

PROJECT BASED LEARNING

INTERKONEKSI SISTEM INSTRUMENTASI

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

”Sistem Monitoring Temperatur dan kelembapan pada Penyimpanan Bahan Mentah Pabrik Roti Rumahan”



Disusun Oleh Kelompok 11 :

Yudhistira Ananda Kuswantoro(2042231015)
Lusty Hanna Isyajidah (2042231045)
M. Daffi Aryatama (2042231075)

PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI
DEPARTEMEN TEKNIK INSTRUMENTASI
FAKULTAS VOKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
2024

DAFTAR ISI

DAFTAR ISI.....	1
BAB I PENDAHULUAN.....	2
1.1 Latar Belakang	2
1.2 Rumusan Masalah.....	2
1.3 Tujuan	3
BAB II TINJAUAN PUSTAKA	4
2.1 <i>State of The Art</i>	4
2.2 Sensor SHT20	5
2.3 Modbus Client (Rust).....	5
2.4 TCP Server (Rust).....	6
2.5 InfluxDB	7
2.6 Grafana.....	8
2.7 PyQt	9
2.8 Web3	9
2.9 Blockchain	10
2.10 Ganache.....	11
2.11 Metamask.....	11
BAB III METODOLOGI.....	13
3.2 Code main.rs Sensor SHT20.....	14
3.3 Code main.rs Server.....	15
3.4 Pengkomunikasian Seluruh Sistem.....	17
BAB IV HASIL DAN PEMBAHASAN	27
4.1 Hasil Tampilan Grafana.....	27
4.2 InfluxDB	27
4.3 Blockchain dan Web3	28
4.4 Hasil Interface pada Qt	30
BAB V KESIMPULAN.....	32
5.1 Kesimpulan	32
DAFTAR PUSTAKA	33
LAMPIRAN.....	35

BAB I PENDAHULUAN

1.1 Latar Belakang

Dalam proses produksi roti, kualitas bahan mentah seperti tepung, ragi, dan bahan tambahan lainnya sangat mempengaruhi hasil akhir produk. Penyimpanan bahan-bahan tersebut memerlukan perhatian khusus, terutama terhadap suhu dan kelembapan ruang penyimpanan. Ketidakstabilan dua parameter ini dapat mempercepat proses degradasi bahan dan menurunkan mutu adonan roti yang akan diproses. Oleh karena itu, diperlukan sistem monitoring yang mampu memantau kondisi lingkungan penyimpanan secara real-time untuk menjamin kualitas bahan tetap terjaga.

Pada industri roti skala rumah tangga, proses produksi sering dilakukan secara manual dengan keterbatasan tenaga kerja dan infrastruktur. Hal ini membuat sistem monitoring manual menjadi kurang efisien, apalagi jika lokasi penyimpanan terpisah dari area pengolahan. Khozainuz Zuhri, Ihkwan, dan Fahurian (2021) menjelaskan bahwa suhu dan kelembapan pada ruang penyimpanan bahan sangatlah penting untuk menjamin kualitas bahan produksi dan jika tidak dimonitor secara efisien dapat menghambat proses kerja.

Perkembangan teknologi Internet of Things (IoT) memberikan solusi efektif untuk memantau kondisi lingkungan penyimpanan secara jarak jauh dan otomatis. Penggunaan sensor suhu dan kelembapan seperti DHT20 serta mikrokontroler berbasis ESP8266 memungkinkan sistem untuk mengirimkan data secara real-time ke aplikasi berbasis web(Dedy, Axel, & Ivan, 2021). Dengan sistem ini, pekerja tidak perlu lagi memeriksa secara manual, sehingga efisiensi dan akurasi meningkat.

Selain itu, penerapan sistem monitoring suhu dan kelembapan berbasis IoT telah terbukti mendukung pengambilan keputusan yang tepat dalam hal penyimpanan bahan makanan. Studi oleh Wijaya dan Nurjaman (2020) menunjukkan bahwa pemanfaatan IoT dalam sistem pendukung keputusan berbasis metode Weighted Product dapat memberikan rekomendasi penyimpanan bahan roti dengan akurasi tinggi, serta membantu unit usaha roti dalam mempertahankan kualitas produk dan efisiensi operasional.

Dengan mempertimbangkan pentingnya kualitas bahan mentah dalam produksi roti rumahan serta potensi kerugian akibat kerusakan bahan karena suhu dan kelembapan yang tidak sesuai, maka pengembangan sistem monitoring temperatur dan kelembapan berbasis IoT menjadi langkah strategis. Sistem ini tidak hanya mendukung efisiensi kerja tetapi juga menjaga konsistensi mutu produk yang dihasilkan.

1.2 Rumusan Masalah

Berdasarkan paparan latar belakang, maka rumusan masalah yang diambil pada tugas ini adalah:

1. Bagaimana cara melakukan monitoring suhu dan kelembaban secara efektif pada ruang penyimpanan bahan mentah untuk produksi roti rumahan?
2. Apa saja dampak dari tidak terkendalinya suhu dan kelembaban terhadap kualitas bahan mentah yang digunakan dalam pembuatan roti?
3. Bagaimana sistem monitoring dapat membantu menjaga mutu bahan mentah selama masa penyimpanan?

1.3 Tujuan

Berdasarkan paparan rumusan masalah yang didapat maka tujuan dari tugas ini adalah:

1. Mengetahui dan menganalisis pengaruh suhu dan kelembaban terhadap kualitas bahan mentah dalam proses produksi roti.
2. Merancang sistem monitoring sederhana yang mampu mencatat dan memberikan informasi mengenai kondisi lingkungan ruang penyimpanan bahan mentah.
3. Memberikan solusi yang dapat membantu pelaku usaha roti rumahan dalam menjaga kestabilan kondisi penyimpanan untuk mempertahankan mutu bahan baku.

BAB II TINJAUAN PUSTAKA

2.1 State of The Art

Tabel 2.1 State of The Art

Topik, Penulis dan Tahun	Metode	Hasil
Sistem monitoring gudang penyimpanan roti berbasis IoT (Wijaya & Nurjaman, 2020)	Weighted Product (WP) + Node-RED	Sistem memberikan rekomendasi gudang penyimpanan roti terbaik dengan akurasi tinggi menggunakan parameter lingkungan.
Monitoring suhu dan kelembapan ruang penyimpanan roti (Zuhri et al., 2021)	Prototyping; Sensor DHT22; Web & App	Prototipe sistem monitoring real-time berbasis sensor dan aplikasi berhasil dibangun dan mempermudah pengawasan suhu dan kelembapan ruang penyimpanan roti.
Pemantauan suhu & kelembapan menggunakan MQTT dan Wemos D1 Mini (Dedy et al., 2021)	MQTT, Node-RED, DHT11	Sistem dapat memantau suhu dan kelembapan secara efisien dengan notifikasi saat melebihi ambang batas menggunakan MQTT dan visualisasi Node-RED.
Traceability Sistem Roti Tradisional berbasis Blockchain & IoT (Cocco et al., 2021)	Blockchain, IPFS, Sensor Network	Sistem memungkinkan pelacakan kondisi suhu & kelembapan bahan roti secara akurat di sepanjang rantai pasok untuk menjamin kualitas dan keamanan pangan.

2.2 Sensor SHT20



Gambar 2.1 Sensor SHT20

Sensor SHT20 merupakan sensor digital yang digunakan untuk mengukur suhu dan kelembaban dengan tingkat akurasi yang cukup baik. Sensor ini banyak diaplikasikan dalam berbagai sistem monitoring lingkungan karena bentuknya yang kompak dan konsumsi daya yang rendah. Dalam penelitian Khumaidi et al. (2024), sensor SHT20 digunakan untuk memantau suhu dan kelembaban pada panel listrik guna menjaga kestabilan kinerja sistem distribusi daya.

Pengujian yang dilakukan menunjukkan bahwa sensor SHT20 memiliki rata-rata tingkat kesalahan (error) sebesar 2,4% untuk suhu dan 1,0% untuk kelembaban jika dibandingkan dengan alat ukur HTC-2 sebagai acuan. Hasil ini menunjukkan bahwa SHT20 memiliki tingkat presisi yang cukup baik untuk aplikasi monitoring suhu dan kelembaban secara real-time, terutama dalam sistem kendali otomatis yang memerlukan respon cepat terhadap perubahan lingkungan (Khumaidi et al., 2024).

Keunggulan lain dari sensor SHT20 adalah kemampuannya untuk berintegrasi dengan mikrokontroler seperti ESP32, yang memungkinkan data hasil pengukuran dikirim dan ditampilkan secara langsung melalui antarmuka seperti HMI atau aplikasi Android. Dalam sistem yang diuji oleh Khumaidi et al., data dari sensor ini digunakan untuk mengaktifkan kipas pendingin jika suhu melebihi 33°C, atau mengaktifkan pemanas jika kelembaban melebihi 65%, sebagai bagian dari sistem proteksi otomatis terhadap panel listrik.

Dengan akurasi yang cukup tinggi dan kemudahan integrasi, SHT20 menjadi salah satu pilihan sensor yang ideal dalam sistem pemantauan suhu dan kelembaban untuk berbagai aplikasi, termasuk penyimpanan bahan makanan, ruang produksi, dan lingkungan industri.

2.3 Modbus Client (Rust)

Modbus merupakan salah satu protokol komunikasi industri yang banyak digunakan untuk pertukaran data antara perangkat seperti PLC, sensor, dan sistem monitoring. Dalam pengembangannya, protokol ini membutuhkan perangkat lunak klien yang andal, ringan, dan aman, terutama ketika diimplementasikan dalam sistem tertanam (*embedded systems*). Bahasa

pemrograman Rust hadir sebagai alternatif modern yang menjawab tantangan ini, karena menawarkan keamanan memori tanpa memerlukan garbage collector, serta efisiensi performa setara C/C++.

Dalam penelitian Pérez dan Mejía (2021), dikembangkan sebuah Modbus Client berbasis Rust yang didesain dengan prinsip keselamatan memori dan arsitektur modular. Salah satu fitur utama dari implementasi ini adalah penggunaan RingBuffer untuk menyimpan dan mengelola data paket Modbus RTU tanpa alokasi memori dinamis (heap), sehingga sangat cocok untuk sistem tertanam dengan sumber daya terbatas. Klien ini dirancang menggunakan prinsip ownership dan lifetime Rust, sehingga dapat menghindari kesalahan umum seperti dangling pointer atau kebocoran memori.

Modbus Client ini bersifat generik terhadap perangkat keras, karena mengandalkan traits Read dan Write dari pustaka embedded-hal dalam ekosistem Rust. Ini memungkinkan implementasi kode yang sama digunakan di berbagai mikrokontroler, selama implementasi HAL (Hardware Abstraction Layer) sesuai dengan standar tersebut. Dalam demo mereka, klien digunakan pada dua platform yaitu FRDM-K64F dan STM32F4Discovery, dengan komunikasi UART antar klien dan server Modbus berbasis C#.

Arsitektur perpustakaan Modbus ini menunjukkan keunggulan Rust dalam membangun sistem tertanam yang aman, modular, dan dapat dipelihara, sekaligus menghindari kompleksitas dan kerentanan yang umum dijumpai saat menggunakan C/C++ dalam pengembangan firmware.

2.4 TCP Server (Rust)



TCP (Transmission Control Protocol) adalah protokol utama pada lapisan transport dalam model TCP/IP yang digunakan untuk menyediakan koneksi andal antar dua titik komunikasi. Dalam konteks pengembangan sistem jaringan modern, pengujian dan verifikasi implementasi TCP menjadi sangat penting agar sistem tidak hanya berjalan dengan benar tetapi juga aman terhadap berbagai kemungkinan kesalahan komunikasi.

Cavoj *et al.* (2024) memperkenalkan pendekatan baru dalam membangun TCP server menggunakan bahasa pemrograman Rust dengan menerapkan konsep Multiparty Session Types (MPST). MPST adalah metode pemodelan komunikasi formal yang membantu menjamin urutan dan jenis pesan antar entitas komunikasi sesuai protokol yang telah ditentukan. Rust dipilih karena memiliki sistem tipe dan mekanisme kepemilikan (ownership) yang ketat, sehingga sangat cocok untuk pemrograman sistem tingkat rendah dengan keamanan memori yang tinggi.

Implementasi TCP Server dalam penelitian ini mencakup proses tiga tahap utama dalam protokol TCP: establishing a connection, data transmission, dan closing the connection, seluruhnya dimodelkan dalam bentuk session types di dalam Rust. Salah satu kontribusi utama adalah penyusunan struktur Rust seperti OfferOne, OfferTwo, dan SelectOne, yang mewakili berbagai bentuk cabang dalam komunikasi protokol. Struktur ini memudahkan pengecekan oleh compiler Rust untuk memastikan implementasi mengikuti alur komunikasi yang benar.

Prototipe server yang dibangun telah diuji terhadap TCP stack kernel Linux dan berhasil menangani skenario nyata seperti retransmisi akibat kehilangan paket, urutan paket yang tidak sesuai, serta penutupan koneksi abnormal. Hasil evaluasi menunjukkan bahwa kombinasi Rust dan MPST dapat meningkatkan keandalan serta keamanan dalam pengembangan sistem komunikasi berbasis TCP, sekaligus membuka jalan untuk penerapan session types dalam protokol transport layer secara lebih luas.

2.5 InfluxDB



Gambar 2.2 InfluxDB

InfluxDB adalah salah satu *Time Series Database* (TSDB) yang dirancang khusus untuk menangani data berbasis waktu atau timestamped data. InfluxDB dikembangkan oleh InfluxData dan ditulis menggunakan bahasa Go. Basis data ini dioptimalkan untuk melakukan pencatatan dan analisis data secara real-time, seperti data suhu, kelembaban, trafik jaringan, log sistem, serta data IoT lainnya yang terus-menerus berubah dalam interval waktu tertentu (Naqvi & Yfantidou, 2017).

