Cargo.toml | Cargo.toml | main.rs | Cargo.toml | main.rs | main(1).rs | Cargo.lock x

# 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 = "dfbe2d77e56a376000877090da837660b4427aad530e3028d44e0bffe4f89a1c1"
dependencies = [
    "glib",
]

[[package]]
name = "glib"
version = "2.0.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "c12761e0bb2578dd7380c6baa0f4ce03e84ff95e0960231d1dec8bf4d7d6e2627"

[[package]]
name = "aho-corasick"
version = "1.1.3"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "86e0d3430d3a69478ad0993f19238d2df97c50709a52b3c10addcd7f6bc916"
dependencies = [
    "memchr",
]

[[package]]
name = "android-izdata"
```

Program di atas menjelaskan proses bagaimana server TCP menerima data dalam format JSON dari client, melakukan parsing, dan kemudian menyimpan data tersebut ke InfluxDB. Proses dimulai dari :

1. Fungsi main yang membuat koneksi ke InfluxDB menggunakan URL, organisasi, dan token autentikasi.
2. Server TCP kemudian dijalankan pada alamat 127.0.0.1:7878 dan terus menunggu koneksi dari client.

3. Saat client mengirimkan data, server membaca data mentah dari soket, lalu mencoba mengonversinya dari byte menjadi string UTF-8. Setelah berhasil, string tersebut diperlakukan sebagai JSON dan diparsing menggunakan serde\_json menjadi struktur SensorData, yang telah didefinisikan dengan field timestamp, sensor\_id, location, process\_stage, temperature\_celsius, dan humidity\_percent.
4. Selanjutnya, nilai timestamp yang awalnya berupa string ISO 8601 diparsing menjadi objek DateTime dan dikonversi menjadi timestamp dalam satuan nanodetik (i64).
5. Nilai-nilai dari JSON kemudian digunakan untuk membuat DataPoint baru yang siap dikirim ke InfluxDB. DataPoint tersebut berisi informasi tag dan field sesuai dengan struktur data sensor.
6. Jika proses penulisan ke InfluxDB berhasil, server mengirim respons "OK" ke client. Jika terjadi kesalahan, server akan mengirimkan pesan error yang relevan.

```
#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let influx_url = "http://localhost:8086";
    let influx_org = "Institute Teknologi Sepuluh Nopember";
    let influx_token =
        "hkyHqP236AVjN2JL84XYNJPWCDXnC756c1Gxc2CO6n_nkG-dCTLB3Ik-
        c761g3YRqasOSGRZrOandYZhN8QrQ==";
    let influx_bucket = "coffe_monitoring_db";

    let client = Client::new(influx_url, influx_org, influx_token);

    match client.health().await {
        Ok(health) => println!("InfluxDB connection healthy: {:?}", health),
        Err(e) => {
            eprintln!("Failed to connect to InfluxDB: {}", e);
            return Err(e.into());
        }
    }

    let listener = TcpListener::bind("127.0.0.1:7878").await?;
    println!("Server running on 127.0.0.1:7878");

    loop {
        let (mut socket, _) = listener.accept().await?;
        let client = client.clone();
        let bucket = influx_bucket.to_string();
    }
}
```

```

tokio::spawn(async move {
    let mut buf = [0; 1024];

    match socket.read(&mut buf).await {
        Ok(n) if n == 0 => return,
        Ok(n) => {
            let data = match std::str::from_utf8(&buf[..n]) {
                Ok(d) => d,
                Err(e) => {
                    eprintln!("Error parsing data: {}", e);
                    let _ = socket.write_all(b"ERROR: Invalid UTF-8 data").await;
                    return;
                }
            };
            println!("Received raw data: {}", data);

