

LAPORAN TUGAS 4 KELOMPOK

Monitoring Suhu dan Kelembaban dalam Fermentasi Kakao untuk Menjaga Kualitas Cokelat

I. Identitas Kelompok

Nama Kelompok: Kelompok 5

- Anggota Kelompok:

- Aulia Zakhrine Ramadhani Setiawan (2042231042)
- Fadillah Wahyu Anggraini (2042231052)
- Fortunia Putri Syahari (2042241078)

- Dosen Pengampu: Ahmad Radhy, S.Si., M.Si.

- Mata Kuliah: Interkoneksi Sistem Instrumentasi

II. Latar Belakang



Fermentasi biji kakao merupakan tahap krusial dalam proses pasca-panen yang secara langsung memengaruhi mutu akhir produk cokelat. Kualitas aroma, rasa, dan warna cokelat sangat ditentukan oleh keberhasilan proses fermentasi yang berlangsung selama beberapa hari. Salah satu faktor penting yang menentukan keberhasilan fermentasi adalah kondisi mikroklimat, terutama suhu dan kelembaban di dalam wadah fermentasi. Suhu yang terlalu rendah dapat memperlambat aktivitas mikroba fermentatif, sedangkan suhu yang terlalu tinggi dapat membunuh mikroorganisme bermanfaat. Demikian pula, kelembaban yang tidak terkendali dapat menghambat proses biokimia dan menurunkan mutu biji kakao yang dihasilkan.

Namun, di berbagai sentra produksi kakao, pemantauan terhadap parameter lingkungan ini masih banyak dilakukan secara manual. Petani atau pelaku fermentasi biasanya mengandalkan pengamatan harian menggunakan termohigrometer analog atau bahkan berdasarkan pengalaman semata. Pendekatan ini tidak hanya menyita waktu dan tenaga, tetapi juga rawan terhadap kesalahan pencatatan dan tidak mampu memberikan informasi kondisi fermentasi secara real-time. Selain itu, tidak tersedianya data historis menjadikan proses evaluasi dan perbaikan fermentasi menjadi sulit dilakukan secara objektif.

Untuk mengatasi permasalahan ini, dibutuhkan sebuah sistem monitoring otomatis berbasis sensor yang dapat merekam suhu dan kelembaban secara kontinu selama proses fermentasi berlangsung. Sistem ini idealnya mampu menyimpan data secara historis dan menampilkannya dalam bentuk visualisasi grafik melalui dashboard digital. Dengan adanya sistem monitoring semacam ini, pelaku fermentasi dapat melakukan pengambilan keputusan secara cepat dan tepat, seperti menentukan waktu pembalikan biji atau penyesuaian kondisi lingkungan. Hal ini akan membantu menjaga konsistensi fermentasi, meminimalkan cacat rasa, dan pada akhirnya meningkatkan kualitas dan nilai jual produk coklat yang dihasilkan.

III. Rumusan Masalah

Dari judul yang saya ambil mengenai Sistem Monitoring Suhu dan Kelembaban dalam Fermentasi Kakao untuk Menjaga Kualitas Cokelat, sebagai berikut

1. Bagaimana merancang sistem monitoring suhu dan kelembaban berbasis sensor industri (SHT20) yang dapat berkomunikasi menggunakan protokol Modbus RTU dan diintegrasikan dengan server melalui jaringan TCP/IP untuk mendukung proses fermentasi biji kakao?
2. Bagaimana mengimplementasikan sistem monitoring suhu dan kelembaban yang dapat merekam data secara kontinu selama proses fermentasi biji kakao, serta menjamin akurasi dan kestabilan pengukuran di lingkungan fermentasi?
3. Bagaimana menyimpan data suhu dan kelembaban dalam format time-series menggunakan InfluxDB secara efisien, dan menampilkannya dalam bentuk visualisasi real-time menggunakan Grafana untuk memantau kualitas proses fermentasi?
4. Bagaimana membangun aplikasi desktop berbasis Qt yang dapat digunakan untuk monitoring suhu dan kelembaban secara lokal dengan tampilan antarmuka yang intuitif dan mendukung pengambilan keputusan dalam proses fermentasi biji kakao?

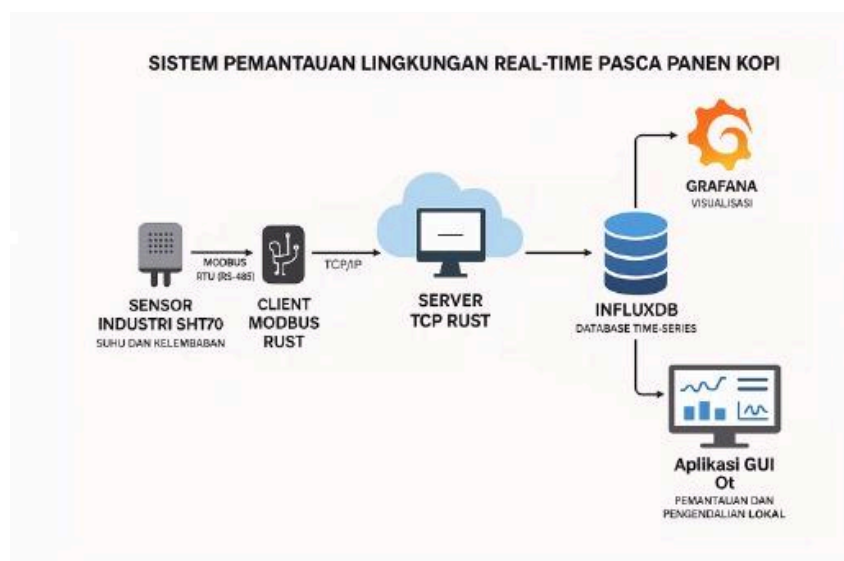
IV. Tujuan Proyek

Berikut tujuan dari sistem Monitoring Suhu dan Kelembaban dalam Fermentasi Kakao untuk Menjaga Kualitas Cokelat

