



SMARTINE

SMART INCUBATOR EGG INSTRUMENT SISTEM MONITORING SUHU DAN KELEMBABAN UNTUK INKUBATOR TELUR BERBASIS IOT DAN VISUALISASI REALTIME

Kelompok 3 :

Raffi Fitra Akbar

[2042231018]

Rafi Muhammad Zhafir

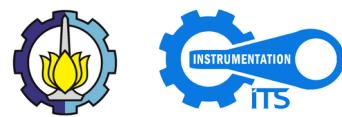
[2042231038]

Rany Surya Oktavia

[2042231060]

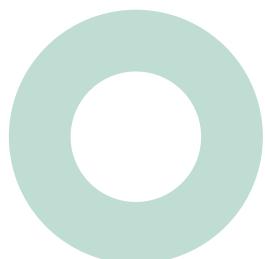


INSTITUT TEKNOLOGI
SEPULUH NOPEMBER



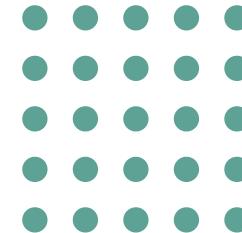
Latar Belakang

Permintaan daging ayam terus meningkat, namun proses penetasan telur secara manual kurang efisien dan bergantung pada indukan terbatas serta kondisi lingkungan. Mesin penetas telur otomatis berbasis IoT dikembangkan untuk menjaga suhu ideal (38-39°C) dan kelembapan (35-60%) secara stabil. Teknologi ini memungkinkan peternak memantau proses penetasan secara real-time melalui aplikasi mobile, sehingga meningkatkan keberhasilan penetasan, efisiensi kerja, dan kuantitas anak ayam unggul.



SMARTINE PRESENTED BY : KELOMPOK 3

02



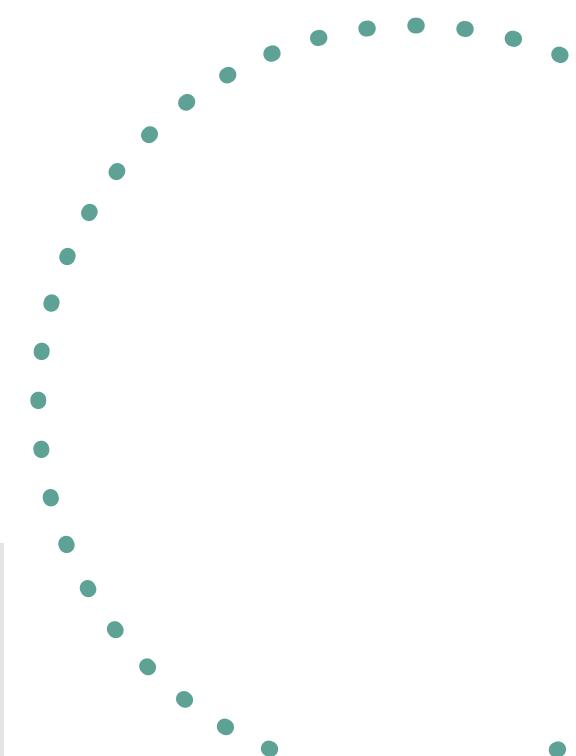
Rumusan Masalah



- 01. Rumusan Satu**
Bagaimana sistem monitoring suhu dan kelembapan pada alat penetasan telur secara real time dalam industri perunggasan rumahan?

- 02. Rumusan Dua**
Bagaimana inovasi sistem alat penetas telur ayam dapat meningkatkan efisiensi dan nilai ekonomi dalam industri perunggasan rumahan?

03



Tujuan Penelitian

01.

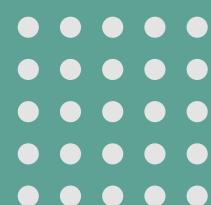
Tujuan Satu

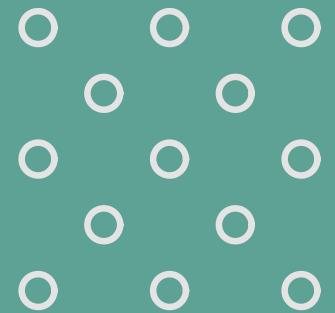
Untuk mengetahui sistem monitoring suhu dan kelembapan pada alat penetasan telur secara real-time dalam industri perunggasan rumahan.

02.