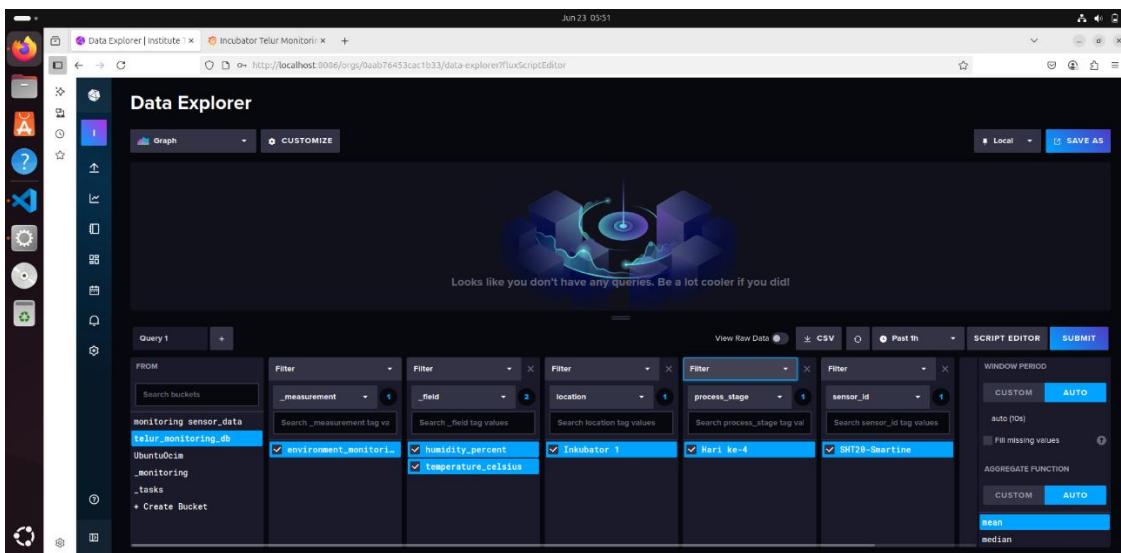
            match serde_json::from_str<SensorData>(data) {
                Ok(sensor_data) => {
                    // Parsing timestamp dan menyusun DataPoint
                    let timestamp = match
                        DateTime::parse_from_rfc3339(&sensor_data.timestamp) {
                            Ok(dt) => dt.with_timezone(&Utc),
                            Err(e) => {
                                eprintln!("Invalid timestamp format: {}", e);
                                let _ = socket.write_all(b"ERROR: Invalid timestamp
format").await;
                                return;
                            }
                        };
                    let timestamp_ns = timestamp.timestamp_nanos_opt().unwrap_or(0);

                    let point = DataPoint::builder("environment_monitoring")
                        .tag("sensor_id", &sensor_data.sensor_id)
                        .tag("location", &sensor_data.location)
                        .tag("process_stage", &sensor_data.process_stage)
                        .field("temperature_celsius", sensor_data.temperature_celsius)
                        .field("humidity_percent", sensor_data.humidity_percent)
                        .timestamp(timestamp_ns)
                        .build()
                        .unwrap();
                }
            }

            match client.write(&bucket, stream::iter(vec![point])).await {
                Ok(_) => {
                    println!("Data successfully written to InfluxDB");
                }
            }
        }
    }
})

```

#### **4.3 Konfigurasi InfluxDB dan Integrasi**





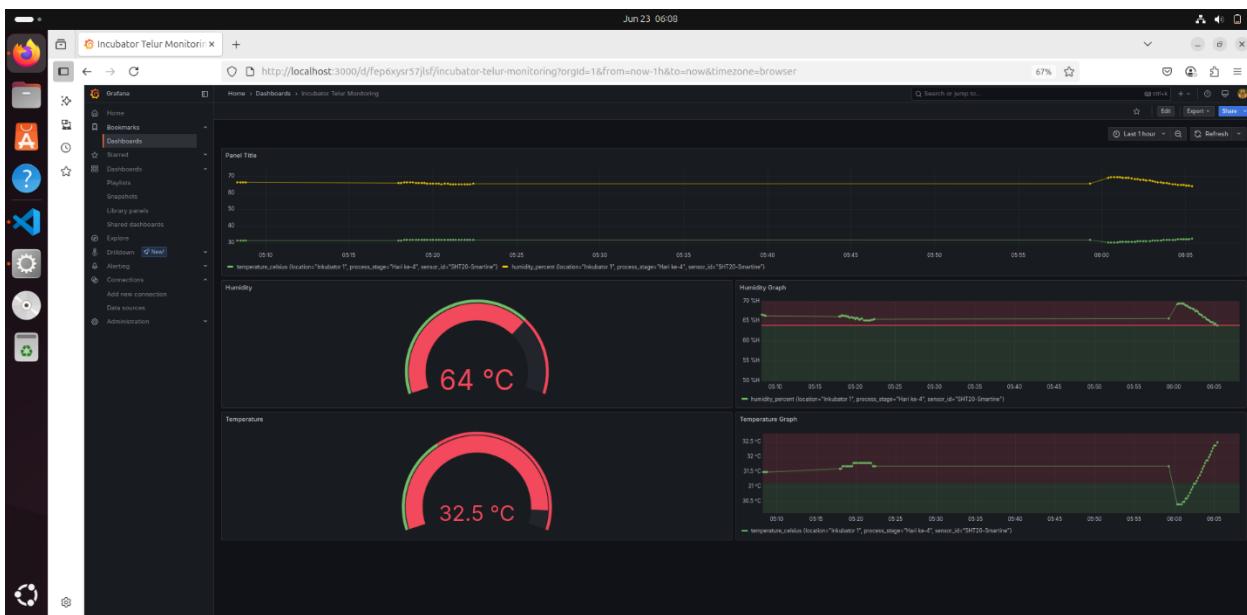
```