InfluxDB bersifat schema-less dan menyediakan antarmuka query yang mirip dengan SQL, yaitu InfluxQL, yang memudahkan pengguna dalam menulis perintah meskipun berasal dari latar belakang relasional database. Struktur data InfluxDB terdiri atas measurement (setara dengan tabel), tags (yang diindeks untuk mempercepat pencarian), fields (nilai-nilai aktual yang disimpan), dan timestamps (penanda waktu unik untuk setiap data).

InfluxDB memiliki keunggulan signifikan dalam hal performa dan efisiensi penyimpanan. Berdasarkan pengujian yang dilakukan, InfluxDB mampu memberikan kompresi disk 27 kali lebih efisien dibandingkan SQL Server dan memiliki rata-rata waktu eksekusi query 8 kali lebih cepat. Hal ini menjadikannya sangat cocok untuk penggunaan dalam sistem monitoring seperti DevOps, *Internet of Things* (IoT), dan analisis data waktu nyata (Naqvi & Yfantidou, 2017).

Dalam hal arsitektur, InfluxDB menggunakan mesin penyimpanan TSM Tree (Time Structured Merge Tree) yang menawarkan kompresi tinggi dan penghapusan data yang lebih efisien dibandingkan LSM Tree yang digunakan di versi sebelumnya. InfluxDB juga mendukung fitur continuous queries dan retention policies untuk menyederhanakan agregasi data dan pengelolaan data historis.

Secara keseluruhan, InfluxDB merupakan solusi database yang efisien, *scalable*, dan mudah diintegrasikan dalam sistem pemantauan waktu nyata, termasuk dalam proyek sistem monitoring temperatur dan kelembaban pada ruang penyimpanan.

2.6 Grafana



Grafana merupakan aplikasi visualisasi dan analisis data berbasis web yang dirancang untuk menampilkan data waktu nyata (real-time) secara interaktif dan informatif. Aplikasi ini pertama kali dikembangkan oleh Torkel Ödegaard pada tahun 2014 dan berbasis open-source. Awalnya Grafana dirancang untuk bekerja dengan basis data deret waktu seperti InfluxDB dan Prometheus, namun seiring waktu, dukungannya meluas hingga mencakup basis data relasional seperti PostgreSQL dan Microsoft SQL Server (Mattsson, 2024).

Grafana memungkinkan pengguna untuk membuat dashboard atau panel visualisasi yang dapat menampilkan data dalam bentuk grafik, diagram batang, meteran (gauge), hingga notifikasi alarm. Panel-panel ini dapat disesuaikan untuk menampilkan data dengan cara yang paling sesuai dengan kebutuhan pengguna. Fitur threshold dan alerting memungkinkan pengguna untuk diberi tahu secara otomatis jika parameter yang dipantau melebihi ambang batas yang ditentukan.

Dalam studi yang dilakukan oleh Mattsson (2024), Grafana digunakan untuk memvisualisasikan data real-time dari proses ekstrusi pada perusahaan manufaktur. Salah satu keuntungan utamanya adalah kemampuan untuk diakses melalui browser tanpa memerlukan lisensi khusus atau proses login, sehingga sangat cocok ditampilkan di layar umum seperti monitor ruang produksi atau TV. Pengguna cukup mengakses dashboard melalui tautan web yang telah dikonfigurasi sebelumnya.

Namun, studi tersebut juga mencatat keterbatasan Grafana, seperti tidak mendukung pengambilan data dari database Sybase dan kesulitan dalam menggabungkan data dari lebih dari dua basis data yang memiliki kunci berbeda. Meskipun demikian, Grafana tetap mampu menyajikan metrik penting seperti utilisasi mesin, parameter proses, dan status produksi harian hingga tahunan secara informatif dan responsif.

Dengan kemampuan menyajikan data secara real-time, dukungan terhadap banyak sumber data, serta tampilan antarmuka yang interaktif, Grafana menjadi alat visualisasi yang sangat efektif untuk diterapkan dalam sistem monitoring industri, termasuk untuk pemantauan suhu dan kelembaban di fasilitas produksi maupun penyimpanan bahan mentah.

2.7 PyQt



Gambar 2.4 PyQt

PyQt adalah pustaka Python yang merupakan binding dari framework Qt, yang dirancang untuk membangun antarmuka grafis pengguna (GUI) yang kaya fitur dan interaktif. PyQt5 secara khusus memungkinkan para pengembang untuk menggabungkan kekuatan pemrograman berorientasi objek dalam Python dengan pustaka antarmuka Qt yang luas dan stabil, sehingga cocok digunakan dalam pembuatan aplikasi desktop yang kompleks dan profesional (Soliev & G'iyosiddinov, 2024).

Salah satu keunggulan utama PyQt adalah integrasinya dengan Qt Designer, sebuah alat visual yang memungkinkan perancangan antarmuka secara drag-and-drop. Qt Designer mempercepat proses perancangan UI dengan mengurangi kebutuhan penulisan kode secara manual. Integrasi ini memungkinkan UI yang dirancang secara visual dapat diubah menjadi kode Python menggunakan perintah pyuic5, sehingga mempercepat iterasi dan prototyping (Soliev & G'iyosiddinov, 2024).

Dalam pengembangan aplikasi, PyQt mendukung pendekatan modular, yang mendorong pembuatan komponen GUI dalam bentuk widget khusus atau kelas terpisah. Hal ini tidak hanya meningkatkan keterbacaan dan pemeliharaan kode, tetapi juga memungkinkan penggunaan kembali (reusability) komponen UI dalam berbagai proyek. Selain itu, PyQt menyediakan mekanisme signal-slot untuk menangani interaksi pengguna secara efisien, seperti menghubungkan klik tombol dengan aksi tertentu di backend aplikasi.

Penelitian oleh Soliev dan G'iyosiddinov (2024) menunjukkan bahwa kombinasi PyQt5 dan Qt Designer menghasilkan pengembangan GUI yang lebih cepat, terstruktur, dan ramah pengguna. Aplikasi yang dikembangkan dapat disesuaikan dengan kebutuhan pengguna akhir, serta mendukung internasionalisasi, aksesibilitas, dan pengujian yang lebih mudah. Fitur seperti shortcut keyboard, menu interaktif, dan layout dinamis juga memperkaya pengalaman pengguna secara keseluruhan.

2.8 Web3

Web3 adalah evolusi generasi ketiga dari arsitektur web yang mengedepankan prinsip desentralisasi, transparansi, dan kepemilikan data oleh pengguna. Konsep ini merupakan kelanjutan dari Web1 yang bersifat hanya baca (read-only), dan Web2 yang memungkinkan interaksi dua arah (read-write). Dalam Web3, pengguna tidak hanya dapat membaca dan menulis informasi, tetapi juga memiliki kendali penuh atas data yang mereka hasilkan melalui sistem terdesentralisasi berbasis blockchain (Sheridan et al., 2022).

Web3 bertumpu pada beberapa pilar teknologi utama, yaitu blockchain publik (seperti Ethereum dan Bitcoin), smart contracts, digital wallets (identitas pengguna), dan pustaka

pemrograman Web3 seperti Web3.js, Ethers.js, dan Truffle Suite. Semua elemen ini bekerja sama untuk membangun ekosistem aplikasi terdesentralisasi (DApps) yang tidak lagi bergantung pada otoritas pusat seperti server atau perusahaan penyedia layanan. Sistem ini memungkinkan pengguna untuk berinteraksi secara langsung dan aman dengan data serta layanan berbasis logika terbuka yang tertanam dalam blockchain (Sheridan et al., 2022).

Keunggulan Web3 terletak pada kemampuannya untuk memungkinkan pemrosesan data yang lebih cerdas dan aman melalui smart contracts—kode otomatis yang mewakili perjanjian digital dan dijalankan secara transparan oleh node dalam jaringan blockchain. Dengan pendekatan ini, Web3 memungkinkan pertukaran informasi dan nilai secara peer-to-peer tanpa perantara, meningkatkan kecepatan transaksi, keandalan data, serta memberikan pengguna kendali penuh atas identitas dan informasi mereka.

Namun, perkembangan Web3 juga menghadirkan tantangan, terutama dalam hal regulasi, privasi data, dan akurasi informasi. Tanpa otoritas pusat yang mengatur konten dan interaksi, Web3 membuka potensi penyalahgunaan data dan distribusi informasi yang tidak dapat diverifikasi. Oleh karena itu, meskipun Web3 menjanjikan perubahan mendasar dalam ekosistem internet, perlu kesiapan infrastruktur, kerangka hukum, dan kesadaran pengguna dalam menghadapinya (Sheridan et al., 2022).

2.9 Blockchain

Blockchain merupakan teknologi yang dirancang untuk menciptakan kepercayaan dalam sistem terdistribusi tanpa memerlukan otoritas pusat. Teknologi ini pertama kali diperkenalkan oleh Satoshi Nakamoto dalam makalah berjudul "Bitcoin: A Peer-to-Peer Electronic Cash System", yang menjadi fondasi dari mata uang kripto Bitcoin. Blockchain kini telah berkembang melampaui penggunaannya di dunia cryptocurrency dan digunakan dalam berbagai bidang seperti sistem keuangan, pencatatan hak kepemilikan, dan smart contract (Di Pierro, 2017).

Secara sederhana, blockchain adalah struktur data berbentuk rantai blok yang masing-masing berisi catatan transaksi, cap waktu (timestamp), serta hash kriptografi dari blok sebelumnya. Dengan demikian, setiap blok terkait satu sama lain secara berurutan dan membentuk satu rantai yang aman dari manipulasi data. Jika ada perubahan pada satu blok, maka hash seluruh blok berikutnya juga akan berubah, sehingga modifikasi dapat terdeteksi secara instan (Di Pierro, 2017).

Keunggulan utama dari blockchain adalah mekanisme pencatatan yang bersifat terdistribusi. Alih-alih menyimpan data di satu pusat (centralized ledger), blockchain menyebarkan salinan catatan transaksi ke banyak node atau komputer dalam jaringan. Hal ini menjadikan sistem lebih tahan terhadap kerusakan data, pemalsuan, maupun kegagalan sistem, karena salinan yang tersebar dapat saling memverifikasi keabsahan transaksi.

Untuk menjamin integritas data, blockchain menggunakan fungsi hash kriptografi seperti SHA1. Fungsi ini menghasilkan output unik dari setiap data transaksi, dan sangat sulit untuk ditemukan dua input berbeda yang menghasilkan hash yang sama (collision). Dalam penerapannya, blockchain digunakan tidak hanya untuk mencatat transfer mata uang kripto, tetapi juga untuk menyimpan informasi kepemilikan aset, sertifikat, atau bahkan janji berbasis kode yang akan dieksekusi secara otomatis — dikenal sebagai smart contract (Di Pierro, 2017).

Dengan kemampuannya menyimpan data secara transparan, aman, dan tidak dapat dimanipulasi, blockchain menawarkan solusi modern dalam pencatatan data yang dapat diaudit secara independen dan digunakan dalam berbagai skenario di luar sektor keuangan.

2.10 Ganache

Ganache merupakan salah satu komponen penting dalam pengembangan dan pengujian aplikasi berbasis blockchain, khususnya pada ekosistem Ethereum. Ganache adalah jaringan blockchain lokal yang disediakan oleh Truffle Suite yang memungkinkan pengembang menjalankan blockchain pribadi secara lokal di komputer mereka. Keunggulan utama Ganache terletak pada kemampuannya untuk mensimulasikan jaringan Ethereum secara lengkap, termasuk transaksi, smart contract, serta akun-akun pengguna dengan saldo Ether uji coba (Ahmad et al., 2022).

Dalam konteks penelitian yang dilakukan oleh Ahmad et al. (2022), Ganache digunakan sebagai lingkungan uji untuk pengembangan sistem e-voting berbasis blockchain. Jaringan lokal ini memungkinkan pencatatan transaksi pemilihan suara secara terdistribusi tanpa memerlukan server pusat. Setiap transaksi yang terjadi, seperti pemilihan kandidat, dicatat dalam blok baru dan dapat diverifikasi oleh pengguna melalui antarmuka blockchain lokal. Hal ini memastikan transparansi dan imutabilitas data pemilu, meskipun dijalankan dalam skala lokal untuk keperluan simulasi.

Penggunaan Ganache memberikan keuntungan dalam hal kecepatan dan efisiensi pengujian karena tidak perlu terkoneksi dengan jaringan Ethereum publik yang memerlukan gas fee riil dan waktu konfirmasi transaksi yang lebih lama. Ganache juga memungkinkan penciptaan akun-akun Ethereum fiktif secara instan, masing-masing dengan private key dan saldo Ether simulasi, sehingga sangat ideal untuk pengembangan awal dan demonstrasi sistem (Ahmad et al., 2022).

2.11 Metamask



Gambar 2.5 Metamask

MetaMask merupakan ekstensi dompet digital berbasis web yang memungkinkan pengguna untuk terhubung langsung ke jaringan blockchain melalui browser seperti Chrome, Firefox, Opera, dan Brave. Ekstensi ini mengubah browser biasa menjadi "blockchain browser", memungkinkan pengguna untuk menjalankan aplikasi terdesentralisasi (DApp) di jaringan Ethereum tanpa harus menjalankan seluruh node Ethereum secara lokal (Khan et al., 2024).