1. Merancang dan mengimplementasikan sistem monitoring suhu dan kelembaban berbasis sensor industri SHT20 yang menggunakan protokol komunikasi Modbus RTU untuk memperoleh data fermentasi biji kakao secara akurat dan andal.
2. Mengembangkan sistem komunikasi client-server berbasis protokol TCP/IP yang mampu mentransmisikan data suhu dan kelembaban dari perangkat akuisisi menuju server secara real-time selama proses fermentasi berlangsung.
3. Membangun arsitektur penyimpanan data berbasis time-series menggunakan InfluxDB, guna mencatat data suhu dan kelembaban secara efisien, berkelanjutan, dan mendukung proses evaluasi kualitas fermentasi.
4. Menyediakan visualisasi data secara real-time dan historis yang informatif dan mudah dipahami melalui platform Grafana untuk mendukung pengambilan keputusan dalam pemantauan fermentasi.

V. Metodologi dan Arsitektur Sistem

1. Desain Arsitektur Sistem



Sistem ini dirancang untuk memantau suhu dan kelembaban secara real-time di ruang fermentasi biji kakao menggunakan sensor industri dan teknologi berbasis komunikasi data terstruktur. Sensor SHT20 digunakan untuk mengukur suhu dan kelembaban, serta berkomunikasi melalui protokol Modbus RTU (RS-485) untuk mengirimkan data ke Modbus Client yang dikembangkan menggunakan bahasa Rust. Data dari sensor kemudian diteruskan melalui jaringan menggunakan protokol TCP/IP ke TCP Server Rust, yang bertugas menerima, memproses (parsing), dan menyimpan data dalam format time-series ke dalam basis data InfluxDB.

Data yang telah tersimpan dapat divisualisasikan secara real-time melalui platform Grafana, sehingga operator atau pengelola proses fermentasi dapat memantau kondisi lingkungan fermentasi biji kakao secara langsung melalui tampilan dashboard. Selain itu, sistem ini dilengkapi dengan aplikasi GUI berbasis Qt yang dapat digunakan untuk pemantauan dan pengendalian secara lokal di perangkat komputer, memberikan fleksibilitas dalam pemantauan baik secara langsung di lokasi maupun dari ruang kontrol.

Arsitektur sistem ini dirancang untuk bekerja secara otomatis, akurat, dan efisien guna mendukung proses fermentasi biji kakao yang optimal. Informasi suhu dan kelembaban yang tercatat secara kontinu membantu pengambilan keputusan berdasarkan kondisi aktual dan historis, seperti penentuan waktu pembalikan biji atau evaluasi mutu fermentasi, sehingga berkontribusi terhadap peningkatan kualitas dan nilai jual produk kakao.

2. Deskripsi Komponen

- Sensor: SHT20 (industrial) dengan komunikasi Modbus RTU.
- Modbus Client (Rust): Pembacaan data suhu dan kelembaban dari sensor.
- TCP Server (Rust): Menerima data JSON, parsing, dan simpan ke InfluxDB.
- InfluxDB: Menyimpan data time-series.
- Grafana: Menampilkan dashboard suhu dan kelembaban.
- Web3 : untuk menampilkan data transparansi dari transaksi yang dihubungkan dengan metamask

3. Format Payload

Format JSON

```
data: {  
  labels: [],  
  datasets: [  
    {  
      label: 'Temperature (°C)',  
      data: [],  
      borderColor: 'rgba(255,99,132,1)',  
      fill: false  
    },  
    {  
      label: 'Humidity (%)',  
      data: [],
```

```

borderColor: 'rgba(54,162,235,1)',

fill: false

}

```

VI. Implementasi dan Kode Program

1. Kode Rust Modbus Client

Cuplikan kode utama dan cara kirim data ke TCP Server.

```

putri@putri:~/modbus_server$ cargo run
Finished dev profile [unoptimized + debuginfo] target(s) in 0.35s
Running target/debug/sht20
[2025-05-27 19:38:41] Gudang Fermentasi 1 - Fermentasi: Temp=26.5°C, RH=63.4%
Sending JSON: {"humidity_percent":63.40000152587896,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":26.5,"timestamp":"2025-05-27T19:38:41+07:00"}
Server response: OK
[2025-05-27 19:38:52] Gudang Fermentasi 1 - Fermentasi: Temp=25.4°C, RH=56.6%
Sending JSON: {"humidity_percent":56.599998474121894,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":25.399999618530273,"timestamp":"2025-05-27T19:38:52+07:00"}
Server response: OK
[2025-05-27 19:39:02] Gudang Fermentasi 1 - Fermentasi: Temp=25.4°C, RH=56.6%
Sending JSON: {"humidity_percent":56.599998474121894,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":25.399999618530273,"timestamp":"2025-05-27T19:39:02+07:00"}
Server response: OK