1 from(bucket: "telur_monitoring_db")
2 > range(start: v.timeRangeStart, stop: v.timeRangeStop)
3 > filter(fn: (r) => r["measurement"] == "environment_monitoring")
4 > filter(fn: (r) => r["field"] == "humidity_percent" or r["field"] == "temperature_celsius")
5 > filter(fn: (r) => r["location"] == "Inkubator 1")
6 > filter(fn: (r) => r["process_stage"] == "Hari ke-4")
7 > filter(fn: (r) => r["sensor_id"] == "SHT20-Smarline")
8 > aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
9 |> yield(name: "mean")

```

Gambar tersebut menunjukkan tampilan antarmuka baris perintah (*Command Line Interface*) dari InfluxDB saat melakukan eksekusi query terhadap basis data time-series. Pada bagian atas tampak perintah query yang digunakan, telur\_monitoring\_db, yang bertujuan untuk mengambil seluruh data dari measurement, field, location, sensor\_id, dan proses. Hasil query ditampilkan dalam bentuk tabel yang terdiri atas beberapa kolom seperti waktu (time), nilai suhu, kelembapan (value), dan tag atau field lainnya yang relevan. Setiap baris data mewakili satu titik data (data point) yang terekam pada waktu tertentu. Tampilan ini memudahkan pengguna untuk memverifikasi isi data yang tersimpan serta memantau pembacaan sensor secara historis melalui terminal.

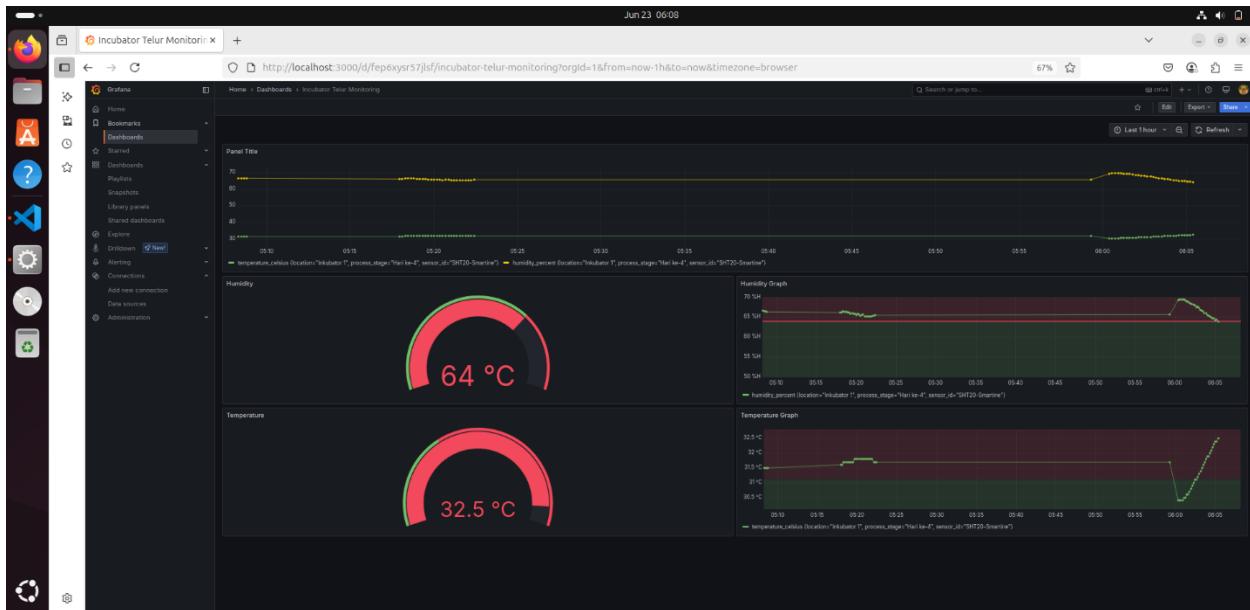
#### 4.4 Dashboard Grafana



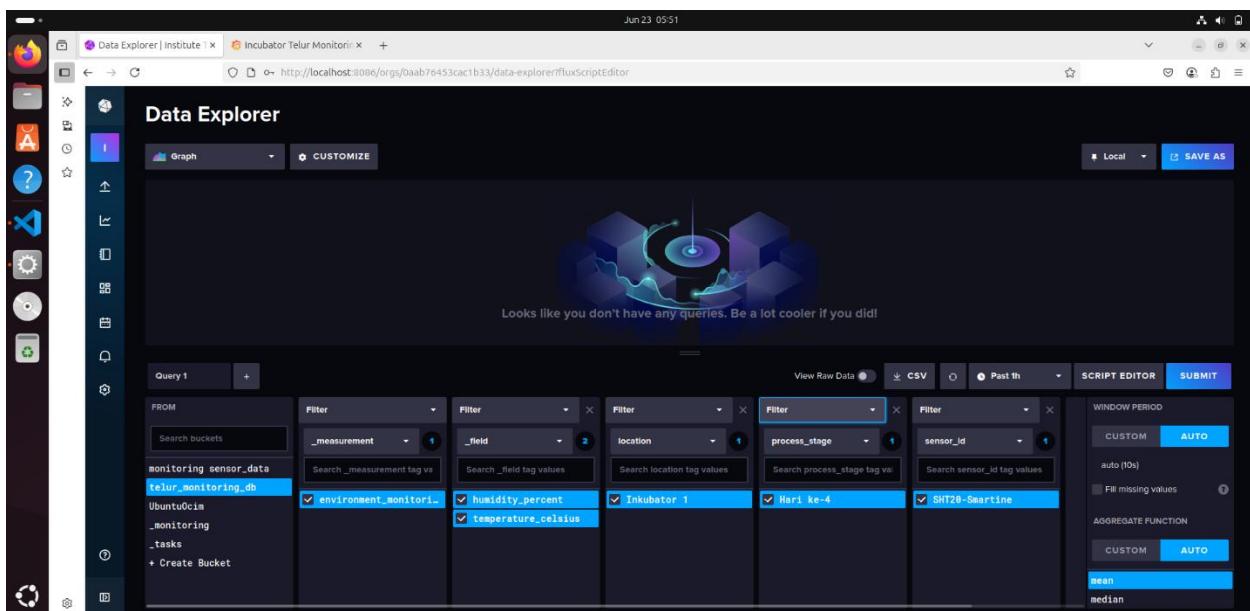
Gambar ini memperlihatkan tampilan antarmuka dashboard sistem monitoring suhu dan kelembaban yang menyajikan visualisasi data dalam bentuk grafik waktu nyata. Dalam satu layar, ditampilkan kurva suhu dan kelembaban yang masing-masing digambarkan dengan warna berbeda, memungkinkan pengguna untuk memantau perubahan kondisi lingkungan secara dinamis dari waktu ke waktu. Selain grafik, tampak pula elemen-elemen pendukung pada

dashboard seperti indikator nilai terkini, status konektivitas, serta komponen navigasi lainnya yang terintegrasi dalam satu desain antarmuka.

## BAB 5. PENGUJIAN DAN HASIL



Gambar 11. Dashboard Grafana

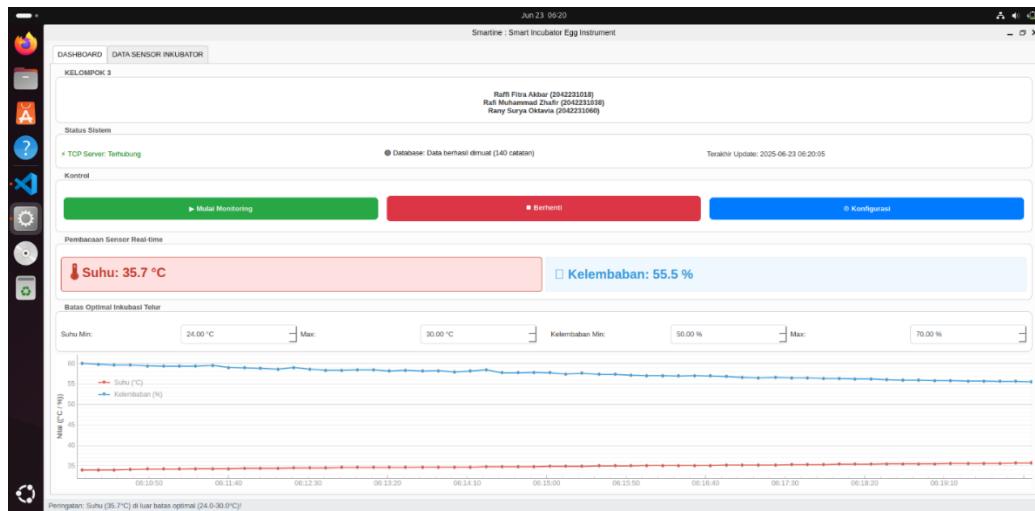


Gambar 12. Dashboard influxdb

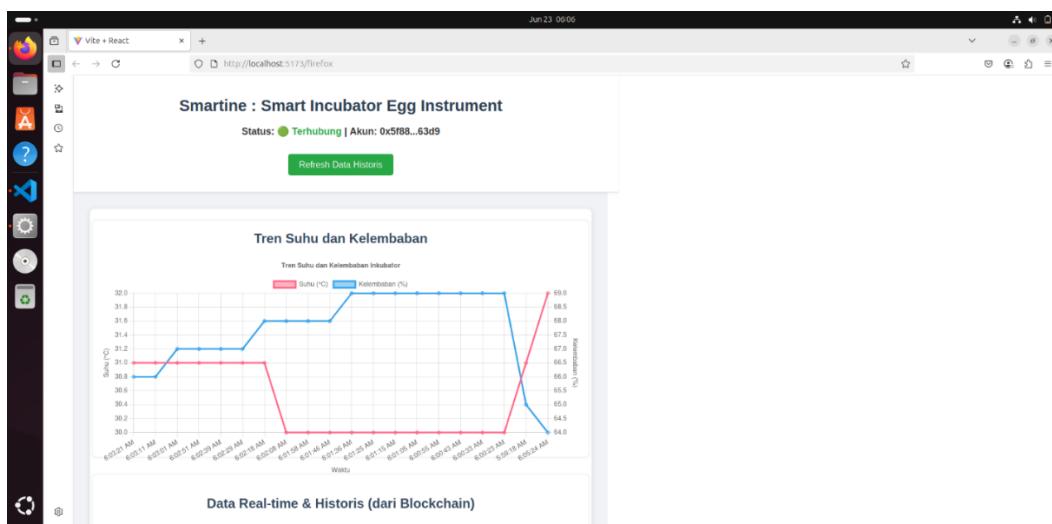