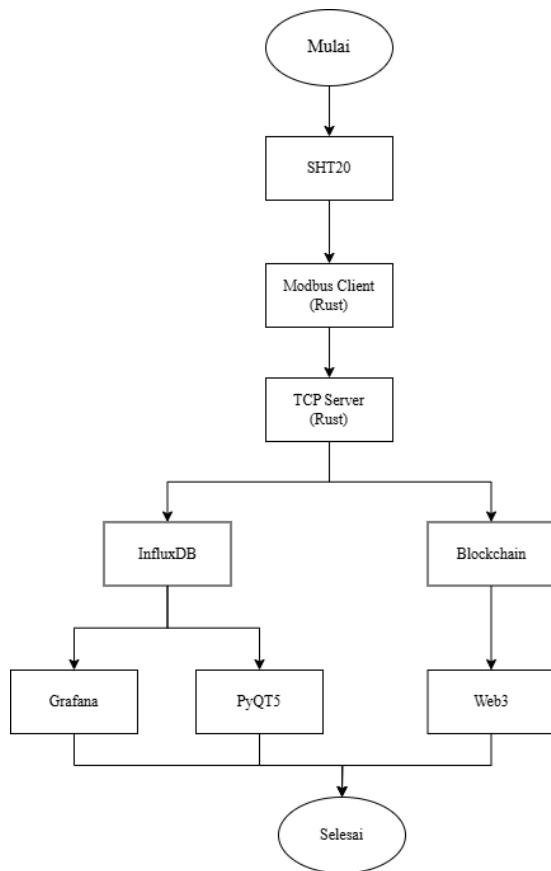
MetaMask bekerja dengan membuat akun Ethereum yang terdiri atas alamat publik dan private key, yang digunakan untuk menandatangi transaksi digital secara aman. Selain itu, MetaMask juga menyimpan dan mengelola riwayat transaksi pengguna serta memungkinkan impor dan ekspor private key. Untuk mendukung pengalaman pengguna, MetaMask menyediakan antarmuka yang ramah pengguna, sekaligus menyediakan fungsi-fungsi penting seperti verifikasi tanda tangan digital, pengelolaan nonce, serta otentikasi berbasis tanda tangan (signature-based authentication) terhadap DApps yang terhubung dengannya (Khan et al., 2024).

Dalam konteks autentikasi terdesentralisasi, MetaMask memainkan peran penting sebagai perantara antara pengguna dan aplikasi blockchain. Ketika pengguna mengakses sebuah DApp, aplikasi akan memunculkan permintaan koneksi ke dompet MetaMask. Setelah disetujui, DApp akan mengirimkan pesan otentikasi ke wallet pengguna untuk ditandatangani. Proses ini memungkinkan validasi identitas pengguna tanpa memerlukan server pusat atau penyimpanan

kredensial konvensional, sehingga meningkatkan keamanan dan privasi data (Khan et al., 2024).

BAB III METODOLOGI

3.1 Alur Proses



Gambar 3.1

Diagram alur proses pada **Gambar 3.1** tersebut menggambarkan alur sistem pemantauan suhu dan kelembapan menggunakan sensor SHT20 yang mengirimkan data melalui Modbus Client berbasis Rust ke TCP Server yang juga dikembangkan dengan Rust. Data yang diterima kemudian dicatat ke dalam InfluxDB sebagai basis penyimpanan time-series, memungkinkan pemantauan historis berbasis waktu. Visualisasi data dilakukan melalui dashboard Grafana dan antarmuka GUI menggunakan PyQt5 untuk kemudahan akses pengguna. Secara paralel, data juga dicatat ke dalam sistem blockchain untuk menjamin transparansi, keamanan, dan integritas data, yang dapat diakses melalui platform Web3. Sistem ini menyediakan solusi monitoring lingkungan yang terdesentralisasi, real-time, dan terpercaya bagi kebutuhan industri rumah tangga.

3.2 Code main.rs Sensor SHT20



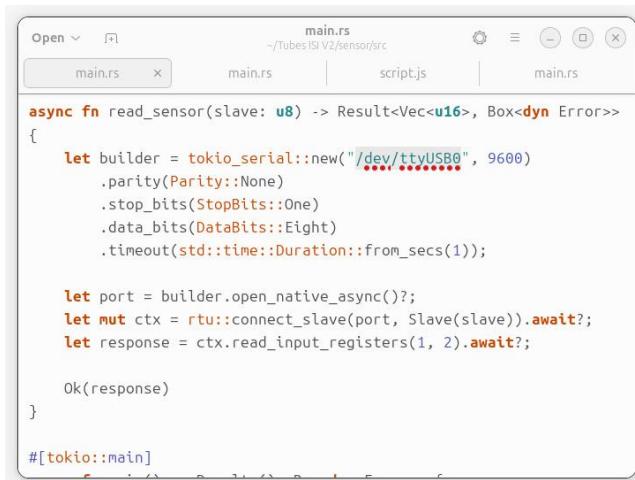
```
use tokio_modbus::{client::rtu, prelude::*};
use tokio_serial::{SerialPortBuilderExt, Parity, StopBits,
DataBits};
use tokio::net::TcpStream;
use tokio::io::AsyncWriteExt;
use serde::Serialize;
use chrono::Utc;
use std::error::Error;
use tokio::time::{sleep, Duration};

#[derive(Serialize)]
struct SensorData {
    timestamp: String,
    sensor_id: String,
    location: String,
    process_stage: String,
```

Gambar 3.2 Code main.rs untuk Sensor

Pada **gambar 3.2** tercantum library tokio modbus dan tokio serial. Library tokio_modbus merupakan pustaka Rust language yang menggunakan komunikasi Modbus RTU secara asinkron. Pada cuplikan kode tersebut, bagian `use tokio_modbus::{client::rtu, prelude::*};` mengindikasikan bahwa sistem akan bertindak sebagai klien Modbus RTU, yang artinya dia akan menginisiasi permintaan pembacaan data dari sensor.

Sementara itu, tokio_serial merupakan pustaka tambahan yang berguna untuk membuka dan mengelola port serial (UART) dalam konteks asinkron menggunakan runtime tokio. Dengan menggunakan `tokio_serial::{SerialPortBuilderExt, Parity, StopBits, DataBits};`, sistem dapat menentukan konfigurasi port serial secara detail, seperti jumlah bit data, bit stop, dan parity check, yang semuanya penting dalam menjaga keandalan transmisi data sensor. Integrasi antara tokio_modbus dan tokio_serial membuat sistem ini mampu melakukan pembacaan sensor suhu dan kelembapan secara real-time, cocok untuk sistem monitoring suhu dan kelembapan penyimpanan roti berskala rumahan.



```
async fn read_sensor(slave: u8) -> Result<Vec<u16>, Box<dyn Error>>
{
    let builder = tokio_serial::new("/dev/ttyUSB0", 9600)
        .parity(Parity::None)
        .stop_bits(StopBits::One)
        .data_bits(DataBits::Eight)
        .timeout(std::time::Duration::from_secs(1));

    let port = builder.open_native_async()?;
    let mut ctx = rtu::connect_slave(port, Slave(slave)).await?;
    let response = ctx.read_input_registers(1, 2).await?;

    Ok(response)
}

#[tokio::main]
```

Gambar 3.3 Code main.rs Port USB dari Sensor

Pada **Gambar 3.3** merupakan code main.rs yang mendefinisikan port USB sensor dilakukan melalui `tokio_serial::new("/dev/ttyUSB0", 9600)`, yang mengatur komunikasi serial pada baud rate 9600. Konfigurasi ditetapkan dengan parity none, 1 stop bit, dan 8 data bit. Port ini digunakan untuk membaca data dari sensor secara asynchronous.

```

main.rs
~/Tubes ISI V2/sensor/src
main.rs | script.js | main.rs

match TcpStream::connect("192.168.1.66:9000").await
{
    Ok(mut stream) => {
        stream.write_all(json.as_bytes()).await?;
        stream.write_all(b"\n").await?;
        println!("✅ Data dikirim ke TCP server");
    },
    Err(e) => {
        println!("❌ Gagal koneksi ke TCP server: {}", e);
    }
},
Ok(other) => {
    println!("⚠️ Data tidak lengkap: {:?}", other);
},
Err(e) => {
    println!("❌ Gagal baca sensor: {}", e);
}
}

```

Gambar 3.4 Code main.rs Data dari Sensor ke TCP Server

Pada **gambar 3.4** merupakan Code main.rs yang menghubungkan perangkat ke TCP server di alamat IP 192.168.1.66:9000, lalu mengirimkan data sensor dalam format JSON. Jika berhasil, data dikirim dan ditampilkan pesan sukses. Jika gagal, sistem akan mencetak pesan kesalahan koneksi atau pembacaan sensor.

3.3 Code main.rs Server

a. Penginputan Influx Token

```

main.rs
~/Tubes ISI V2/Server/src
main.rs | script.js

async fn main() -> anyhow::Result<()> {
    // --- InfluxDB setup ---
    let influx_url = "http://localhost:8086/api/v2/write?org=ITS&bucket=ISImonitor&precision=s";
    let influx_token =
        "mTj04KT4rlVlwTISNiewi_JWxJFrd67bxRbx_pSrHmVz8-
        eTjqcT_IxbEoTSJRUd49uBsZ9uaPC7aYMRwZ0==";
    let http_client = Client::new();

    // --- Ethereum setup ---
    let provider = Provider::<Http>::try_from("http://
    192.168.1.66:8545")?;
    let wallet: LocalWallet =
        "0xa0bba7892d7df21091d9321298bdafce973e6ac427a9f6350ea345bf9d777f0f"
        .parse::<LocalWallet>()?
        .with_chain_id(1337u64);
    let client = Arc::new(SignerMiddleware::new(provider, wallet));

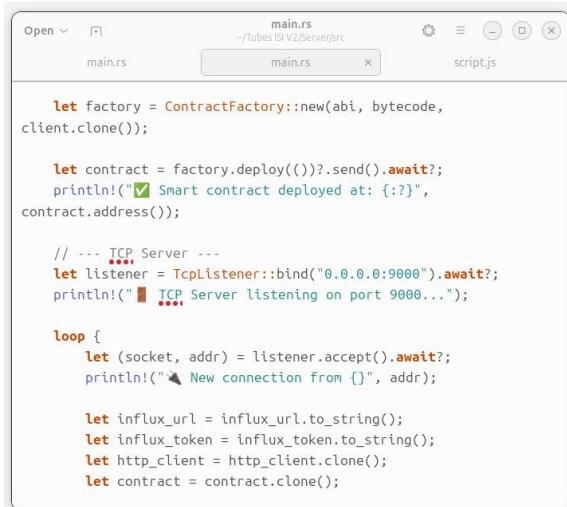
    // Baca dan parse ABI dan bytecode dengan benar
    let abi = fs::read_to_string("build/ContractABI.json")?;
}

```

Gambar 3.5 Code main.rs TCP Server

Pada **gambar 3.5** menampilkan code yang mengatur koneksi ke InfluxDB melalui influx_url dan influx_token, lalu membuat klien HTTP untuk mengirim data sensor dari TCP Server ke database time-series InfluxDB. Data dikirim melalui endpoint API untuk pencatatan *real-time*, dengan format dan autentikasi yang telah dikonfigurasi.

b. Smart Contract



```
let factory = ContractFactory::new(abi, bytecode,
client.clone());

let contract = factory.deploy()?.send().await?;
println!("✅ Smart contract deployed at: {:?}", contract.address());

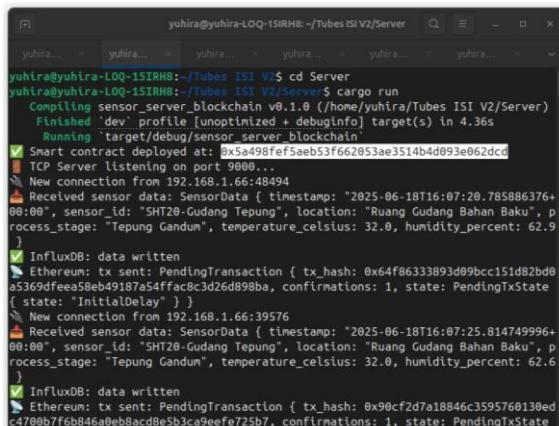
// --- TCP Server ---
let listener = TcpListener::bind("0.0.0.0:9000").await?;
println!("🔴 TCP Server listening on port 9000...");

loop {
    let (socket, addr) = listener.accept().await?;
    println!("⚡ New connection from {}", addr);

    let influx_url = influx_url.to_string();
    let influx_token = influx_token.to_string();
    let http_client = http_client.clone();
    let contract = contract.clone();
}
```

Gambar 3.6 Code main.rs Smart Contract

Pada **gambar 3.6** menunjukkan potongan kode program dalam bahasa Rust yang digunakan untuk melakukan deployment smart contract sekaligus membangun server TCP. Pembuatan smart contract dimulai dengan mendefinisikan ContractFactory yang berisi ABI (Application Binary Interface) dan bytecode hasil kompilasi smart contract Solidity. Kemudian, smart contract di deploy ke jaringan blockchain menggunakan perintah factory.deploy().send().await, dan alamat kontrak yang berhasil di-deploy ditampilkan sebagai output. Setelah proses deploy, sistem melanjutkan dengan membuka koneksi TCP pada port 9000 menggunakan TcpListener, yang memungkinkan sistem menerima data dari klien eksternal. Data yang masuk dapat digunakan untuk mengakses ulang kontrak pintar yang telah di deploy dan menjalankan fungsionalitas sesuai kebutuhan. Dengan pendekatan ini, integrasi antara blockchain dan server backend dapat berjalan secara sinkron dan otomatis.