```

2. Kode Rust TCP Server

Pada sistem pemantauan lingkungan untuk fermentasi biji kakao, peran Rust TCP Server sangat vital sebagai penghubung antara perangkat pembaca sensor (Modbus Client) dan sistem penyimpanan data time-series di InfluxDB. Server ini dibangun menggunakan pustaka Tokio untuk mendukung pemrosesan asynchronous, sehingga mampu menangani beberapa koneksi secara bersamaan tanpa blocking. Hal ini sangat penting dalam sistem monitoring yang beroperasi secara kontinu selama proses fermentasi. Data yang dikirimkan dari client dikemas dalam format JSON, yang memuat informasi seperti suhu, kelembaban, ID sensor, lokasi ruang fermentasi, dan fase fermentasi (contoh: hari ke-1, hari ke-2, pembalikan, akhir fermentasi). Setiap data yang diterima akan di-deserialize menjadi struktur data `SensorData` menggunakan pustaka `serde_json`.

Setelah parsing berhasil, data tersebut diubah menjadi objek `DataPoint`, lengkap dengan tag (seperti `sensor_id`, `location`, dan `fermentation_stage`) serta field (seperti `temperature_celsius` dan `humidity_percent`) untuk kemudian disimpan ke dalam bucket InfluxDB bernama SHT20. Proses ini juga mencakup konversi timestamp dari format RFC3339 ke format nanodetik, agar sesuai dengan standar penyimpanan di InfluxDB. Jika proses penulisan ke database berhasil, server akan mengirimkan respons "OK" ke client sebagai tanda keberhasilan. Namun, jika terjadi kesalahan baik dalam parsing, format data, maupun saat menyimpan ke InfluxDB, server akan mengirimkan pesan error yang deskriptif untuk memudahkan proses debugging. Dengan desain seperti ini, sistem monitoring dapat berjalan secara real-time, menjaga kestabilan suhu dan kelembaban yang krusial untuk keberhasilan fermentasi biji kakao, serta menyimpan data historis yang dapat diakses melalui visualisasi pada dashboard Grafana. Sistem ini tidak hanya mendukung efisiensi operasional dalam pengelolaan fermentasi, tetapi juga memungkinkan pengambilan keputusan berbasis data untuk meningkatkan konsistensi dan kualitas produk kakao.

Cuplikan kode

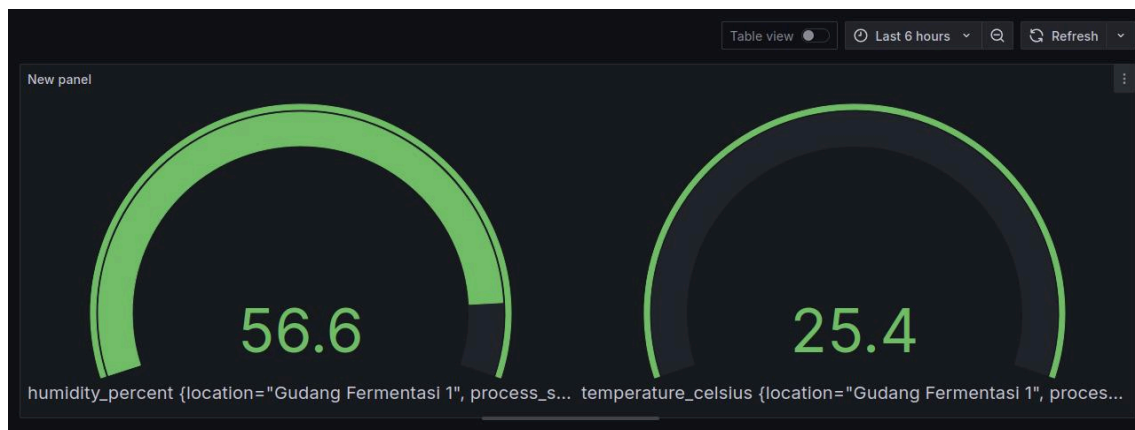
```
use influxdb2::{Client, models::DataPoint};
```

Berikut adalah tampilan terminal saat sensor terbaca dan terhubung dengan tcp server

```
putri@putri:~/modbus_servers$ cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.35s
    Running `target/debug/sht20`
[2025-05-27 19:38:41] Gudang Fermentasi 1 - Fermentasi: Temp=26.5°C, RH=63.4%
Sending JSON: {"humidity_percent":63.400001525878906,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":26.5,"timestamp":"2025-05-27T19:38:41+07:00"}
Server response: OK
[2025-05-27 19:38:52] Gudang Fermentasi 1 - Fermentasi: Temp=25.4°C, RH=56.6%
Sending JSON: {"humidity_percent":56.599998474121094,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":25.399999610530273,"timestamp":"2025-05-27T19:38:52+07:00"}
Server response: OK
[2025-05-27 19:39:02] Gudang Fermentasi 1 - Fermentasi: Temp=25.4°C, RH=56.6%
Sending JSON: {"humidity_percent":56.599998474121094,"location":"Gudang Fermentasi 1","process_stage":"Fermentasi","sensor_id":"SHT20-PascaPanen-001","temperature_celsius":25.399999610530273,"timestamp":"2025-05-27T19:39:02+07:00"}
Server response: OK
```