Gambar tersebut menyajikan dokumentasi hasil pembacaan suhu dan kelembaban selama proses inkubasi yang berlangsung. Data tersebut ditampilkan dalam bentuk hasil penyimpanan di database time-series InfluxDB dan tampilan real-time dashboard pada platform visualisasi Grafana. Pada InfluxDB menunjukkan bahwa data berhasil direkam secara berkala dengan struktur timestamp, field value, dan tag yang merepresentasikan parameter suhu dan kelembaban dari sensor. Sementara itu, pada dashboard Grafana terlihat grafik dinamis yang merekam perubahan

nilai suhu dan kelembaban sepanjang waktu, memungkinkan pengguna untuk melakukan pemantauan secara langsung.

Berdasarkan analisis terhadap grafik dan data yang ditampilkan, nilai suhu selama proses inkubasi umumnya berada pada rentang 25–29 °C, sedangkan kelembaban berkisar antara 55–68%. Hal ini menunjukkan bahwa kedua parameter tersebut berada dalam rentang optimal yang direkomendasikan, yaitu 24–30 °C untuk suhu dan 50–70% untuk kelembaban. Kondisi ini dianggap stabil dan sesuai untuk mendukung proses inkubasi, sehingga peluang keberhasilan penetasan telur atau pertumbuhan biologis dalam ruang inkubasi dapat dikatakan tinggi.



Gambar 13. Grafik pembacaan Qt



Gambar 14. Halaman Web3 Smartine

## BAB 6. KESIMPULAN DAN REKOMENDASI

### 6.1 Kesimpulan

1. Proyek ini berhasil merancang dan mengimplementasikan sistem pemantauan suhu dan kelembaban berbasis Internet of Things (IoT) menggunakan sensor SHT20 dan protokol Modbus RTU yang diolah oleh Modbus client berbasis Rust. Data yang dibaca dikirimkan melalui TCP server dan disimpan dalam InfluxDB, lalu divisualisasikan secara real-time melalui dashboard Grafana dan antarmuka Qt.
2. Arsitektur sistem berhasil mengintegrasikan berbagai komponen secara modular dan berlapis, mulai dari sensor, pemrosesan data, penyimpanan basis data, hingga visualisasi pada platform web dan GUI Python. Penggunaan Rust memberikan efisiensi dan keandalan tinggi dalam komunikasi data TCP dan proses parsing JSON.