Tujuan Dua

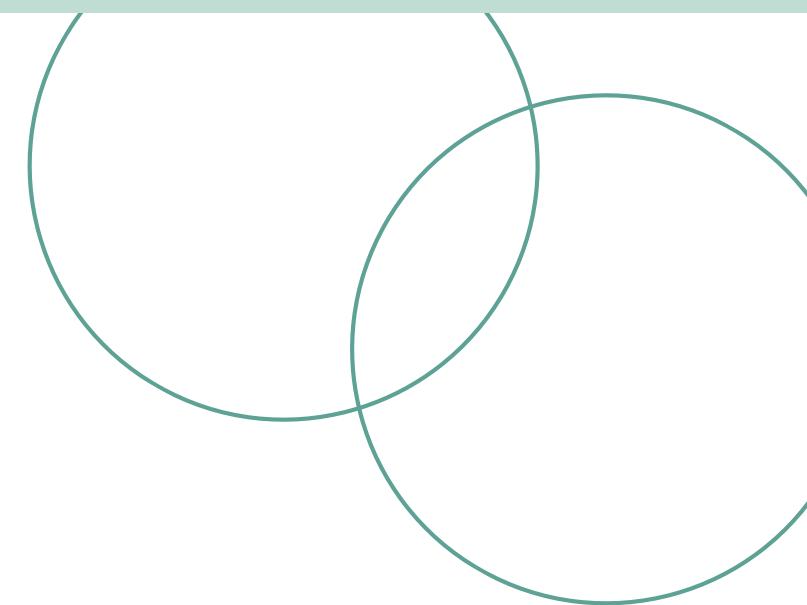
Untuk mengetahui inovasi sistem alat penetasan telur ayam dapat meningkatkan efisiensi dan nilai ekonomi dalam industri perunggasan rumahan.