3. Dashboard Grafana

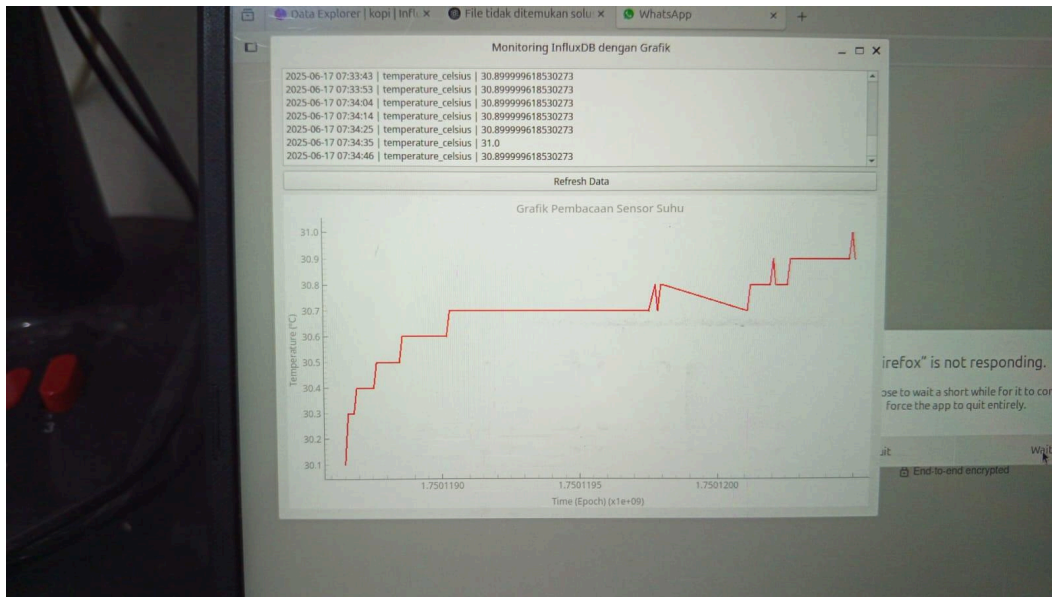
- Desain visual



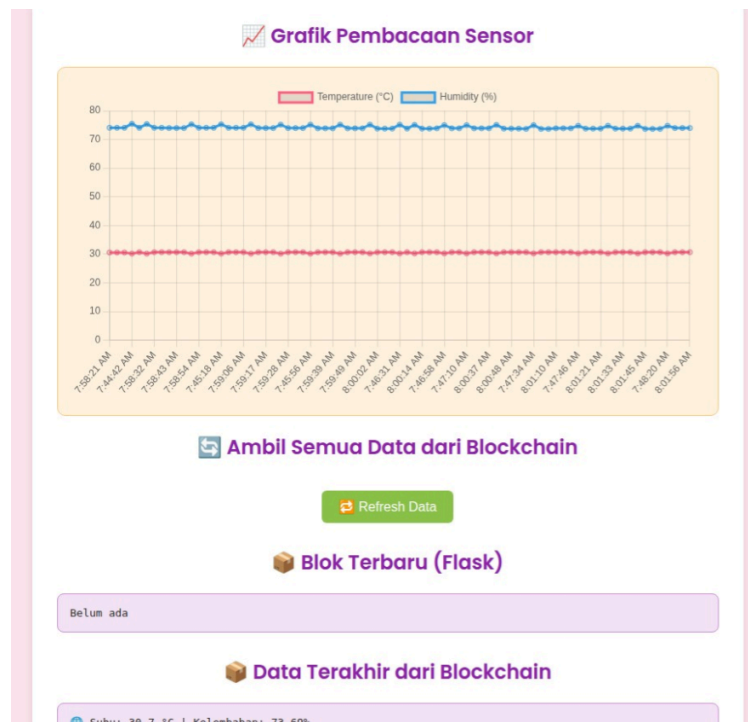
Pada tampilan dashboard digrafana sudah langsung secara real time memberikan hasil yang aktual, dan juga sudah dapat menampilkan