3. Berdasarkan hasil pengujian, suhu selama proses inkubasi berkisar antara 25–29 °C dan kelembaban 55–68%, yang keduanya berada dalam rentang optimal yang direkomendasikan untuk proses inkubasi telur (suhu 24–30 °C dan kelembaban 50–70%). Hal ini menunjukkan bahwa sistem mampu menjalankan fungsinya secara akurat dan konsisten dalam mendukung proses biologis.
4. Dashboard Grafana dan GUI Python Qt mampu menyajikan grafik waktu nyata yang memudahkan pengguna dalam memantau tren suhu dan kelembaban secara historis maupun saat ini, sekaligus memungkinkan interaksi berbasis Web3 melalui integrasi dengan Metamask.

### 6.2 Rekomendasi

1. Prosedur startup sistem yang saat ini memerlukan banyak terminal secara manual dapat diotomatisasi melalui penggunaan skrip shell (bash script) atau layanan seperti systemd agar sistem dapat berjalan otomatis saat booting atau dijalankan satu perintah saja.
2. Mengingat sistem sudah terhubung dengan Web3 dan blockchain (Metamask), maka sistem autentikasi, validasi data, dan keamanan komunikasi perlu diperkuat, seperti menggunakan HTTPS, autentikasi token, atau sistem logging untuk jejak audit.
3. Disarankan untuk menambahkan fitur notifikasi (seperti email, Telegram, atau WhatsApp API) apabila nilai suhu atau kelembaban keluar dari batas optimal, agar pengguna dapat segera melakukan tindakan korektif tanpa harus terus-menerus memantau dashboard.
4. Lakukan kalibrasi rutin terhadap sensor suhu dan kelembaban serta validasi silang dengan alat ukur standar untuk menjamin akurasi pembacaan sensor dari waktu ke waktu.
5. Sistem ini dapat dikembangkan lebih lanjut dengan mendukung banyak titik sensor (multi-node), edge computing, serta deployment ke server cloud agar mampu diterapkan dalam skala produksi peternakan yang lebih besar.

## **DAFTAR PUSTAKA**

- Agustian, M. (2019). *ANALISIS KINERJA SISTEM FAILOVER LINK*.
- Ariwibisono, F. X., Muljanto, W. P., & Pemanfaatan, A. (2023). *Terakreditasi SINTA 5 Implementasi Sistem Monitoring Produksi Energi PLTS Berbasis Protokol Modbus RTU Dan Modbus TCP* (Vol. 17). <https://journal.fkom.uniku.ac.id/ilkom>