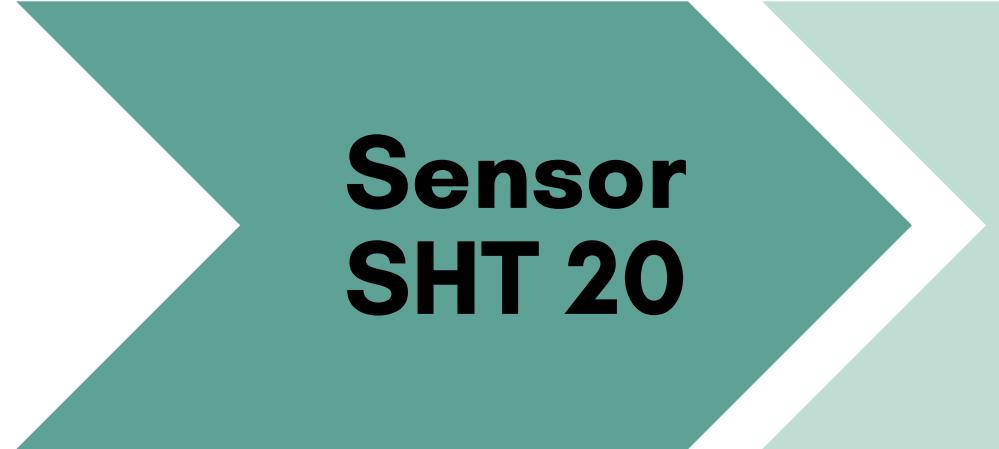


Landasan Teori

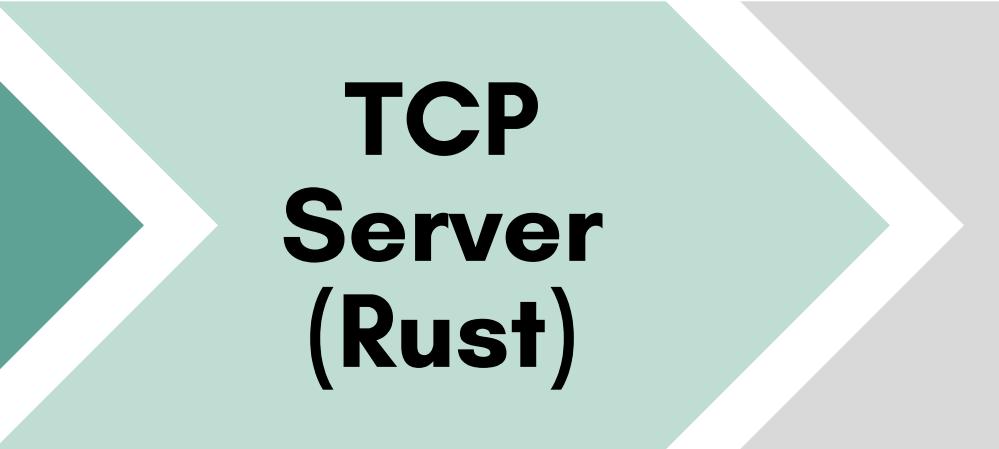
Topik, Penulis dan Tahun	Teknologi yang digunakan	Hasil
Sistem Monitoring Suhu Pada Inkubator Penetas Telur Berbasis IoT. Yunus et al. (2024)	Sensor DHT11, NodeMCU ESP8266, platform Blynk	Sistem efektif dalam meningkatkan kualitas penetasan telur dan efisiensi pemantauan
Monitoring Inkubator Telur Menggunakan 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	Sensor DHT11, NodeMCU ESP8266, OLED I2C, MQTT, dashboard web	Error suhu 1,7%, kelembapan 2,1%, sistem stabil dan efisien