VII. Pengujian dan Hasil

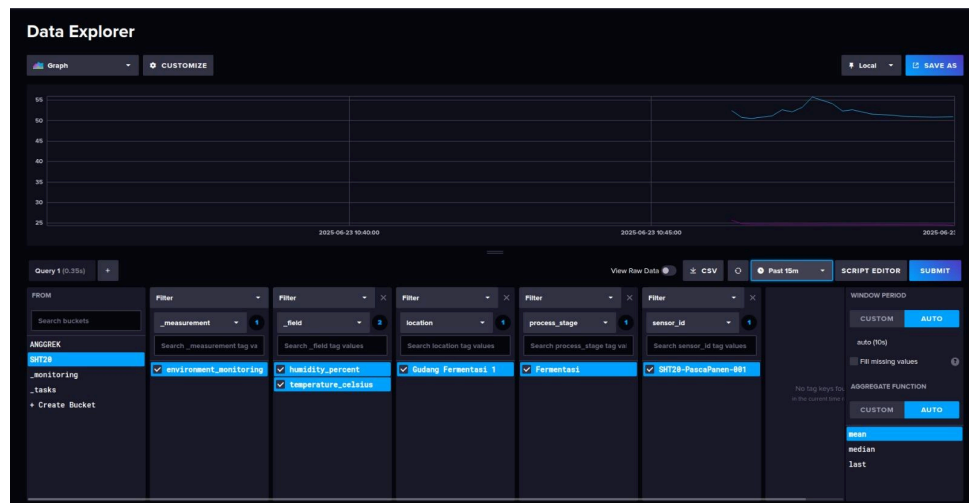
- Hasil Tampilan pada Qt



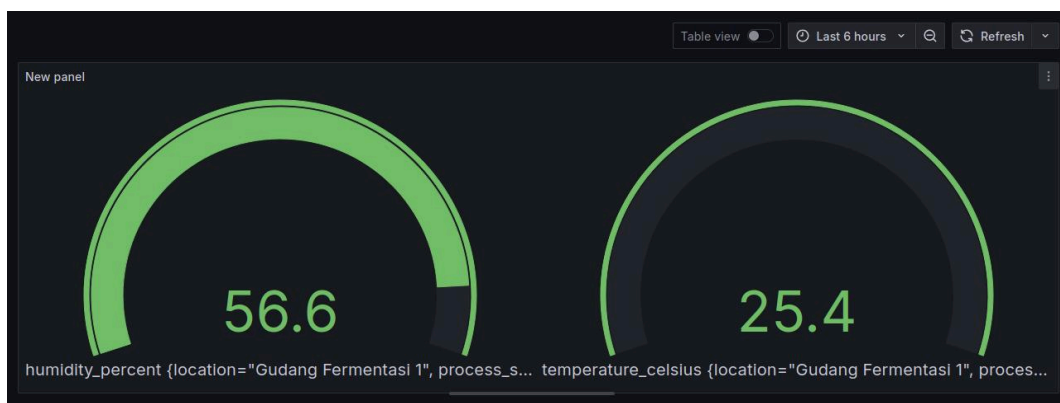
- Hasil Tampilan pada web3 yang telah terhubung dengan blockchain (sudah ter-update secara otomatis)



- Screenshot hasil penyimpanan di InfluxDB.



- Screenshot real-time dashboard di Grafana.



VIII. Kesimpulan dan Rekomendasi

Proyek ini berhasil merancang dan mengimplementasikan sistem monitoring suhu dan kelembaban untuk mendukung proses fermentasi biji kakao secara lebih akurat, efisien, dan real-time. Sistem yang dikembangkan terdiri dari sensor industri SHT20 yang terhubung melalui protokol Modbus RTU, client Modbus yang mengakuisisi data, serta TCP Server berbasis bahasa Rust yang mampu menerima dan memproses data secara asynchronous menggunakan pustaka Tokio. Data lingkungan fermentasi yang dikirim dalam format JSON berhasil diparsing menjadi struktur data yang sesuai, kemudian dikonversi menjadi format Line Protocol untuk disimpan secara efisien dalam basis data time-series InfluxDB. Melalui integrasi dengan Grafana, sistem ini mampu menampilkan visualisasi suhu dan kelembaban secara real-time dan historis dalam bentuk dashboard, sehingga sangat membantu pengelola dalam memantau kestabilan kondisi fermentasi biji kakao.

Aplikasi desktop berbasis Qt yang dikembangkan juga memungkinkan pemantauan secara lokal di area fermentasi, tanpa perlu bergantung pada koneksi internet, sehingga meningkatkan fleksibilitas dan keandalan sistem secara keseluruhan. Dengan implementasi sistem ini, proses fermentasi dapat dikontrol dengan lebih baik, yang pada akhirnya berkontribusi terhadap peningkatan mutu dan konsistensi produk kakao yang dihasilkan.

IX. Daftar Pustaka

- Sensirion AG. (2020). *SHT2x (SHT20) Datasheet – Humidity and Temperature Sensor IC*.
Afoakwa, E. O. (2014). *Cocoa Production and Processing Technology*. CRC Press.
Schwan, R. F., & Wheals, A. E. (2004). The microbiology of cocoa fermentation and its role in chocolate quality. *Critical Reviews in Food Science and Nutrition*, 44(4), 205–221.

X. Lampiran

- Listing kode penuh

1. sensor blockchain (app.js)

```
let web3;

let contract;

let userAccount;

const contractAddress =
"0x8C7e2BA424569c0B1c90bF5dAa66e45a1Cba058E";

const contractABI = [
  {
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "anonymous": false,
    "inputs": [
      { "indexed": true, "internalType": "uint256", "name": "id",
        "type": "uint256" },
```

```

        { "indexed": false, "internalType": "int256", "name":
"temperature", "type": "int256" },

        { "indexed": false, "internalType": "uint256", "name":
"humidity", "type": "uint256" },

        { "indexed": false, "internalType": "uint256", "name":
"timestamp", "type": "uint256" }

    ],

    "name": "DataLogged",

    "type": "event"
},

{

    "inputs": [{ "internalType": "uint256", "name": "index",
"type": "uint256" }],

    "name": "getReading",

    "outputs": [

        {

            "components": [

                { "internalType": "uint256", "name": "timestamp",
"type": "uint256" },

                { "internalType": "int256", "name": "temperature",
"type": "int256" },

                { "internalType": "uint256", "name": "humidity", "type":
"uint256" }

            ],

            "internalType": "struct SHT20Sensor.SensorData",

            "name": "",

            "type": "tuple"

        }

    ],

    "stateMutability": "view",

    "type": "function"
},

```

```

{
  "inputs": [],
  "name": "getTotalReadings",
  "outputs": [{ "internalType": "uint256", "name": "", "type":
"uint256" }],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    { "internalType": "int256", "name": "_temperature", "type":
"int256" },
    { "internalType": "uint256", "name": "_humidity", "type":
"uint256" }
  ],
  "name": "logData",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
}
];