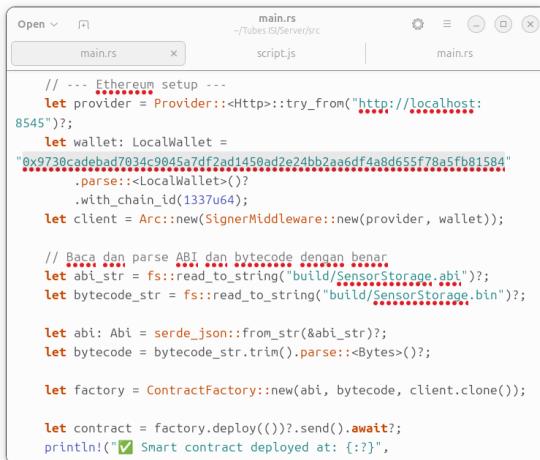


```
yuhira@yuhira-LOQ-15IRHB: ~/Tubes ISI V2/Server$ cargo run
Compiling sensor_server.blockchain v0.1.0 (/home/yuhira/Tubes ISI V2/Server)
Finished 'dev' profile [unoptimized + debuginfo] targets(s) in 4.36s
Running `target/debug/sensor_server.blockchain`
✅ Smart contract deployed at: 0x5a498fef5aeb53f662053ae3514b4d093e062dcd
🔴 TCP Server listening on port 9000...
⚡ New connection from 192.168.1.66:48494
⚠ Received sensor data: SensorData { timestamp: "2025-06-18T16:07:20.785886376+00:00", sensor_id: "SHT20-Gudang Tepung", location: "Ruang Gudang Bahan Baku", process_stage: "Tepung Gandum", temperature_celsius: 32.0, humidity_percent: 62.9 }
⚡ InfluxDB: data written
⚠ Ethereum: tx sent: PendingTransaction { tx_hash: 0x64f86333893d09bcc151d82bd0a5369dfeea58eb49187a54ffac8c3d26d898ba, confirmations: 1, state: PendingTxState { state: "InitialDelay" } }
⚡ New connection from 192.168.1.66:39576
⚠ Received sensor data: SensorData { timestamp: "2025-06-18T16:07:25.814749996+00:00", sensor_id: "SHT20-Gudang Tepung", location: "Ruang Gudang Bahan Baku", process_stage: "Tepung Gandum", temperature_celsius: 32.0, humidity_percent: 62.9 }
⚡ InfluxDB: data written
⚠ Ethereum: tx sent: PendingTransaction { tx_hash: 0x90cf2d7a18846c3595760130edc4700b7f6b846a0eb8acd8e5b3ca9effe725b7, confirmations: 1, state: PendingTxState }
```

Gambar 3.7 Hasil Smart Contract

Pada **Gambar 3.7** menunjukkan hasil eksekusi program Rust yang menjalankan deployment smart contract serta komunikasi data sensor. Setelah program dijalankan menggunakan perintah cargo run, sistem berhasil melakukan deploy smart contract ke jaringan blockchain dan menghasilkan sebuah alamat kontrak, yaitu 0x5a498fef5aeb53f662053ae3514b4d093e062dcd. Alamat ini merupakan identitas unik smart contract yang telah tersimpan secara permanen di jaringan blockchain, dan

digunakan untuk semua interaksi selanjutnya dengan kontrak tersebut. Selanjutnya, server TCP yang berjalan pada port 9000 menerima data dari sensor yang terhubung melalui jaringan lokal. Data yang diterima tidak hanya dikirim ke database InfluxDB, tetapi juga direkam ke dalam blockchain melalui transaksi Ethereum. Hal ini dibuktikan dengan munculnya hash transaksi yang mengindikasikan bahwa data sensor telah dicatat ke dalam smart contract dalam bentuk transaksi yang menunggu konfirmasi.



```

Open  ⌂ main.rs -/Tubes IoT/Server/src main.rs
main.rs x script.js main.rs
// --- Ethereum setup ---
let provider = Provider::try_from("http://localhost:8545")?;
let wallet: LocalWallet =
    "0x9730cadefbad7034c9845a7df2ad1450ad2e24bb2aa6df4a8d655f78a5fb81584"
        .parse()?
        .with_chain_id(1337u64);
let client = Arc::new(SignerMiddleware::new(provider, wallet));

// Baca dan parse ABI dan bytecode dengan benar
let abi_str = fs::read_to_string("build/SensorStorage.abi")?;
let bytecode_str = fs::read_to_string("build/SensorStorage.bin")?;

let abi: Abi = serde_json::from_str(&abi_str)?;
let bytecode = bytecode_str.trim().parse()?;

let factory = ContractFactory::new(abi, bytecode, client.clone());

let contract = factory.deploy()?;
println!("✅ Smart contract deployed at: {:?}", contract.address);

```

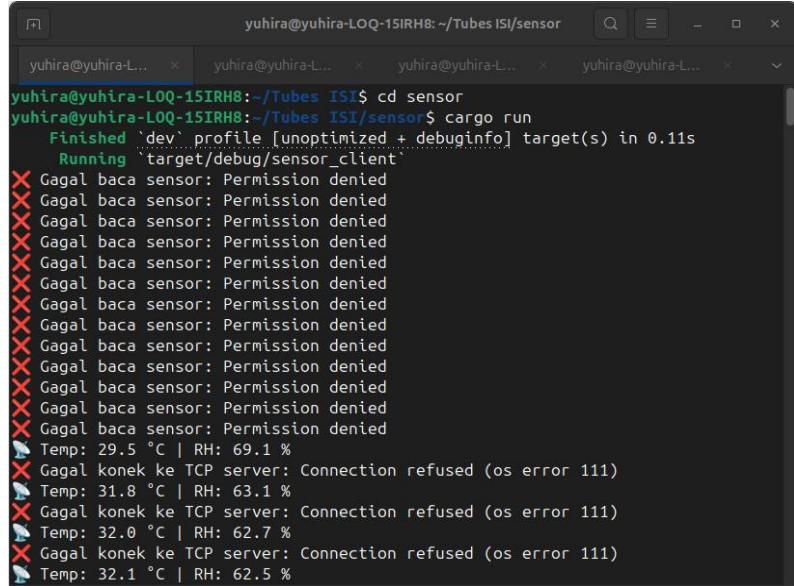
Gambar 3.8

Gambar di atas memperlihatkan bagian kode main.rs pada sisi server yang digunakan untuk mengatur koneksi dengan jaringan Ethereum serta menyiapkan dompet digital (wallet) untuk melakukan transaksi. Pada bagian ini, Provider digunakan untuk menghubungkan server dengan node Ethereum lokal melalui alamat http://localhost:8545. Selanjutnya, dibuat objek LocalWallet dari private key Ethereum, yang berfungsi sebagai identitas digital untuk menandatangani transaksi di blockchain. Wallet ini dikombinasikan dengan provider dalam SignerMiddleware untuk membentuk client, yaitu antarmuka yang dapat digunakan untuk berinteraksi dengan jaringan Ethereum.

Selain itu, file ABI (SensorStorage.abi) dan bytecode (SensorStorage.bin) dari smart contract yang telah dikompilasi sebelumnya dibaca dan di-parse agar bisa digunakan dalam proses deployment. Dengan konfigurasi ini, server dapat melakukan deploy smart contract ke blockchain menggunakan ContractFactory, dan jika berhasil, alamat kontrak akan dicetak sebagai output. Proses ini menunjukkan bagaimana smart contract dikaitkan langsung dengan wallet dan jaringan blockchain melalui kode backend di Rust.

3.4 Pengkomunikasian Seluruh Sistem

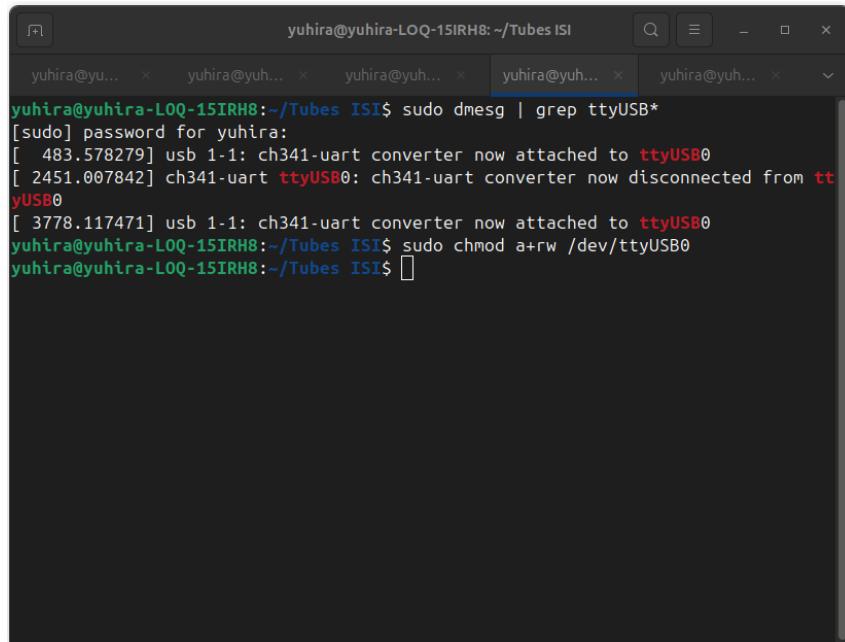
- 1) Klik kanan pada folder kemudian klik ‘Open in Terminal’, kemudian masukkan command cd sensor kemudian enter, lalu masukkan command cargo run lalu enter, untuk membaca kemudian mengirimkan data dari bacaan sensor ke Rust TCP Server. Maka tampilannya akan seperti berikut:



```
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI/sensor$ cd sensor
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI/sensor$ cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.11s
        Running `target/debug/sensor_client`
✖ Gagal baca sensor: Permission denied
✖ Temp: 29.5 °C | RH: 69.1 %
✖ Gagal koneksi ke TCP server: Connection refused (os error 111)
✖ Temp: 31.8 °C | RH: 63.1 %
✖ Gagal koneksi ke TCP server: Connection refused (os error 111)
✖ Temp: 32.0 °C | RH: 62.7 %
✖ Gagal koneksi ke TCP server: Connection refused (os error 111)
✖ Temp: 32.1 °C | RH: 62.5 %
```

terlihat pada tampilan awal tertera “Gagal baca sensor: Permission denied.

- 2) Lalu buat tab baru untuk terminal, kemudian masukkan seperti pada gambar berikut:



```
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI$ sudo dmesg | grep ttyUSB*
[sudo] password for yuhira:
[ 483.578279] usb 1-1: ch341-uart converter now attached to ttyUSB0
[ 2451.007842] ch341-uart ttyUSB0: ch341-uart converter now disconnected from ttyUSB0
[ 3778.117471] usb 1-1: ch341-uart converter now attached to ttyUSB0
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI$ sudo chmod a+r /dev/ttyUSB0
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI$ 
```

- command sudo dmesg | grep ttyUSB*, berguna untuk mengidentifikasi koneksi dari USB konverter dari modbus sensor SHT20.
- kemudian command sudo chmod a+rw /dev/ttyUSB0, berguna untuk mendapatkan permission untuk membaca data dari sensor. Setelah memasukkan command diatas maka tampilan dari terminalnya akan berubah menjadi seperti ini:

```

yuhira@yuhira-L... x yuhira@yuhira-L... x yuhira@yuhira-L... x yuhira@yuhira-L... x
Temp: 29.5 °C | RH: 69.1 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 31.8 °C | RH: 63.1 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.0 °C | RH: 62.7 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.1 °C | RH: 62.5 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.3 °C | RH: 61.9 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.4 °C | RH: 61.2 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.6 °C | RH: 60.7 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.7 °C | RH: 60.5 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 32.9 °C | RH: 60.2 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 33.0 °C | RH: 59.6 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 33.1 °C | RH: 59.0 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)
Temp: 33.3 °C | RH: 58.6 %
X Gagal koneksi ke TCP server: Connection refused (os error 111)

```

- 3) Lalu buat tab baru untuk terminal, kemudian masukkan command ganache, kemudian akan muncul tampilan seperti gambar berikut:

```

yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI$ ganache
This version of μWS is not compatible with your Node.js build:

Error: Cannot find module '../binaries/uws_linux_x64_109.node'
Require stack:
- /home/yuhira/.npm-global/lib/node_modules/ganache/node_modules/@trufflesuite/uws-js-unofficial/src/uws.js
- /home/yuhira/.npm-global/lib/node_modules/ganache/dist/node/cli.js
Falling back to a NodeJS implementation; performance may be degraded.

ganache v7.9.2 (@ganache/cli: 0.10.2, @ganache/core: 0.10.2)
Starting RPC server

Available Accounts
=====
(0) 0xd10d6CC7dc8Ab2E282930b2b7274892Fe86FbDf0 (1000 ETH)
(1) 0x2C660bfEfE83e6Ced0b8c304de88ecd3ac0722E2 (1000 ETH)
(2) 0x81763054cCaB38E9937864E74e0E6C7B88A8c8Bc (1000 ETH)
(3) 0x7B7A276F17350DC4a32b1758EC5f6E0aD7123b7D (1000 ETH)
(4) 0x623afAfC65D0EfB7235b601C5efaF939F1368E7e (1000 ETH)
(5) 0x3549484fcCB95B4779ae5861f336304cC2260b5E3 (1000 ETH)
(6) 0xE9b34033A28ba3e4A3b052FbDc32DA3fb29B5DE1 (1000 ETH)
(7) 0x17Ed9Bb31b6848086a242a899c733986f4c6dfDe (1000 ETH)