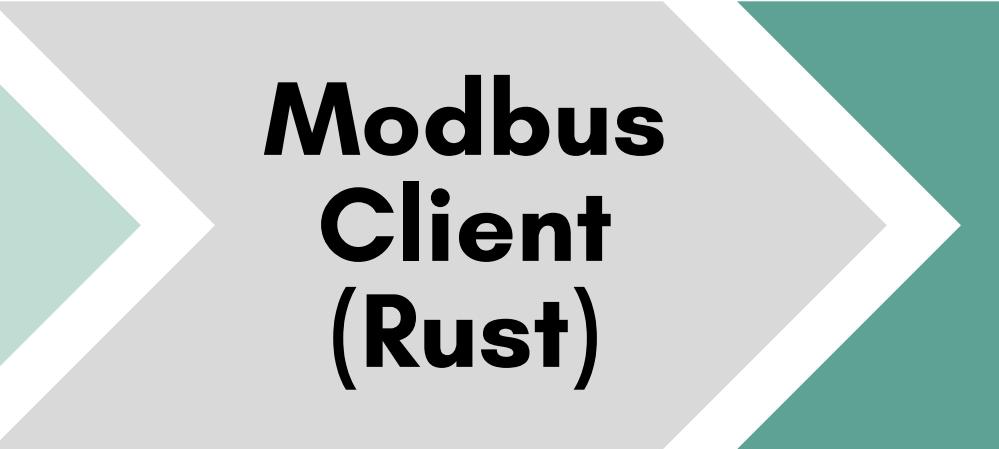
Arsitektur Sistem



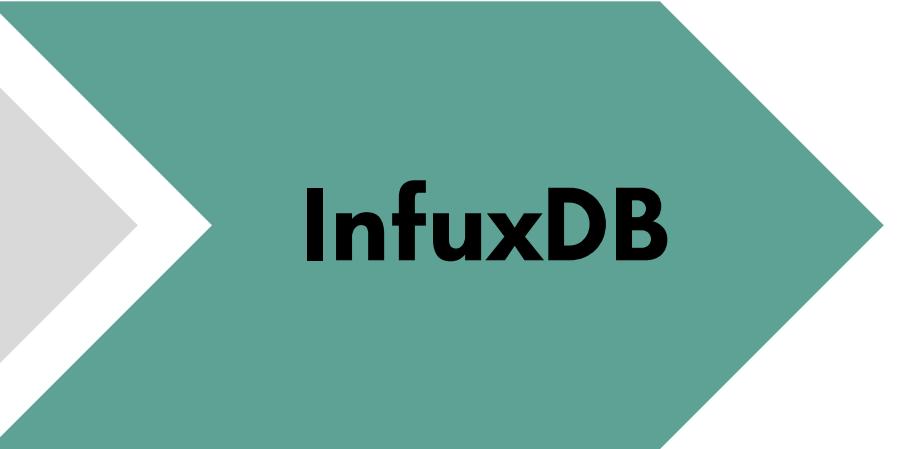
**Sensor
SHT 20**



**TCP
Server
(Rust)**



**Modbus
Client
(Rust)**



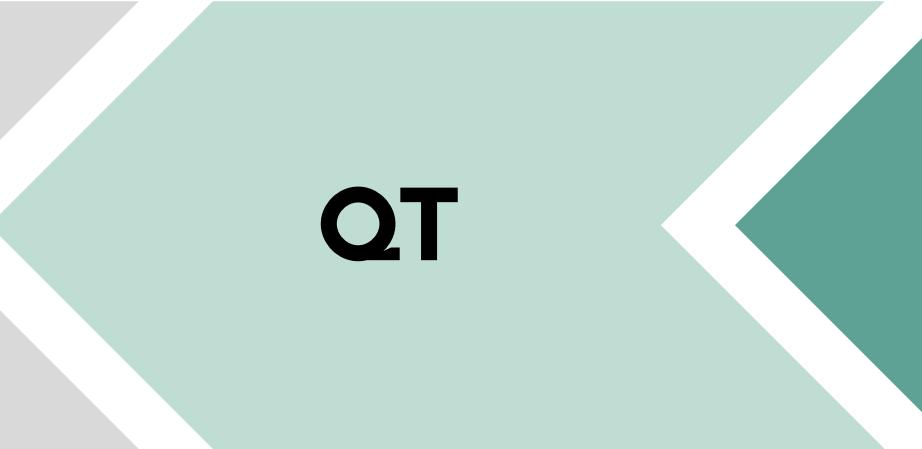
InfluxDB



Web3



Blockchain



QT

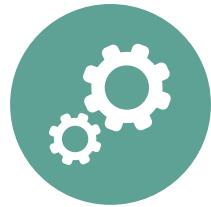


Grafana



Prosedur Operasional

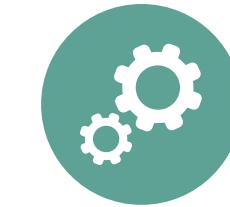
Komunikasi Sistem



1. menjalankan layanan berbasis Node.js
2. Buka terminal atau command prompt, lalu arahkan direktori kerja ke folder project Node menggunakan perintah cd Dokuments/ISI
3. jalankan program dengan perintah node index.js atau gunakan node start_dev_env jika script sudah didefinisikan dalam file package.json.
4. Jika sudah maka akan tertulis Hardhat node dan deploy selesai. Biarkan terminal ini terbuka untuk Hardhat node
5. 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/

Prosedur Operasional

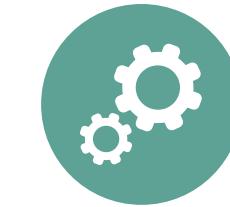
Komunikasi Sistem



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
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/.

Prosedur Operasional

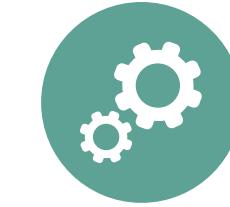
Komunikasi Sistem



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
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

Prosedur Operasional

Komunikasi Sistem



19. aktifkan mode pengembangan dengan perintah node npm run dev
20. Dari local host yang tertera, buka firefox dan search laman <http://localhost:5173/>
21. Jika ingin menghubungkan ke metamask, klik pada extension di dalam firefox lalu klik pada Metamask untuk menghubungkan blockchain
22. Setelah itu klik pada tampilan kiri, Lalu klik 'add a custom network'
23. Lalu sesuaikan dengan localhost dan klik save
24. 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
25. 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

IMPLEMENTASI DAN KODE PROGRAM

4.1 Rust Modbus Client

```

use chrono::Local, SecondsFormat; // Ubah dari Utc ke Local
use tokio_modbus::{client::rtu, 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<dyn Error>> {
    let port = tokio_serial::new("/dev/ttyUSB0", 9600)
        .parity(tokio_serial::Parity::None)
        .stop_bits(tokio_serial::StopBits::One)
        .data_bits(tokio_serial::DataBits::Eight)
        .timeout(Duration::from_secs(1));

    let port = SerialStream::open(&port)?;
    let slave = Slave(slave);

    let response = {
        let mut ctx = rtu::attach_slave(port, slave);
        ctx.read_input_registers(1, 2).await?
    };

    Ok(response)
}

async fn send_to_server(

```

```

Open ▾ Cargo.toml Cargo.toml main.rs Cargo.toml

async fn send_to_server(
    sensor_id: &str,
    location: &str,
    process_stage: &str,
    temperature: f32,
    humidity: f32,
    timestamp: chrono::DateTime<Local>, // Ubah dari Utc ke 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(())
}