```

```

async function connectMetamask() {
  if (window.ethereum) {
    try {
      const accounts = await ethereum.request({ method:
'eth_requestAccounts' });
      userAccount = accounts[0];
      document.getElementById('status').innerText = '●
Connected';

      web3 = new Web3(window.ethereum);

```

```

        contract = new web3.eth.Contract(contractABI,
contractAddress);

        getLastReading();

    } catch (error) {

        console.error("Metamask error:", error);

    }
} else {

    alert("Metamask tidak ditemukan");

}
}

const ctx =
document.getElementById('sensorChart').getContext('2d');

const sensorChart = new Chart(ctx, {

    type: 'line',

    data: {

        labels: [],

        datasets: [

            {

                label: 'Temperature (°C)',

                data: [],

                borderColor: 'rgba(255,99,132,1)',

                fill: false

            },

            {

                label: 'Humidity (%)',

                data: [],

                borderColor: 'rgba(54,162,235,1)',

```

```

        fill: false
    }
]
},
options: {
    responsive: true,
    scales: {
        y: { beginAtZero: true }
    }
}
});

```

```

function updateChart(temp, humid) {
    const timeLabel = new Date().toLocaleTimeString();
    sensorChart.data.labels.push(timeLabel);
    sensorChart.data.datasets[0].data.push(temp);
    sensorChart.data.datasets[1].data.push(humid);
    sensorChart.update();
}

```

```

async function sendData() {
    const sensorId = document.getElementById('sensorId').value;

    const temperature =
parseFloat(document.getElementById('temperature').value);

    const humidity =
parseFloat(document.getElementById('humidity').value);

    if (isNaN(temperature) || isNaN(humidity)) {
        alert("Masukkan nilai suhu dan kelembaban yang valid!");
        return;
    }
}

```

```

    }

    const response = await fetch('/add_block', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ sensor_id: sensorId, temperature,
        humidity })
    });

    const result = await response.json();

    document.getElementById('latestBlock').innerText =
      JSON.stringify(result, null, 2);

    updateChart(temperature, humidity);

    try {
      const tempScaled = Math.round(temperature * 100);
      const humidScaled = Math.round(humidity * 100);

      await contract.methods.logData(tempScaled, humidScaled).send({
        from: userAccount });

      console.log("Data berhasil disimpan ke blockchain");
      getLastReading();

    } catch (error) {
      console.error("Gagal menyimpan ke blockchain:", error);
    }
  }

  async function getLastReading() {

```

```

    try {

        const total = await
contract.methods.getTotalReadings().call();

        const latestIndex = total - 1;


        if (latestIndex >= 0) {

            const reading = await
contract.methods.getReading(latestIndex).call();

            const temp = reading.temperature / 100;

            const humid = reading.humidity / 100;


            document.getElementById('latestFromBlockchain').innerText =

                `🌐 Suhu: ${temp} °C | Kelembaban: ${humid}%`;


            updateChart(temp, humid);

        }

    } catch (err) {

        console.error("Gagal ambil data terakhir:", err);

    }

}

```

```

async function refreshAllData() {

    const statusEl = document.getElementById('status');

    try {

        statusEl.innerText = '⌚ Mengambil data dari blockchain...';

        const total = await
contract.methods.getTotalReadings().call();


        sensorChart.data.labels = [];

        sensorChart.data.datasets[0].data = [];

        sensorChart.data.datasets[1].data = [];

```



```

    for (let i = 0; i < total; i++) {

        const reading = await contract.methods.getReading(i).call();

        const temp = reading.temperature / 100;

        const humid = reading.humidity / 100;

        const timestamp = new Date(reading.timestamp *
1000).toLocaleTimeString();


        sensorChart.data.labels.push(timestamp);

        sensorChart.data.datasets[0].data.push(temp);

        sensorChart.data.datasets[1].data.push(humid);


    }

    sensorChart.update();

    statusEl.innerText = `  ${total} data berhasil dimuat dari
blockchain`;

    } catch (err) {

        console.error("Gagal refresh data:", err);

        statusEl.innerText = "  Gagal refresh dari blockchain!";

    }

}

```

2. sensor blockchain (index.html)

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width,
initial-scale=1.0"/>

    <title>Fermentasi KAKAO DApp</title>

    <link rel="stylesheet" href="style.css"/>

```

```

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/web3@1.10.0/dist/web3.min.js"></
script>

</head>

<body>

  <div class="wrapper">

    <div class="container">

      <h1>🔗 BIJI KAKAO 🔗</h1>

      <button onclick="connectMetamask()">Connect to
Metamask</button>

      <p id="status">🕒 Not Connected</p>

      <div class="form-section">

        <h2>☀️ Tambah Data Sensor</h2>

        <input type="text" id="sensorId" placeholder="Sensor ID"
/>

        <input type="text" id="temperature"
placeholder="Temperature (°C)" />

        <input type="text" id="humidity" placeholder="Humidity
(%)" />

        <button onclick="sendData()">📡 Kirim Data</button>

      </div>

      <div class="chart-section">

        <h2>📊 Grafik Pembacaan Sensor</h2>

        <div class="chart-container">

          <canvas id="sensorChart"></canvas>

        </div>

      </div>

```

```

        <div class="refresh-section">

            <h2><img alt="refresh icon" data-bbox="305 113 325 128" /> Ambil Semua Data dari Blockchain</h2>