```

setelah itu salin salah satu Private Keys yang diberikan dari tampilan tersebut

```

yuhira@yuhira-LOQ-15IRH8: ~/Tubes ISI
=====
(0) 0xd10d6CC7dc8AbE282930b2b7274892Fe86FbDf0 (1000 ETH)
(1) 0x2C660bfEfE83e6Ced0b8c304de88ecd3ac0722E2 (1000 ETH)
(2) 0x81763054cCaB38E9937864E74e0E6C7B88A8c8Bc (1000 ETH)
(3) 0x7B7A276F17350DC4a32b1758EC5f6E0aD7123b7D (1000 ETH)
(4) 0x623afAfc65D0EFb7235b601C5efaF939F1368E7e (1000 ETH)
(5) 0x3549484fCB95B4779ae5861f336304cC2260b5E3 (1000 ETH)
(6) 0xE9b34033A28ba3e4A3b052FbDc32DA3fb29B5DE1 (1000 ETH)
(7) 0x17Ed9Bb31b6848086a242a899c733986f4c6dfDe (1000 ETH)
(8) 0xf114a2b75af54fC2504322CE65E48B4dB213172c (1000 ETH)
(9) 0xb41f90B64Dc79e76D9890C53BA7A7A8d48E22bCD (1000 ETH)

Private Keys
=====
(0) 0x2fd4495a3194aa91a35384fa7816c36d12db9d19412c2ede528e1539bf8290e8
(1) 0x2daba510bec7af551359cf274d2b97f47f7518e0141678dfe9a5e7890899c0
(2) 0xd8ad54737160ce602671600f190df81cebffb05a34dcccb193b0bb062173c585
(3) 0xa93c67a96970989b4b70afa2fc95136970d14d4d32145bac149bab6f2d9e1af
(4) 0xb0f8c6651627b1cbf7408fd2a077e66740fb9382d085d5b1c8f39ff2feb633da
(5) 0x5e05a35843f934f83a8ed1e5e52886887c1cc96617d0d159c2c03e3018bf11b
(6) 0x292942cbd7ecd5f291ad344779d5707db9e26954a00c460f3a5f8a50115d201a
(7) 0xf1849197d11e59b9b71aedd8881fefcc1f650da569d55aacc33dee38573a1fe6
(8) 0xa85daad8c1671446b054e768ecc6b6c4c35d35c869441ffbe8a77fd3cb254e28
(9) 0x5bfc7d3cd549974e11d568f8a87b4accd3acfbbd019ad92c8cc5be63e1e9acaa

```

- 4) Kemudian buka file `main.rs` yang terdapat di `/Server/src`. Kemudian ganti local wallet yang ada di bawah ‘let wallet: LocalWallet =’ kemudian save progressnya.

```

// --- Ethereum setup ---
let provider = Provider::try_from("http://localhost:8545")?;
let wallet: LocalWallet =
    .parse::LocalWallet()?;
    .with_chain_id(1337u64);
let client = Arc::new(SignerMiddleware::new(provider, wallet));

// Baca dan parse ABI dan bytecode dengan benar
let abi_str = fs::read_to_string("build/SensorStorage.abi")?;
let bytecode_str = fs::read_to_string("build/SensorStorage.bin")?;

let abi: Abi = serde_json::from_str(&abi_str)?;
let bytecode = bytecode_str.trim().parse::Bytes)?;

let factory = ContractFactory::new(abi, bytecode, client.clone());

let contract = factory.deploy()?.send().await?;
println!("Smart contract deployed at: {:?}", contract.address);

```

- 5) Klik untuk tambahkan tab baru dari Terminal, kemudian masukkan command `cd Server` kemudian enter, lalu masukkan command `cargo run` lalu enter, untuk menjalankan Rust TCP Server dan mengirimkan data dari bacaan sensor yang dikirim ke TCP Server juga dikirimkan ke InfluxDB. Tampilan terminal akan seperti berikut:

```

yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI/sensor$ cd ..
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI$ cd Server
yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI/Server$ cargo run
    Compiling sensor_server_blockchain v0.1.0 (/home/yuhira/Tubes ISI/Server)
      Finished 'dev' profile [unoptimized + debuginfo] target(s) in 7.11s
        Running `target/debug/sensor_server_blockchain`
✓ Smart contract deployed at: 0xa08f4f5be82ca1b5483d65504b76b147d26510e6
容忍 TCP Server listening on port 9000...
容忍 New connection from 127.0.0.1:36576
容忍 Received sensor data: SensorData { timestamp: "2025-06-18T17:58:37.217576137+00:00", sensor_id: "SHT20-Gudang Tepung", location: "Ruang Gudang Bahan Baku", process_stage: "Tepung Gandum", temperature_celsius: 34.3, humidity_percent: 55.9 }
容忍 InfluxDB: data written
容忍 Ethereum: tx sent: PendingTransaction { tx_hash: 0xc5b9e3563943d4697655e6ffc73810b9ec08f0615730eff63494b422eb715efa, confirmations: 1, state: PendingTxState { state: "InitialDelay" } }
容忍 New connection from 127.0.0.1:43574
容忍 Received sensor data: SensorData { timestamp: "2025-06-18T17:58:42.246713017+00:00", sensor_id: "SHT20-Gudang Tepung", location: "Ruang Gudang Bahan Baku", process_stage: "Tepung Gandum", temperature_celsius: 34.4, humidity_percent: 55.5 }
容忍 InfluxDB: data written
容忍 Ethereum: tx sent: PendingTransaction { tx_hash: 0xe429d39cd22b2354979216fe }

```

Setelah itu salin Smart contract yang tertera pada hasil tampilan terminal

- 6) Kemudian buka [script.js](#) yang ada di /Web3 with features, ganti contract Adress yang ada di dalamnya(yang sudah disorot) dengan Smart contract yang sudah disalin tadi lalu save progressnya.

```

const contractAddress =
"0xb9286162c759ae4e7855b5a4390cda44940e10e2"; // Ganti dengan alamat
kontrak dari Ganache
const abiPath = "abi/SensorStorage.abi";

let chart;

async function loadSensorData() {
  const abiRes = await fetch(abiPath);
  const abi = await abiRes.json();

  const provider = new ethers.BrowserProvider(window.ethereum ||
"http://localhost:8545");
  await provider.send("eth_requestAccounts", []);
  const signer = await provider.getSigner();
  const contract = new ethers.Contract(contractAddress, abi, signer);

  const filter = contract.filters.DataStored();
  const events = await contract.queryFilter(filter, 0, "latest");
}

```

- 7) Kemudian klik kanan pada Folder ‘Web3 with feature’ lalu klik open in terminal, kemudian jalankan command ‘python3 -m http.server 8005’ kemudian enter, untuk menjalankan halaman Web3 dari program sistem.

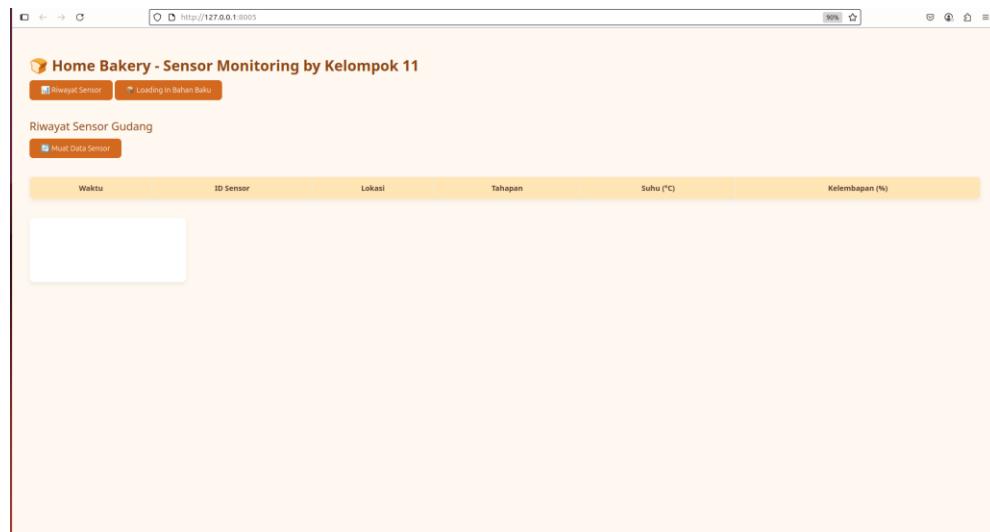
```

yuhira@yuhira-LOQ-15IRH8:~/Tubes ISI/Web3 with feature$ python3 -m http.server 8005
Serving HTTP on 0.0.0.0 port 8005 (http://0.0.0.0:8005/) ...
127.0.0.1 - - [19/Jun/2025 01:44:14] "GET / HTTP/1.1" 304 -
127.0.0.1 - - [19/Jun/2025 01:44:14] "GET /style1.css HTTP/1.1" 200 -
-----
Exception occurred during processing of request from ('127.0.0.1', 35060)
Traceback (most recent call last):
  File "/usr/lib/python3.12/socketserver.py", line 692, in process_request_thread
    self.finish_request(request, client_address)
  File "/usr/lib/python3.12/http/server.py", line 1311, in finish_request
    self.RequestHandlerClass(request, client_address, self,
  File "/usr/lib/python3.12/http/server.py", line 672, in __init__
    super().__init__(*args, **kwargs)
  File "/usr/lib/python3.12/socketserver.py", line 761, in __init__
    self.handle()
  File "/usr/lib/python3.12/http/server.py", line 436, in handle
    self.handle_one_request()
  File "/usr/lib/python3.12/http/server.py", line 424, in handle_one_request
    method()
  File "/usr/lib/python3.12/http/server.py", line 679, in do_GET
    self.copyfile(f, self.wfile)
  File "/usr/lib/python3.12/http/server.py", line 878, in copyfile

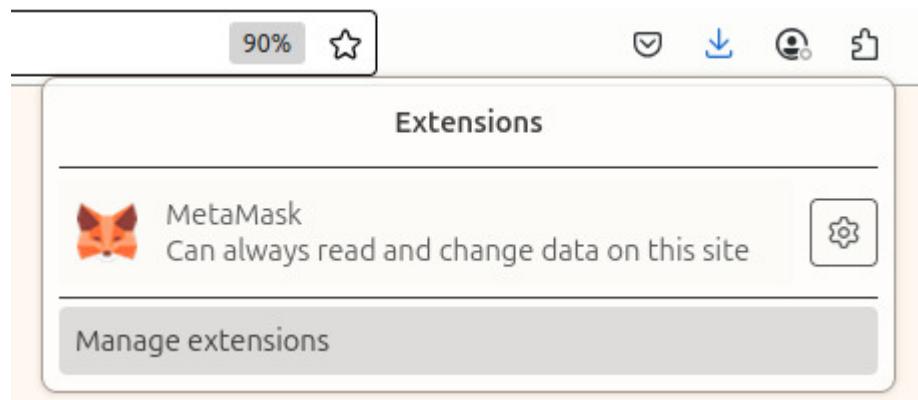
```

setelah itu klik kanan pada (<http://0.0.0.0:8005/>) kemudian open link

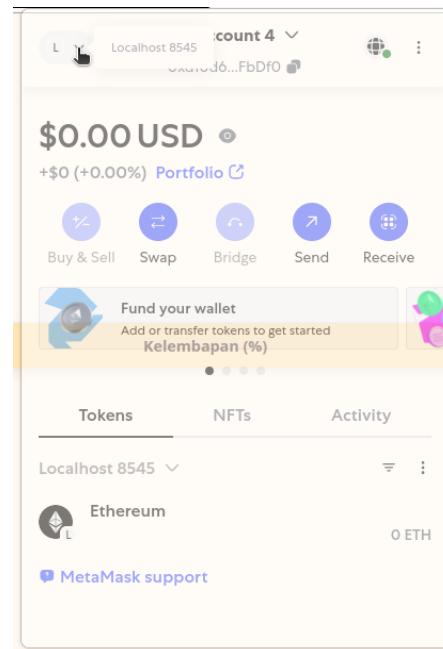
- 8) Kemudian akan diarahkan ke pada tampilan berikut.



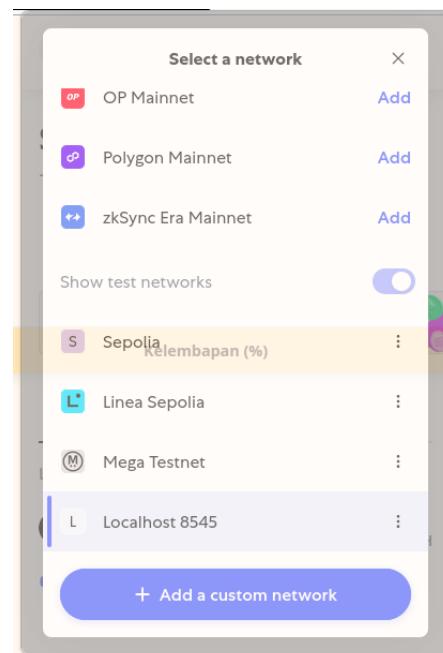
- 9) Kemudian klik pada extension di dalam firefox lalu klik pada Metamask untuk menghubungkan blockchain.