```

IMPLEMENTASI DAN KODE PROGRAM

4.1 Rust Modbus Client



Pembacaan sensor SHT20 menggunakan Modbus RTU dilakukan melalui komunikasi serial dengan library `tokio_modbus` dan `tokio_serial`. Sensor terhubung ke port serial `/dev/ttyUSB0` dengan konfigurasi standar (baudrate 9600, parity none, 1 stop bit, 8 data bit). Proses dimulai dengan membuka koneksi serial, menetapkan alamat slave, lalu membuat koneksi Modbus dengan `rtu::attach_slave`. Program membaca dua register (suhu dan kelembaban) dari alamat 1, lalu mengonversi nilainya ke format desimal. Seluruh proses berjalan secara asinkron dan berulang tiap 10 detik.

IMPLEMENTASI DAN KODE PROGRAM

Program untuk mengirim data ke TCP server

Data sensor dikirim ke TCP server lokal (127.0.0.1:7878) setiap 10 detik menggunakan fungsi `send_to_server`. Setelah data suhu dan kelembaban dibaca, fungsi ini dipanggil dengan tambahan ID sensor, lokasi, tahap proses, dan timestamp. Koneksi TCP dibuka dengan `TcpStream::connect`, lalu data disusun dalam format JSON menggunakan `serde_json::json!` dan dikirim lewat `stream.write_all`. Respons dari server dibaca dan ditampilkan di konsol, memastikan komunikasi berlangsung sukses dan periodik.

```
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?;
}
```

IMPLEMENTASI DAN KODE PROGRAM

4.2 Rust TCP Server

Server TCP menerima data JSON dari client di alamat 127.0.0.1:7878 dan menyimpannya ke InfluxDB. Setelah koneksi terbuka, data dibaca dari soket, dikonversi ke string, lalu diparsing menjadi struktur SensorData. Timestamp dalam format ISO 8601 diubah menjadi nanodetik, lalu disusun sebagai DataPoint lengkap dengan tag dan field. DataPoint ini dikirim ke InfluxDB melalui koneksi yang dibuat saat inisialisasi program. Server mengirim respons "OK" jika berhasil, atau pesan error jika gagal.

```
Cargo.toml | Cargo.toml | main.rs

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> {
    // Configure InfluxDB connection
    let influx_url = "http://localhost:8086";
    let influx_org = "Institute Teknologi Sepuluh Nopember";
    let influx_token = "hkyHqP236AVjN2JL84XYNJVPHCDXnC756c1Gxc2C06n_nkG-dCTLB3IK-c761g3YRqa";
    let influx_bucket = "coffe_monitoring_db";

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

IMPLEMENTASI DAN KODE PROGRAM

4.2 Rust TCP Server

```
#![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 =
        "hkyHqP236AVjN2JL84XYNJVPWCDXnC756c1Gxc2CO6n_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");

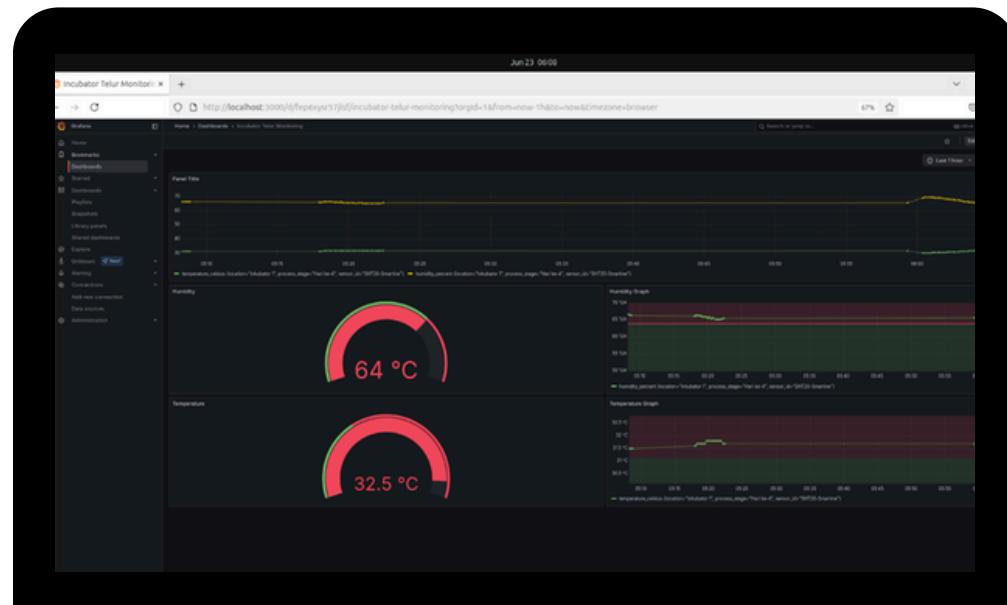
    loop {
        let (mut socket, _) = listener.accept().await?;
        let client = client.clone();
        let bucket = influx_bucket.to_string();
    }
}
```