            <button onclick="refreshAllData()"><img alt="refresh icon" data-bbox="633 141 653 156" /> Refresh
            Data</button>

        </div>

        <div class="latest-block">

            <h2><img alt="block icon" data-bbox="305 270 325 285" /> Blok Terbaru (Flask)</h2>

            <div class="block-box" id="latestBlock">Belum ada</div>

        </div>

        <div class="latest-block">

            <h2><img alt="block icon" data-bbox="305 410 325 425" /> Data Terakhir dari Blockchain</h2>

            <div class="block-box" id="latestFromBlockchain">Belum
            ada</div>

        </div>

    </div>

    <script src="app.js"></script>

</body>

</html>

```

3. sensor_blockchain ([app.py](#))

```

from flask import Flask, request, jsonify

from web3 import Web3

import json

app = Flask(__name__)

# Koneksi ke Ganache

```

```

w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))

# Atur akun default (biasanya akun[0] di Ganache)
w3.eth.default_account = w3.eth.accounts[0]

# Load ABI dari file
with open("build/SensorDataABI.json") as f:
    abi = json.load(f)

# Load alamat kontrak
with open("contract_address.txt") as f:
    contract_address = f.read().strip()

# Inisialisasi kontrak
contract = w3.eth.contract(address=contract_address, abi=abi)

# Endpoint untuk menambahkan data
@app.route("/add", methods=["POST"])
def add_data():
    data = request.get_json()

    try:
        # Kirim transaksi ke smart contract
        tx_hash = contract.functions.addData(
            data["sensor_id"],
            int(data["temperature"]),
            int(data["humidity"])
        ).transact()

```

```

        return jsonify({"status": "success", "tx_hash":
tx_hash.hex()})

    except Exception as e:

        return jsonify({"status": "error", "message": str(e)}),
400

# Endpoint untuk menghitung jumlah data

@app.route("/count", methods=["GET"])

def get_data_count():

    try:

        count = contract.functions.getDataCount().call()

        return jsonify({"status": "success", "count": count})

    except Exception as e:

        return jsonify({"status": "error", "message": str(e)}),
400

if __name__ == "__main__":

app.run(host="0.0.0.0", port=5000)

```

1. tcp_server (src [main.rs](#))

```

main.rs Modbus_client

use chrono::{Local, SecondsFormat};

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(

    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(())
    }

#[tokio::main]

```



```

async fn main() -> Result<(), Box<dyn Error>> {

    let sensor_id = "SHT20-PascaPanen-001";

    let location = "Gudang Fermentasi 1";

    let process_stage = "Fermentasi";

    loop {

        let timestamp = Local::now(); // Ubah dari Utc::now() ke
Local::now()

        match sht20(1).await {

            Ok(response) if response.len() == 2 => {

                let temp = response[0] as f32 / 10.0;

                let rh = response[1] as f32 / 10.0;

                println!("[{}] {} - {}: Temp={:.1}°C, RH={:.1}%",

                    timestamp.format("%Y-%m-%d %H:%M:%S"),

                    location,

                    process_stage,

                    temp,

                    rh);

                if let Err(e) = send_to_server(

                    sensor_id,

                    location,

                    process_stage,

                    temp,

                    rh,

```

```

        timestamp // Tetap menggunakan timestamp yang
sama

        ).await {

            eprintln!("Failed to send data: {}", e);

        }

    }

    Ok(invalid) => eprintln!("Invalid sensor response:
{:?}", invalid),

    Err(e) => eprintln!("Sensor read error: {}", e),

}

sleep(Duration::from_secs(10)).await;

}

}

```

2. tcp_serevr (Cargo toml)

```

cargo.toml Modbus_client

[package]

name = "sht20"

version = "0.1.0"

edition = "2021"


[dependencies]

chrono = "0.4"

serde_json = "1.0"

tokio = { version = "1.0", features = ["full"] }

tokio-modbus = "0.9"

tokio-serial = "5.4"

```

3. modbus (Cargo.toml)

```
[package]

name = "tcp_server"

version = "0.1.0"

edition = "2021"


[dependencies]

tokio = { version = "1.0", features = ["full"] }

serde = { version = "1.0", features = ["derive"] }

serde_json = "1.0"

influxdb2 = "0.4.0"

chrono = "0.4"

futures = "0.3"

dotenvy = "0.15"
```

4. modbus Cargo.toml

```
use serde::Deserialize;

use tokio::{

    io::{AsyncReadExt, AsyncWriteExt},

    net::TcpListener,

};

use futures::stream;

use chrono::{Utc, DateTime};

use dotenvy::dotenv;

use std::env;
```

```
#[derive(Debug, Deserialize)]

struct SensorData {

    timestamp: String,

    sensor_id: String,

    location: String,

    process_stage: String,

    temperature_celsius: f64,

    humidity_percent: f64,

}

#[tokio::main]

async fn main() -> Result<(), Box<dyn std::error::Error>> {

    dotenv().ok();

    let influx_url = "http://localhost:8086";

    let influx_org = "coffee";

    let influx_token = env::var("INFLUX_TOKEN")?;

    let influx_bucket = "SHT20";

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

    match client.health().await {

        Ok(health) => println!("InfluxDB healthy: {:?}", health),

        Err(e) => return Err(e.into()),

    }

    let listener = TcpListener::bind("127.0.0.1:7878").await?;
```

```

println!("TCP Server listening 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];

        loop {

            let n = match socket.read(&mut buf).await {

                Ok(n) if n == 0 => break,

                Ok(n) => n,

                Err(e) => {

                    eprintln!("Socket read error: {}", e);

                    break;