- Cao, L. (2022). Decentralized AI: Edge Intelligence and Smart Blockchain, Metaverse, Web3, and DeSci. In *IEEE Intelligent Systems* (Vol. 37, Issue 3, pp. 6–19). Institute of Electrical and Electronics Engineers Inc. <https://doi.org/10.1109/MIS.2022.3181504>
- Chandra, S. (2016). *FINAL PROJECT-TE 141599 DESIGN AND IMPLEMENT AT ION OF MODBUS PROT OCOL FOR INT EGRAT ED QUEUE SYST EM IN DRIVING LICENCE SERVICE OF RESORT POLICE OFFICE*.
- Monitoring Suhu dan Pencahayaan Berbasis, S., Ariani, F., Yuli Endra, R., Erlangga, E., Aprlinda, Y., Reza Bahar, A., Studi Sistem Informasi, P., & Studi Informatika, P. (n.d.). *Jurnal Manajemen Sistem Informasi dan Teknologi Internet of Thing (IoT) untuk Penetasan Telur Ayam* (Vol. 10, Issue 2).
- Nisa, S., & Andreansyah, I. (2024). Mesin Penetas Telur Otomatis Berbasis Internet of Things. *Tahun*, 5(2). <https://ejournal.unuja.ac.id/index.php/core>
- Titin Nurfadhlila Sudirman. (2019). *PERANCANGAN DASHBOARD DAN QUERY*.
- Utomo, T. (2021). IMPLEMENTASI TEKNOLOGI BLOCKCHAIN DI. *Buletin Perpustakaan Universitas Islam Indonesia*, 4(2), 173–200.

## LAMPIRAN

[https://github.com/ransurya/ProjectInterkoneksi\\_SMARTINE](https://github.com/ransurya/ProjectInterkoneksi_SMARTINE)

[https://its.id/m/PPTProjectISISmartine\\_Kelompok3](https://its.id/m/PPTProjectISISmartine_Kelompok3)

The screenshot shows two tables of data from a blockchain application. The top table is titled "Data Real-time & Historis (dari Blockchain)" and the bottom table is titled "DATA SENSOR INKUBATOR". Both tables have columns: Waktu, Sensor ID, Lokasi Inkubator, Usia Telur (Hari), Fase Inkubasi, Suhu (°C), and Kelembaban (%).

**Data Real-time & Historis (dari Blockchain)**

Waktu	Sensor ID	Lokasi Inkubator	Usia Telur (Hari)	Fase Inkubasi	Suhu (°C)	Kelembaban (%)
6/23/2025, 6:05:24 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:05:13 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:05:03 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:04:53 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:04:43 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:04:33 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	64.0
6/23/2025, 6:04:23 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	32.0	65.0