```
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[0..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);

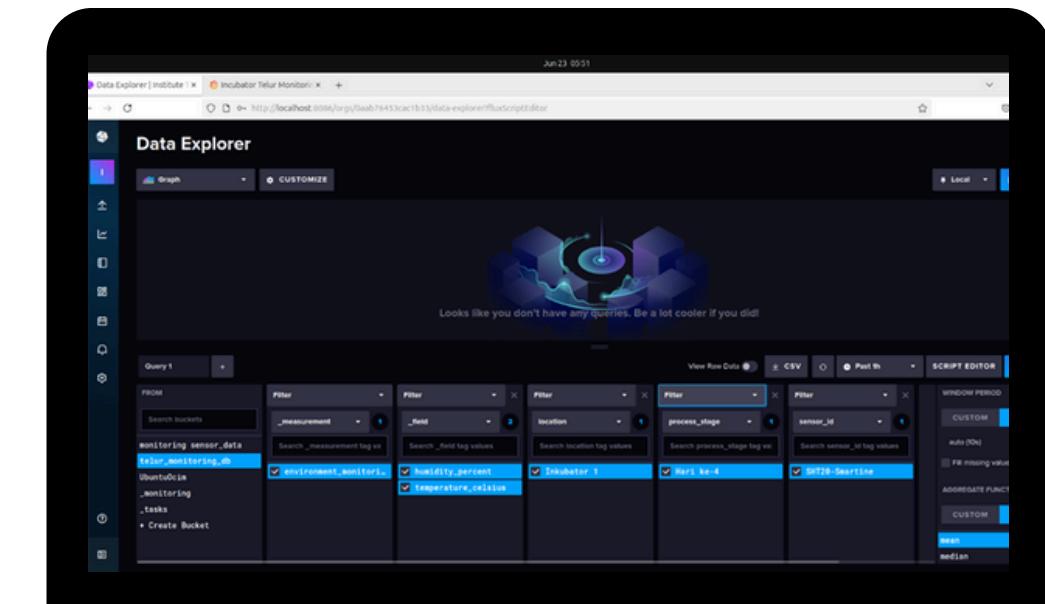
            match serde_json::from_str<SensorData>(data) {
                Ok(sensor_data) => {
                    // Parsing timestamp dan menyusun DataPoint
                    let timestamp = match
                        DateTime::parse_from_rfc3339(&sensor_data.timestamp) {
                            Ok(dt)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