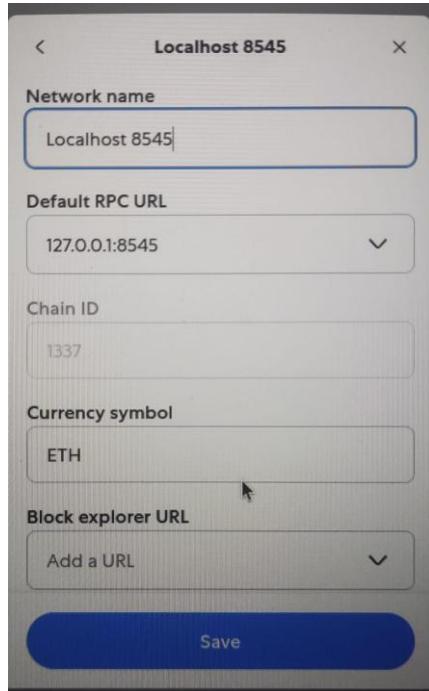
- Setelah itu klik pada tampilan kiri,



- Lalu klik 'add a custom network'

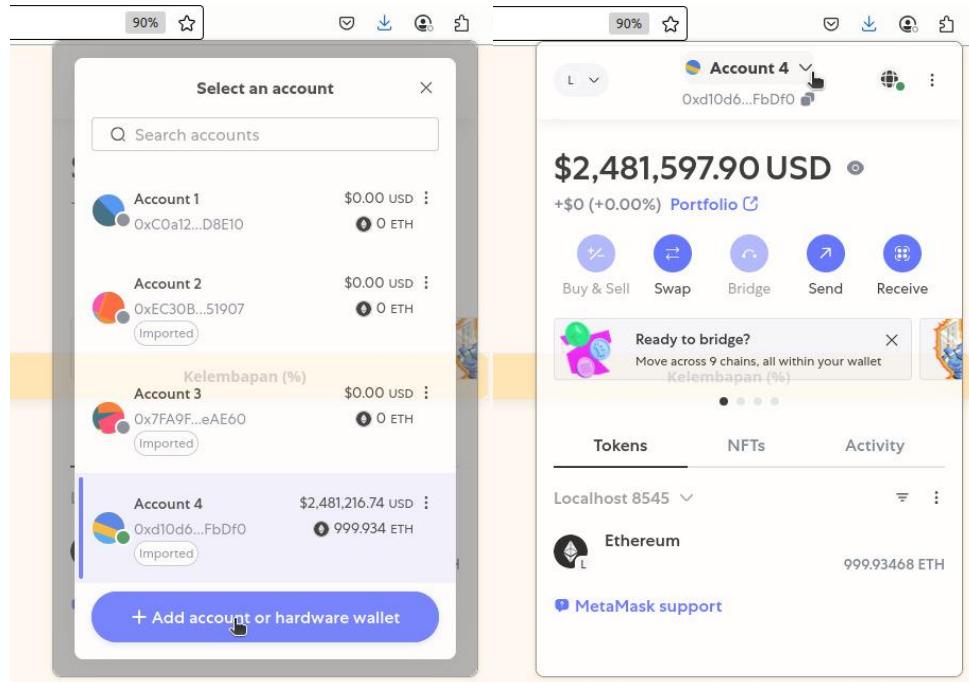


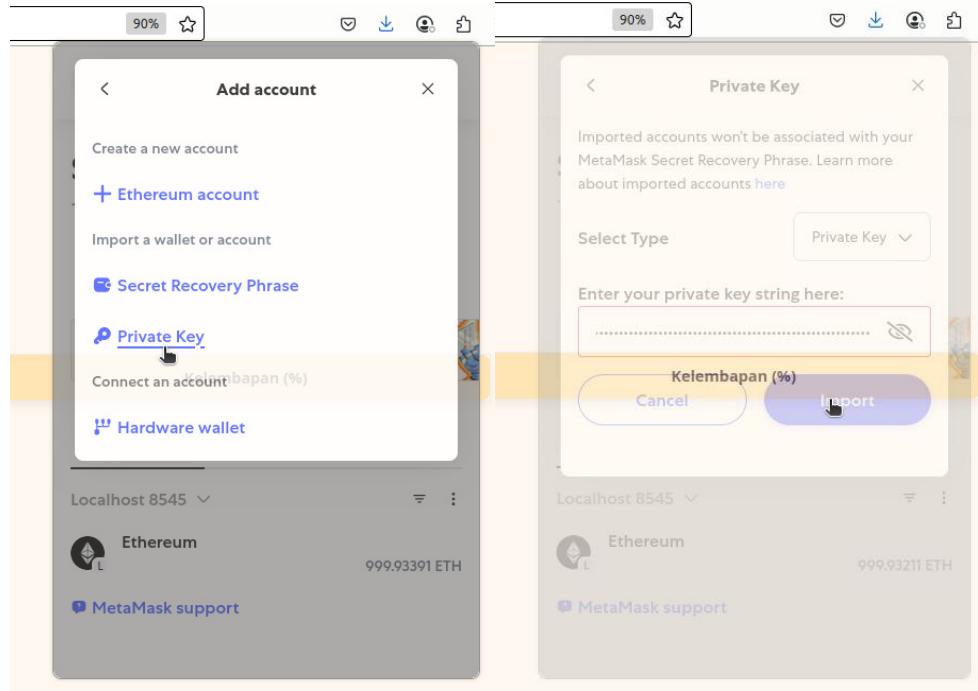
- c. Lalu sesuaikan dengan yang ada digambar berikut:



kemudian klik save.

- d. kemudian klik panah ke bawah di sebelah account, lalu klik ‘add account or hardware wallet, lalu klik private keys, Kemudian isikan Private Keys yang didapatkan dari ganache, kemudian klik import.





10) 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

The Metamask extension interface shows:

- Extension: (MetaMask) - Met...
- Accounts tab selected.
- Account 4: 0xd10d6...FbDf0 (Imported)
- Balance: \$2,475,144.16 USD / 999.93211 ETH
- Permissions tab (not selected).
- Buttons: Cancel, Connect.

The browser window shows:

- Address bar: http://127.0.0.1:8005
- Title: Home Bakery - Sensor Monitoring by Kelompok 11
- Buttons: Riwayat Sensor, Loading In Bahan Baku.
- Section: Riwayat Sensor Gudang
- Button: Muat Data Sensor.

Home Bakery - Sensor Monitoring by Kelompok 11

Anayalt Sensor Loading in Bahan Baku

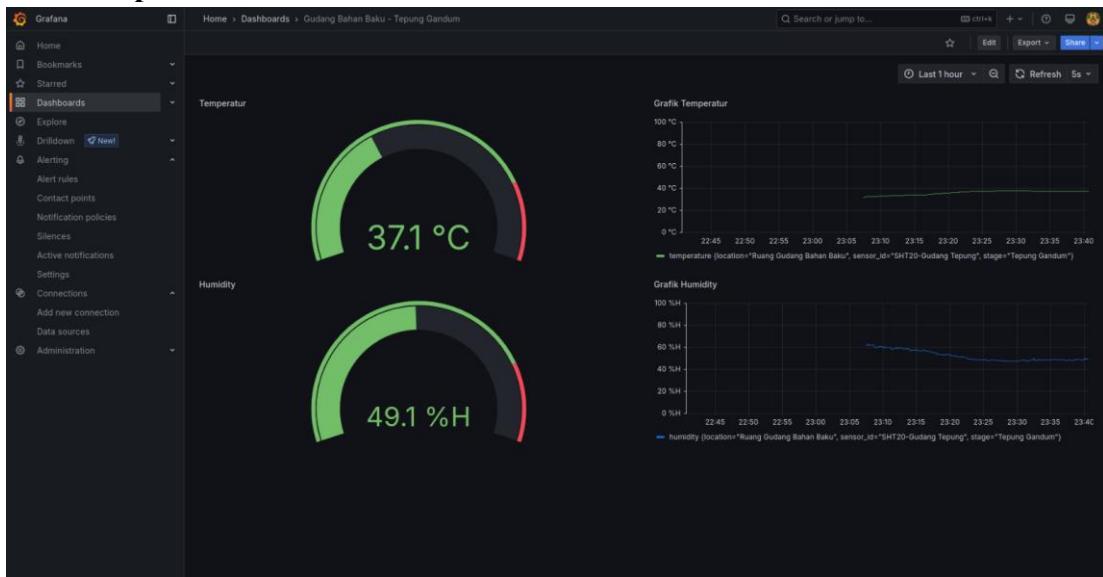
Riwayat Sensor Gudang

Muat Data Sensor

Waktu	ID Sensor	Lokasi	Tahapan	Suhu (°C)	Kelembaban (%)
6/19/2025, 12:58:37 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.30	55.90
6/19/2025, 12:58:42 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.40	55.50
6/19/2025, 12:58:47 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.50	55.40
6/19/2025, 12:58:52 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.70	55.10
6/19/2025, 12:58:57 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.80	54.80
6/19/2025, 12:59:02 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.90	54.60
6/19/2025, 12:59:07 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.00	54.20
6/19/2025, 12:59:12 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.09	53.90
6/19/2025, 12:59:17 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.20	53.70
6/19/2025, 12:59:22 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.40	53.50
6/19/2025, 12:59:27 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.50	53.20
6/19/2025, 12:59:32 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.59	53.10
6/19/2025, 12:59:37 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.70	52.70
6/19/2025, 12:59:42 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.80	52.40

BAB IV HASIL DAN PEMBAHASAN

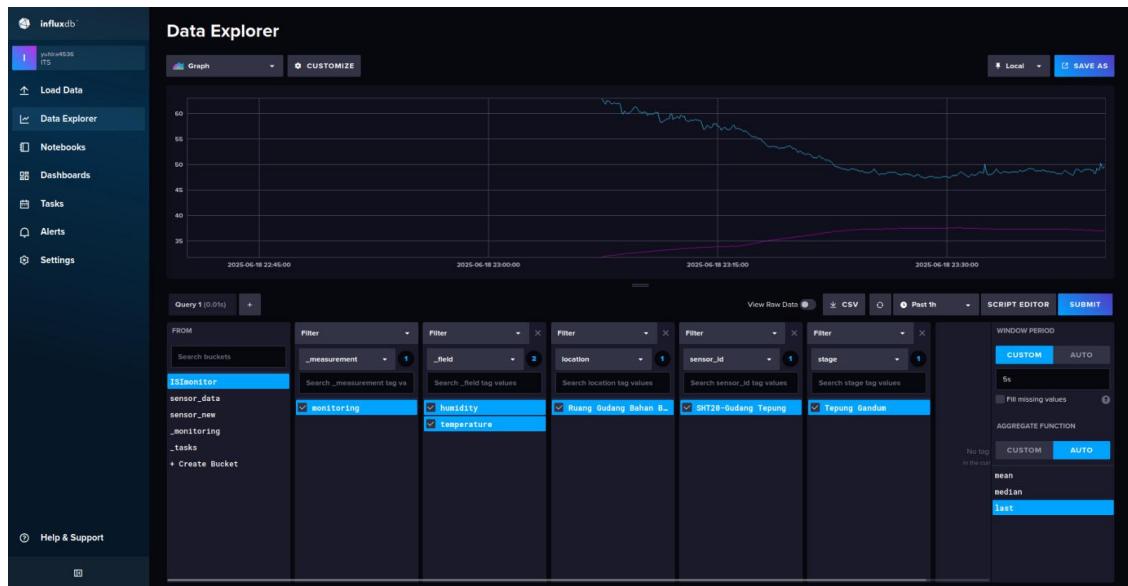
4.1 Hasil Tampilan Grafana



Gambar 4.1 Tampilan Grafik pada Grafana

Dapat dilihat pada **Gambar 4.1** merupakan hasil tampilan grafik pada grafana yang berperan sebagai *interface* visual untuk menampilkan data temperatur dan kelembaban secara *real-time* yang dikirim dari sensor SHT20 dari data yang ada dalam influxDB. Dashboard yang dibuat memberikan grafik time-series chart sehingga pengguna dapat melihat perubahan temperatur dan kelembaban selama proses penyimpanan berlangsung.

4.2 InfluxDB



Gambar 4.2 Tampilan Grafik InfluxDB

The screenshot shows a Data Explorer interface with a table of data. The columns are: timestamp, field, value, unit, location, and two additional columns. The data is filtered by 'field = humidity_measurement' and 'monitoring location = Ruang Gudang Bahan Baku'. The table contains approximately 20 rows of data.

2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:00..	63.80	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:05..	63.30	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:10..	63.20	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:15..	63.30	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:20..	63.30	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:25..	63.30	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:30..	63.20	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:35..	63.30	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:40..	63.40	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:45..	63.20	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum
2025-06-19 08:07:54..	2025-06-19 09:07:54..	2025-06-19 09:07:50..	63.20	humidity	monitoring	Ruang Gudang Bahan Baku, SHT20-Gudang Tepung, Tepung Gandum

Gambar 4.3 Tampilan Tabel data InfluxDB

Dapat dilihat pada **Gambar 4.2** merupakan hasil tampilan grafik pembacaan sensor berdasarkan timestamp pada InfluxDB, dan pada **Gambar 4.3** merupakan tabel data hasil pembacaan sensor berdasarkan timestamp pada InfluxDB. Dalam influxDB berguna untuk menyimpan data pembacaan sensor atau sebagai database yang dikirim dari mikrokontroler dan sensor melalui protokol TCP, lalu akan disimpan dalam struktur data berbasis waktu. Dalam sistem monitoring ini, influxDB yang bertanggung jawab merekam dan menyimpan data, setiap perubahan lingkungan sekecil apapun akan tercatat.

4.3 Blockchain dan Web3

The screenshot shows a web-based monitoring application titled 'Home Bakery - Sensor Monitoring by Kelompok 11'. It displays a table of sensor data with columns: Waktu (Timestamp), ID Sensor, Lokasi (Location), Tahapan (Phase), Suhu (°C) (Temperature), and Kelembapan (%) (Humidity). The data is updated in real-time, as indicated by the 'Loading In Bahan Baku' status bar message. The table shows multiple entries for the SHT20 sensor at different times, with values fluctuating between 34.30°C and 35.80°C and 53.90% to 55.90% humidity.