6/23/2025, 6:04:13 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.0	65.0
6/23/2025, 6:04:03 AM	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.0	65.0

**DATA SENSOR INKUBATOR**

Waktu	Sensor ID	Lokasi Inkubator	Usia Telur (Hari)	Fase Inkubasi	Suhu (°C)	Kelembaban (%)
1 2025-06-22 22:22:24	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	65.50
2 2025-06-22 22:22:14	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	65.40
3 2025-06-22 22:22:04	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	65.40
4 2025-06-22 22:21:54	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.30
5 2025-06-22 22:21:43	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.30
6 2025-06-22 22:21:33	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.20
7 2025-06-22 22:21:23	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.20
8 2025-06-22 22:21:13	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.30
9 2025-06-22 22:21:03	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.30
10 2025-06-22 22:20:53	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.70
11 2025-06-22 22:20:42	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.50
12 2025-06-22 22:20:31	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.40
13 2025-06-22 22:20:21	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.70
14 2025-06-22 22:20:11	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.80
15 2025-06-22 22:20:00	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.60
16 2025-06-22 22:19:50	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.80
17 2025-06-22 22:19:40	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.80	65.80
18 2025-06-22 22:19:30	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	66.00
19 2025-06-22 22:19:20	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	65.90
20 2025-06-22 22:19:10	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	66.20
21 2025-06-22 22:18:59	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	66.20
22 2025-06-22 22:18:49	SHT20-Smartine	Inkubator 1	Hari ke-4	Inkubasi (dengan pembalikan rutin)	31.70	66.40