PENGUJIAN DAN HASIL



Dashboard Grafana

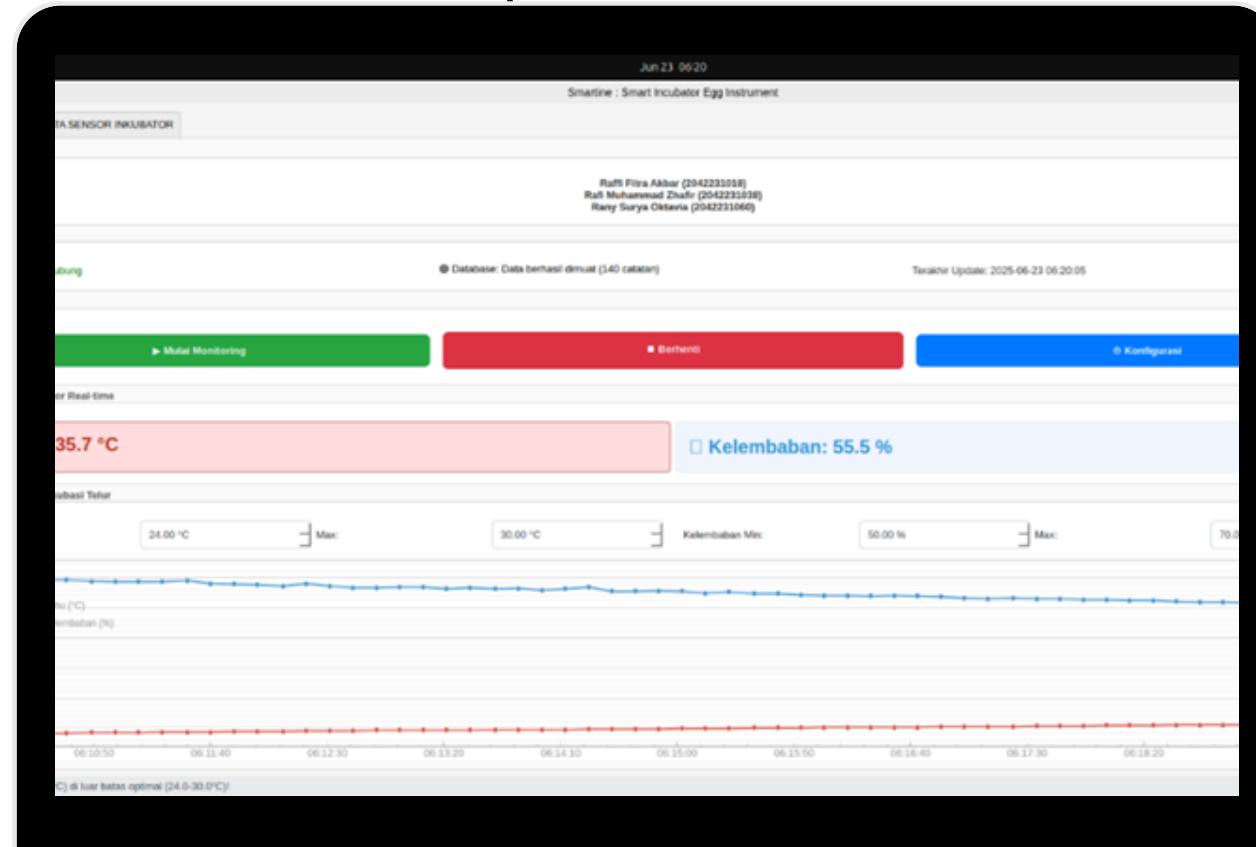
Dashboard Grafana menampilkan grafik dinamis yang merekam perubahan suhu dan kelembaban secara real-time, memungkinkan pengguna memantau kondisi inkubasi secara langsung dan historis dengan tampilan yang informatif dan interaktif.