Waktu	ID Sensor	Lokasi	Tahapan	Suhu (°C)	Kelembapan (%)
6/19/2025, 12:58:37 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.30	55.90
6/19/2025, 12:58:42 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.40	55.50
6/19/2025, 12:58:47 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.50	55.40
6/19/2025, 12:58:52 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.70	55.10
6/19/2025, 12:58:57 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.80	54.80
6/19/2025, 12:59:02 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	34.90	54.60
6/19/2025, 12:59:07 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.00	54.20
6/19/2025, 12:59:12 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.09	53.90
6/19/2025, 12:59:17 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.20	53.70
6/19/2025, 12:59:22 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.40	53.50
6/19/2025, 12:59:27 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.50	53.20
6/19/2025, 12:59:32 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.59	53.10
6/19/2025, 12:59:37 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.70	52.70
6/19/2025, 12:59:42 AM	SHT20-Gudang Tepung	Ruang Gudang Bahan Baku	Tepung Gandum	35.80	52.40

Gambar 4.4 Tampilan Data pada Web3 yang diintegrasikan dengan Blockchain

Pada **Gambar 4.4** merupakan hasil pembacaan tampilan data secara *realtime* dari sensor SHT20 yang diintegrasikan ke dalam sistem blockchain. Setiap data yang ditampilkan akan dicatat sebagai transaksi dalam jaringan blockchain. Integrasi blockchain pada sistem ini memberikan jaminan keaslian dan integritas data.



Gambar 4.5 Tampilan Grafik pada Web3

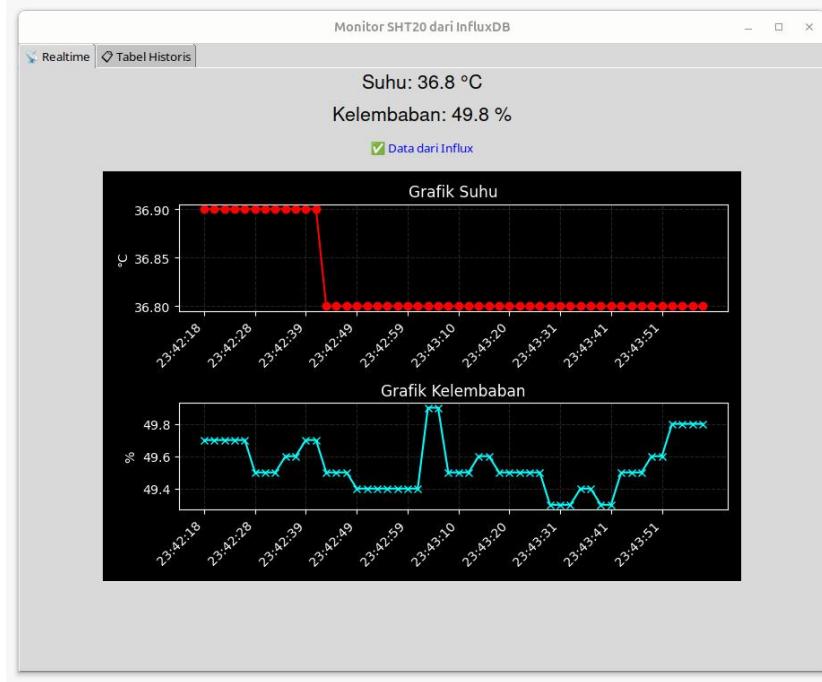
Pada **Gambar 4.5** menunjukkan hasil visualisasi grafik temperatur dan kelembapan yang merupakan bagian dari sistem blockchain yang diintegrasikan ke Web3. Terdapat dua grafik yang tertera dalam satu tampilan, garis merah menunjukkan grafik temperatur dalam derajat celcius (°C), dan garis biru menunjukkan grafik kelembapan dalam persentase (%).



Gambar 4.6 Tampilan halaman Fitur Loading in Bahan Baku

Pada **Gambar 4.6** menunjukkan fitur dari Web3 yang digunakan sebagai pengonfirmasian bahan baku yang sedang melakukan *loading in* ke gudang bahan baku, yang kemudian ketika dikonfirmasikan maka akan menampilkan status dari data sensor dari waktu tersebut.

4.4 Hasil Interface pada Qt



Gambar 4.7 Tampilan Grafik pada PyQt

Dalam **Gambar 4.7** merupakan tampilan hasil grafik tampilan pada software Qt, disini menampilkan data temperatur dan kelembapan yang dikirimkan dari influxDB secara lokal. Pada Qt ini, bertujuan untuk menampilkan dalam bentuk grafik sederhana yang mudah dipahami pengguna untuk melihat data historis dalam bentuk dua grafik terpisah, grafik merah merupakan grafik untuk tampilan data Suhu dan grafik biru merupakan grafik untuk tampilan data kelembaban. Penggunaan warna yang berbeda untuk tampilan grafik ini bertujuan untuk membantu membedakan dua parameter yang berbeda tersebut. Integrasi langsung dengan influxDB memastikan bahwa data selalu ter-update secara *real-time*.

Monitor SHT20 dari InfluxDB

Waktu	Suhu (°C)	Kelembaban (%)
16:42:32	36.9	49.6
16:42:37	36.9	49.7
16:42:43	36.8	49.5
16:42:48	36.8	49.4
16:42:53	36.8	49.4
16:42:58	36.8	49.4
16:43:03	36.8	49.9
16:43:08	36.8	49.5
16:43:13	36.8	49.6
16:43:18	36.8	49.5
16:43:23	36.8	49.5
16:43:28	36.8	49.3
16:43:33	36.8	49.4
16:43:38	36.8	49.3
16:43:43	36.8	49.5
16:43:48	36.8	49.6
16:43:53	36.8	49.8
16:43:58	36.8	49.8
16:44:03	36.8	49.9
16:44:08	36.8	50.1

Gambar 4.8 Tampilan Data dalam Tabel Historis pada PyQt

Dalam **Gambar 4.8** menunjukkan tampilan Tabel Historis dari Qt yang berguna untuk menampilkan daya sensor suhu dan kelembaban. Dalam setiap tabel mencatat *timestamp* secara *realtime*, Suhu dalam °C dan kelembaban dalam %.

BAB V KESIMPULAN

5.1 Kesimpulan

Sistem monitoring yang dirancang telah berhasil menggabungkan sensor SHT20, komunikasi Modbus RTU, dan server TCP berbasis Rust untuk memantau suhu dan kelembapan ruang penyimpanan bahan secara real-time. Data sensor dikirim ke server melalui jaringan lokal dan disimpan dalam database InfluxDB, kemudian divisualisasikan menggunakan Grafana dan PyQt untuk memberikan tampilan data yang mudah dipahami. Konfigurasi alamat IP lokal memastikan komunikasi antara perangkat berjalan dengan lancar dalam jaringan yang sama.

Selain itu, sistem ini juga telah terintegrasi dengan teknologi blockchain melalui implementasi smart contract menggunakan Ganache dan MetaMask. Setiap data sensor yang dikirimkan tercatat sebagai transaksi dalam jaringan blockchain, sehingga keaslian dan keterlacakkan data dapat dijamin. Dengan pendekatan ini, sistem monitoring tidak hanya mampu meningkatkan keandalan pemantauan kondisi lingkungan, tetapi juga memberikan nilai tambah melalui pencatatan data yang transparan dan aman.

DAFTAR PUSTAKA

- Cavoj, S., Nikitin, I., Perkins, C., & Dardha, O. (2024). Session Types for the Transport Layer: Towards an Implementation of TCP. In D. Costa & R. Hu (Eds.), Programming Language Approaches to Concurrency and Communication-centric Software (PLACES'24), EPTCS 401, 22–36. <https://doi.org/10.4204/EPTCS.401.3>
- Dedy, H., Axel, N. S., & Ivan, P. P. (2021). *Sistem monitoring suhu dan kelembapan udara menggunakan protokol MQTT berbasis Wemos D1 Mini*. Seminar Nasional AVoER XIII, Universitas Sriwijaya.
- Di Pierro, M. (2017). What Is the Blockchain? Computing in Science & Engineering, 19(5), 92–95. IEEE. <https://doi.org/10.1109/MCSE.2017.3421554>
- Khumaidi, A., Hasin, M. K., Pujiputra, A. P., Irsyad, S. M., Rinanto, N., Rachman, I., Budi, P. S., Malik, A. T., & Bayu, N. B. (2024). Implementation of integrated temperature, humidity, and dust monitoring system on building electrical panel. Journal of Soft Computing Exploration, 5(4), 342–352. <https://doi.org/10.52465/joscex.v5i4.483>
- Khan, A., Hameed, A. A., Nand, P., Jamil, A., & Bhushan, B. (2024). A review of blockchain-based decentralised authentication solutions and their improvement through MetaMask. In 2024 IEEE 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings) (pp. 1–9). IEEE. <https://doi.org/10.1109/AIBThings63359.2024.10863348>
- Khozainuz Zuhri, K., Ihkwan, A., & Fahurian, F. (2021). Sistem monitoring suhu dan kelembapan pada ruang penyimpanan roti berbasis Internet of Things (IoT). *Jurnal Teknologi dan Informatika (JEDA)*, 2(1), 1–5.
- Mattsson, C. (2024). Visualisering av realtidsdata i Grafana [Bachelor's thesis, Yrkeshögskolan Novia]. Vasa.
- Naqvi, S. N. Z., & Yfantidou, S. (2017). Time Series Databases and InfluxDB. Université libre de Bruxelles – Advanced Databases, Winter Semester 2017–2018. Retrieved from <https://www.influxdata.com/>
- Pérez García, M., & Mejía Sánchez, D. A. (2021). Improving firmware development through the Rust Programming Language (Tesis spesialisasi, Instituto Tecnológico y de Estudios Superiores de Occidente). Retrieved from <https://github.com/miker1423/article-demo>
- Sheridan, D., Harris, J., Wear, F., Cowell Jr, J., Wong, E., & Yazdinejad, A. (2022). Web3: Challenges and Opportunities for the Market. Kennesaw State University & University of Guelph. <https://doi.org/10.48550/arXiv.2209.02446>

- Soliev, B. N., & G‘iyosiddinov, N. (2024). Optimizing PyQt5 development with Qt Designer. *Journal of Python GUI Development*, 2(4), 254–259.
- Wijaya, A. E., & Nurjaman, H. (2020). Implementasi metode weighted product dalam memonitor gudang penyimpanan roti berbasis Internet of Thing pada platform Node-Red. *Jurnal Teknologi Informasi dan Komunikasi STMIK Subang*, 13(1), 1–5.

LAMPIRAN

Source Kode untuk main.rs dari sensor/srs

```
use tokio_modbus::{client::rtu, prelude::*};
use tokio_serial::{SerialPortBuilderExt, Parity, StopBits, DataBits};
use tokio::net::TcpStream;
use tokio::io::AsyncWriteExt;
use serde::Serialize;
use chrono::Utc;
use std::error::Error;
use tokio::time::{sleep, Duration};

#[derive(Serialize)]
struct SensorData {
    timestamp: String,
    sensor_id: String,
    location: String,
    process_stage: String,
    temperature_celsius: f32,
    humidity_percent: f32,
}

async fn read_sensor(slave: u8) -> Result<Vec<u16>, Box<dyn Error>> {
    let builder = tokio_serial::new("/dev/ttyUSB0", 9600)
        .parity(Parity::None)
        .stop_bits(StopBits::One)
        .data_bits(DataBits::Eight)
        .timeout(std::time::Duration::from_secs(1));

    let port = builder.open_native_async()?;
    let mut ctx = rtu::connect_slave(port, Slave(slave)).await?;
    let response = ctx.read_input_registers(1, 2).await?;

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn Error>> {
    loop {
        match read_sensor(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} %", temp, rh);

                let data = SensorData {
                    timestamp: Utc::now().to_rfc3339(),

```

```

        sensor_id: "SHT20-Gudang Tepung".into(),
        location: "Ruang Gudang Bahan Baku".into(),
        process_stage: "Tepung Gandum".into(),
        temperature_celsius: temp,
        humidity_percent: rh,
    };

    let json = serde_json::to_string(&data)?;

    match TcpStream::connect("127.0.0.1:9000").await {
        Ok(mut stream) => {
            stream.write_all(json.as_bytes()).await?;
            stream.write_all(b"\n").await?;

            println!("✅ Data dikirim ke TCP server");
        },
        Err(e) => {
            println!("❌ Gagal koneksi ke TCP server: {}", e);
        }
    },
    Ok(other) => {
        println!("⚠️ Data tidak lengkap: {:?}", other);
    },
    Err(e) => {
        println!("❌ Gagal baca sensor: {}", e);
    }
}

sleep(Duration::from_secs(5)).await;
}
}

```

Source Code untuk main.rs dari Server/src

```

use tokio::net::TcpListener;
use tokio::io::{AsyncBufReadExt, BufReader};
use serde::Deserialize;
use reqwest::Client;

use ethers::prelude::*;
use ethers::abi::Abi;
use std::fs, sync::Arc;
use chrono::DateTime;

#[derive(Deserialize, Debug)]

```

```

struct SensorData {
    timestamp: String,
    sensor_id: String,
    location: String,
    process_stage: String,
    temperature_celsius: f32,
    humidity_percent: f32,
}

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    // --- InfluxDB setup ---
    let influx_url =
        "http://localhost:8086/api/v2/write?org=ITS&bucket=ISImonitor&precision=s";
    let influx_token = "mTjO4KT4rlVLw3IISNiewi_JWxiErLd67bxRbx_oSrHmVz8-
eTjcqT_IxBxEoTSJRud40uBsz9uaPC7aYMOrwZQ==";
    let http_client = Client::new();

    // --- Ethereum setup ---
    let provider = Provider::try_from("http://localhost:8545")?;
    let wallet: LocalWallet =
        "0x3139805a7b405bb7bd369897e0538cafcb156885b1933e9932fd35eab3da35bc"
            .parse::<LocalWallet>()?
            .with_chain_id(1337u64);
    let client = Arc::new(SignerMiddleware::new(provider, wallet));

    // Baca dan parse ABI dan bytecode dengan benar
    let abi_str = fs::read_to_string("build/SensorStorage.abi")?;
    let bytecode_str = fs::read_to_string("build/SensorStorage.bin")?;

    let abi: Abi = serde_json::from_str(&abi_str)?;
    let bytecode = bytecode_str.trim().parse::<Bytes>()?;

    let factory = ContractFactory::new(abi, bytecode, client.clone());

    let contract = factory.deploy()?;
    println!("✅ Smart contract deployed at: {:?}", contract.address());

    // --- TCP Server ---
    let listener = TcpListener::bind("0.0.0.0:9000").await?;
    println!("💡 TCP Server listening on port 9000...");

    loop {
        let (socket, addr) = listener.accept().await?;
        println!("🔌 New connection from {}", addr);

        let influx_url = influx_url.to_string();
        let influx_token = influx_token.to_string();
        let http_client = http_client.clone();
    }
}