                }

            };

            let data = match std::str::from_utf8(&buf[..n]) {

                Ok(d) => d,

                Err(e) => {

                    eprintln!("Invalid UTF-8: {}", e);

                    let _ = socket.write_all(b"ERROR: UTF-8").await;

                    continue;

                }

            };

```

```

    };

    match serde_json::from_str::<SensorData>(data) {

        Ok(sensor_data) => {

            let timestamp = match
DateTime::parse_from_rfc3339(&sensor_data.timestamp) {

                Ok(dt) => dt.with_timezone(&Utc),

                Err(e) => {

                    let _ = socket.write_all(b"ERROR:
timestamp format").await;

                    eprintln!("Timestamp error: {}", e);

                    continue;

                }

            };

            use std::process::Command;

            ...

            match client.write(&bucket, stream::iter(vec![point])).await {

                Ok(_) => {

                    println!("Wrote to InfluxDB");

                    let output = Command::new("python3")

                        .arg("send_to_contract.py")

                        .arg(&sensor_data.sensor_id)

                        .arg(&sensor_data.location)

                        .arg(&sensor_data.process_stage)

                        .arg(sensor_data.temperature_celsius.to_string())

                        .arg(sensor_data.humidity_percent.to_string())

```

```

        .output();

        match output {
            Ok(res) => println!("Blockchain output: {}",
String::from_utf8_lossy(&res.stdout)),
            Err(e) => eprintln!("Blockchain call error: {}", e),
        }

        let _ = socket.write_all(b"OK").await;
    }

    ...
}

        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(_) => {

                let _ = socket.write_all(b"OK").await;

                println!("Wrote to InfluxDB");

            }

            Err(e) => {

                let _ = socket

                    .write_all(format!("ERROR: InfluxDB
write - {}", e).as_bytes())

                    .await;

            }

        }

    }

    Err(e) => {

        let _ = socket

            .write_all(format!("ERROR: JSON parsing -
{}", e).as_bytes())

            .await;

        eprintln!("JSON error: {}", e);

    }

}

});

}

}

```

5. QT (Monitor_gui.py)

```
import sys
```

```

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout,
QPushButton, QTextEdit

import pyqtgraph as pg

from influxdb_client import InfluxDBClient

from datetime import datetime


class InfluxMonitor(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()

        self.connectInfluxDB()

    def initUI(self):

        self.setWindowTitle("Monitoring InfluxDB dengan Grafik")

        self.resize(800, 600)

        layout = QVBoxLayout()

        # Widget output teks

        self.output = QTextEdit()

        self.output.setReadOnly(True)

        layout.addWidget(self.output)

        # Tombol refresh

        self.refresh_btn = QPushButton("Refresh Data")

        self.refresh_btn.clicked.connect(self.loadData)

        layout.addWidget(self.refresh_btn)

        # Widget grafik

        self.graphWidget = pg.PlotWidget()

```

```

self.graphWidget.setBackground('w')

layout.addWidget(self.graphWidget)

self.setLayout(layout)

def connectInfluxDB(self):

    self.token =
    "g3XcrGyuD-wA56C9w2JxJ6LYqXQaESzxIerq5H54e_-yLgo_dD9P-jxWNjw9_-YBK
    8m-i3oB38WXMwBODBqcdQ=="

    self.org = "kopi"

    self.bucket = "SHT20"

    self.url = "http://localhost:8086"

    self.client = InfluxDBClient(url=self.url,
    token=self.token, org=self.org)

    self.query_api = self.client.query_api()

def loadData(self):

    self.output.clear()

    self.graphWidget.clear()

    query = f'''

    from(bucket: "{self.bucket}")

    |> range(start: -1h)

    |> filter(fn: (r) => r._measurement ==
    "environment_monitoring")

    |> filter(fn: (r) => r._field == "temperature_celsius")

    |> filter(fn: (r) => r.location == "Gudang Fermentasi 1")

    |> filter(fn: (r) => r.process_stage == "Fermentasi")

    |> filter(fn: (r) => r.sensor_id ==
    "SHT20-PascaPanen-001")

    '''

```

```

try:

    print("Menjalankan query...")

    print(query)

    tables = self.query_api.query(query)

    times = []

    values = []

    for table in tables:

        for record in table.records:

            waktu = record.get_time().timestamp()

            nilai = record.get_value()

            times.append(waktu)

            values.append(nilai)

            # Output ke QTextEdit

            text_time =
datetime.fromtimestamp(waktu).strftime("%Y-%m-%d %H:%M:%S")

            line = f"{text_time} | {record.get_field()} |
{nilai}"

            self.output.append(line)

    if times and values:

        self.graphWidget.plot(times, values,
pen=pg.mkPen('r', width=2))

        self.graphWidget.setLabel('left', 'Temperature
(°C)')

        self.graphWidget.setLabel('bottom', 'Time
(Epoch)')

```

```
        self.graphWidget.setTitle("Grafik Pembacaan Sensor
Suhu")

    else:

        self.output.append("⚠ Tidak ada data untuk
ditampilkan.")

    except Exception as e:

        self.output.append(f"❌ Gagal mengambil data: {e}")

# Main Program

if __name__ == '__main__':

    app = QApplication(sys.argv)

    window = InfluxMonitor()

    window.show()

sys.exit(app.exec_())
```