Dashboard influxdb

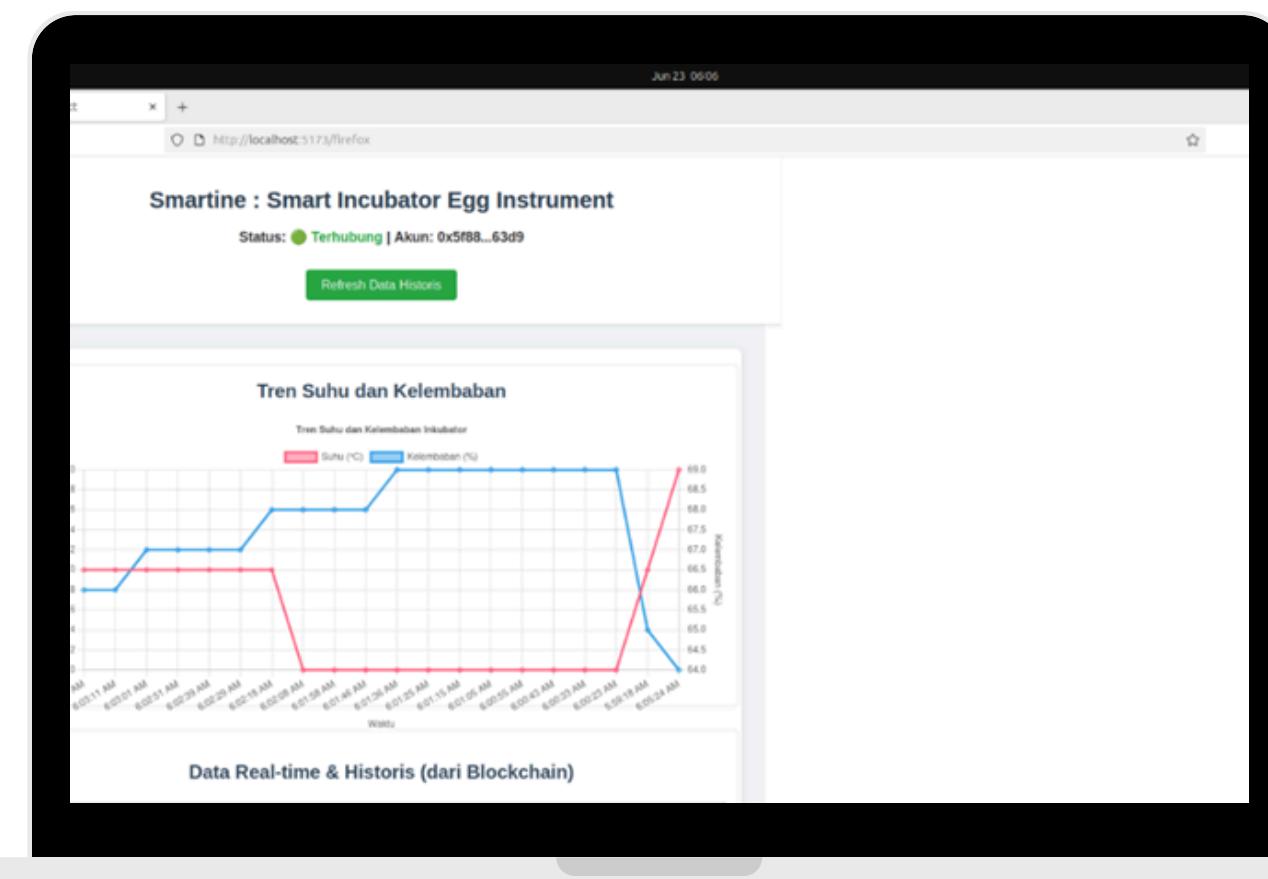
Data suhu dan kelembaban selama proses inkubasi berhasil direkam secara berkala dalam InfluxDB dengan struktur timestamp, field value, dan tag yang merepresentasikan parameter dari sensor. Ini menunjukkan sistem mampu menyimpan data time-series secara akurat dan terstruktur.

Grafik pembacaan Qt



Stabilnya kondisi suhu dan kelembaban menunjukkan bahwa sistem mendukung lingkungan yang sesuai untuk proses biologis, sehingga peluang keberhasilan penetasan telur atau pertumbuhan dalam ruang inkubasi dapat dikatakan tinggi.

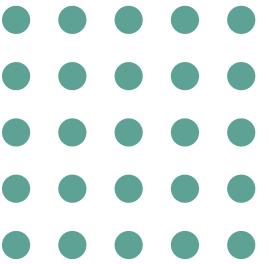
Selama proses inkubasi, suhu tercatat antara 25–29 °C dan kelembaban antara 55–68%, yang berada dalam rentang optimal yang direkomendasikan (24–30 °C untuk suhu dan 50–70% untuk kelembaban).



Halaman Web3 Smartine

Kesimpulan

Proyek ini berhasil merancang sistem pemantauan suhu dan kelembaban berbasis IoT menggunakan sensor SHT20 dan protokol Modbus RTU, dengan pengolahan data oleh Modbus client berbasis Rust. Data dikirim melalui TCP server, disimpan di InfluxDB, dan divisualisasikan secara real-time melalui Grafana dan antarmuka Qt. Arsitektur sistem terintegrasi secara modular dari sensor hingga visualisasi, dengan Rust memberikan efisiensi tinggi dalam komunikasi dan parsing data. Pengujian menunjukkan sistem bekerja akurat dan konsisten dalam mendukung inkubasi telur, dengan suhu dan kelembaban berada dalam rentang optimal. Visualisasi melalui Grafana dan Qt memudahkan pemantauan tren data serta mendukung interaksi Web3 melalui integrasi Metamask.



Referensi Sumber Penelitian

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 IMPLEMENTATION OF MODBUS PROTOCOL FOR INTERNET OF DRIVING LICENCE SERVICE OF RESORT POLICE OFFICE.

Monitoring Suhu dan Pencahayaan Berbasis, S., Ariani, F., Yuli Endra, R., Erlangga, E., Aprilinda, 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 Nurfadila Sudirman. (2019). PERANCANGAN DASHBOARD DAN QUERY.

Utomo, T. (2021). IMPLEMENTASI TEKNOLOGI BLOCKCHAIN DI. Buletin Perpustakaan Universitas Islam Indonesia, 4(2), 173–200.



Terima Kasih