```

```

let contract = contract.clone();

tokio::spawn(async move {
    let reader = BufReader::new(socket);
    let mut lines = reader.lines();

    while let Ok(Some(line)) = lines.next_line().await {
        match serde_json::from_str::<SensorData>(&line) {
            Ok(data) => {
                println!("🕒 Received sensor data: {:?}", data);

                // --- InfluxDB Write ---
                let timestamp = DateTime::parse_from_rfc3339(&data.timestamp)
                    .unwrap()
                    .timestamp();

                let line_protocol = format!(
                    "monitoring,sensor_id={},location={},stage={}
temperature={},humidity={}\n",
                    data.sensor_id.replace(" ", "\\"),  

                    data.location.replace(" ", "\\"),  

                    data.process_stage.replace(" ", "\\"),  

                    data.temperature_celsius,  

                    data.humidity_percent,  

                    timestamp
                );
            }
        }
    }
}

match http_client
    .post(&influx_url)
    .header("Authorization", format!("Token {}", influx_token))
    .header("Content-Type", "text/plain")
    .body(line_protocol)
    .send()
    .await
{
    Ok(resp) if resp.status().is_success() => {
        println!("✅ InfluxDB: data written");
    }
    Ok(resp) => {
        println!("⚠️ InfluxDB error: {}", resp.status());
    }
    Err(e) => {
        println!("❌ InfluxDB HTTP error: {}", e);
    }
}

// --- Ethereum Contract Write ---
let method_call = contract
    .method::<_, H256>("storeData", (

```

```
        timestamp as u64,  
        data.sensor_id.clone(),  
        data.location.clone(),  
        data.process_stage.clone(),  
        (data.temperature_celsius * 100.0) as i64,  
        (data.humidity_percent * 100.0) as i64,  
    ))  
.unwrap();  
  
let tx = method_call.send().await;  
  
        match tx {  
            Ok(pending_tx) => {  
                println!("📡 Ethereum: tx sent: {:?}", pending_tx);  
            }  
            Err(e) => {  
                println!("❌ Ethereum tx error: {:?}", e);  
            }  
        }  
        Err(e) => println!("❌ Invalid JSON received: {}", e),  
    }  
});  
}  
}  
}
```

Source Kode untuk index.html dari Web3 with feature

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Home Bakery 🍞 - Sensor Monitoring</title>
<script src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.umd.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script src="https://cdn.jsdelivr.net/npm/qrcode/build/qrcode.min.js"></script>
<link rel="stylesheet" href="style1.css" />
</head>
<body>
<h1>🍞 Home Bakery - Sensor Monitoring by Kelompok 11</h1>

<!-- Tab Navigation -->
<div class="nav-buttons">
<button onclick="showTab('historyTab')"> Riwayat Sensor</button>
```

```

<button onclick="showTab('inputTab')">  Loading In Bahan Baku</button>
</div>

<!-- Tab 1: Riwayat Sensor -->
<div id="historyTab">
  <h2>Riwayat Sensor Gudang</h2>
  <button onclick="loadSensorData()">  Muat Data Sensor</button>

  <table id="sensorTable">
    <thead>
      <tr>
        <th>Waktu</th>
        <th>ID Sensor</th>
        <th>Lokasi</th>
        <th>Tahapan</th>
        <th>Suhu (°C)</th>
        <th>Kelembapan (%)</th>
      </tr>
    </thead>
    <tbody></tbody>
  </table>

  <canvas id="chart" height="100"></canvas>
</div>

<!-- Tab 2: Konfirmasi Waktu Loading -->
<div id="inputTab" style="display: none;">
  <h2>  Barang Stok Masuk Gudang</h2>

  <label for="datetimeInput">Masukkan Tanggal & Waktu (misal: 6/18/2025, 11:07:25 PM):</label><br />
  <input type="text" id="datetimeInput" placeholder="MM/DD/YYYY, hh:mm:ss AM/PM" style="width: 100%; padding: 8px;" />
  <br /><br />
  <button onclick="confirmLoading()">Konfirmasi Loading In Bahan Baku</button>

  <div id="resultBox" style="margin-top: 20px;">
    <h3>  Data Sensor Ditemukan:</h3>
    <p id="resultData">Belum ada data.</p>
    <div id="qrcode"></div>
  </div>
</div>

<script src="script.js"></script>
</body>
</html>

```

Source kode untuk script.js dari Web3 with feature

```
const contractAddress = "0xf7d269cd81b82775a4646d8a2532c25028e9c38a"; // Ganti dengan alamat kontrak dari Ganache
const abiPath = "abi/SensorStorage.abi";

let chart;

async function loadSensorData() {
  const abiRes = await fetch(abiPath);
  const abi = await abiRes.json();

  const provider = new ethers.BrowserProvider(window.ethereum || "http://localhost:8545");
  await provider.send("eth_requestAccounts", []);
  const signer = await provider.getSigner();
  const contract = new ethers.Contract(contractAddress, abi, signer);

  const filter = contract.filters.DataStored();
  const events = await contract.queryFilter(filter, 0, "latest");

  const tableBody = document.querySelector("#sensorTable tbody");
  tableBody.innerHTML = "";

  const labels = [];
  const temps = [];
  const hums = [];

  events.forEach((e) => {
    const data = e.args;
    const timeStr = new Date(Number(data.timestamp) * 1000).toLocaleString();
    const temp = Number(data.temperature) / 100;
    const hum = Number(data.humidity) / 100;

    tableBody.innerHTML += `
      <tr>
        <td>${timeStr}</td>
        <td>${data.sensorId}</td>
        <td>${data.location}</td>
        <td>${data.stage}</td>
        <td>${temp.toFixed(2)}</td>
        <td>${hum.toFixed(2)}</td>
      </tr>
    `;

    labels.push(timeStr);
    temps.push(temp);
    hums.push(hum);
  });

  renderChart(labels, temps, hums);
}
```

```

}

function renderChart(labels, temps, hums) {
  const ctx = document.getElementById('chart').getContext('2d');

  if (chart) chart.destroy();

  chart = new Chart(ctx, {
    type: 'line',
    data: {
      labels,
      datasets: [
        {
          label: "Temperature (°C)",
          data: temps,
          borderColor: 'rgba(255, 99, 132, 1)',
          fill: false
        },
        {
          label: "Humidity (%)",
          data: hums,
          borderColor: 'rgba(54, 162, 235, 1)',
          fill: false
        }
      ]
    },
    options: {
      responsive: true,
      scales: {
        y: { beginAtZero: true }
      }
    }
  });
}

function showTab(tabId) {
  document.getElementById("historyTab").style.display = "none";
  document.getElementById("inputTab").style.display = "none";
  document.getElementById(tabId).style.display = "block";
}

async function confirmLoading() {
  const input = document.getElementById("datetimeInput").value.trim();
  if (!input) return alert("Harap masukkan tanggal dan waktu.");

  const targetTimestamp = Math.floor(new Date(input).getTime() / 1000);
  if (isNaN(targetTimestamp)) return alert("Format waktu tidak valid.");
}

// Gunakan BrowserProvider seperti sebelumnya
const abiRes = await fetch(abiPath);

```

```

const abi = await abiRes.json();

const provider = new ethers.BrowserProvider(window.ethereum || "http://localhost:8545");
await provider.send("eth_requestAccounts", []);
const signer = await provider.getSigner();
const contract = new ethers.Contract(contractAddress, abi, signer);

const filter = contract.filters.DataStored();
const events = await contract.queryFilter(filter, 0, "latest");

// Cari event dengan timestamp terdekat
let closestEvent = null;
let closestDiff = Infinity;

for (const e of events) {
  const ts = Number(e.args.timestamp);
  const diff = Math.abs(ts - targetTimestamp);
  if (diff < closestDiff) {
    closestDiff = diff;
    closestEvent = e;
  }
}

if (!closestEvent) {
  document.getElementById("resultData").textContent = "Data tidak ditemukan.";
  return;
}

const data = closestEvent.args;
const timeStr = new Date(Number(data.timestamp) * 1000).toLocaleString();
const temp = Number(data.temperature) / 100;
const hum = Number(data.humidity) / 100;

const message = `
<div style="text-align: center; font-size: 18px;">
Status Loading In telah Dilakukan:<br/><br/>
<strong>Waktu:</strong> ${timeStr}<br/>
<strong>Suhu:</strong> ${temp.toFixed(2)} °C<br/>
<strong>Kelembapan:</strong> ${hum.toFixed(2)} %<br/>
`;

// Tampilkan hasil
document.getElementById("resultData").innerHTML = message;

// Tampilkan gambar QR code statis
const qrCodeContainer = document.getElementById("qrcode");
qrCodeContainer.innerHTML =
<div style="text-align: center; font-size: 18px;">
<p><strong>Silahkan Konfirmasikan Loading In Barang pada Kontak  
berikut:</strong></p>

```

```

';
}
```

Source kode untuk style1.css dari Web3 with feature

```
body {
    background-color: #fff8f0;
    color: #5c4033;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    padding: 30px;
}

h1, h2 {
    color: #8b4513;
    margin-bottom: 10px;
}

h2 {
    font-weight: normal;
}

button {
    background-color: #d2691e;
    color: #fff;
    padding: 12px 24px;
    font-size: 16px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    margin-bottom: 20px;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #a0522d;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0,0,0,0.05);
    border-radius: 8px;
    overflow: hidden;
```

```

}

th, td {
    border: 1px solid #eee;
    padding: 12px;
    text-align: center;
}

th {
    background-color: #ffe5b4;
}

canvas {
    margin-top: 40px;
    background: #fff;
    border-radius: 10px;
    padding: 20px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.05);
}

```

Source kode untuk QTgui.py

```

import tkinter as tk
from tkinter import ttk
import requests
import threading
import time
import csv
from io import StringIO
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from collections import deque

# Konfigurasi InfluxDB
INFLUX_QUERY_URL = "http://localhost:8086/api/v2/query"
ORG = "ITS"
BUCKET = "ISImonitor"
TOKEN = "mTjO4KT4rlVLw3IISNiewi_JWxiErLd67bxRbx_oSrHmVz8-
eTjcqT_IxBEoTSJRud40uBsz9uaPC7aYMOrwZQ=="

# Riwayat data
history_length = 50
temp_history = deque(maxlen=history_length)
rh_history = deque(maxlen=history_length)
time_history = deque(maxlen=history_length)

```

```

def get_latest_data():
    flux_query = f"""
from(bucket: "{BUCKET}")
|> range(start: -1m)
|> filter(fn: (r) => r._measurement == "monitoring")
|> filter(fn: (r) => r._field == "temperature" or r._field == "humidity")
|> last()
"""

headers = {
    "Authorization": f"Token {TOKEN}",
    "Content-Type": "application/vnd.flux",
    "Accept": "application/csv"
}

try:
    response = requests.post(
        INFLUX_QUERY_URL,
        params={"org": ORG},
        headers=headers,
        data=flux_query
    )

    reader = csv.DictReader(StringIO(response.text))
    data = {}
    for row in reader:
        try:
            field = row["_field"]
            value = float(row["_value"])
            data[field] = value
        except:
            continue

    if "temperature" in data and "humidity" in data:
        return data["temperature"], data["humidity"]
    return None
except Exception as e:
    print("✖ Exception query Influx:", e)
    return None

def get_data_range(start_time, end_time):
    flux_query = f"""
from(bucket: "{BUCKET}")
|> range(start: {start_time}, stop: {end_time})
|> filter(fn: (r) => r._measurement == "monitoring")
|> filter(fn: (r) => r._field == "temperature" or r._field == "humidity")
"""

headers = {

```

```

    "Authorization": f"Token {TOKEN}",
    "Content-Type": "application/vnd.flux",
    "Accept": "application/csv"
}

try:
    response = requests.post(
        INFLUX_QUERY_URL,
        params={"org": ORG},
        headers=headers,
        data=flux_query
    )

    reader = csv.DictReader(StringIO(response.text))
    rows = []
    for row in reader:
        try:
            time_str = row["_time"]
            field = row["_field"]
            value = float(row["_value"])
            rows.append((time_str, field, value))
        except:
            continue

    return rows
except Exception as e:
    print("✖ Exception query Influx:", e)
    return []

def update_data():
    while True:
        result = get_latest_data()
        current_time = time.strftime("%H:%M:%S")

        if result:
            temp, rh = result
            label_temp.config(text=f"Suhu: {temp:.1f} °C")
            label_rh.config(text=f"Kelembaban: {rh:.1f} %")
            status_label.config(text="✓ Data dari Influx")

            temp_history.append(temp)
            rh_history.append(rh)
            time_history.append(current_time)

            plot_graph()
        else:
            label_temp.config(text="Suhu: ---")
            label_rh.config(text="Kelembaban: ---")

```

```

status_label.config(text="✖ Gagal ambil data")

time.sleep(2)

def plot_graph():
    ax1.clear()
    ax2.clear()

    fig.patch.set_facecolor('black')
    ax1.set_facecolor('black')
    ax2.set_facecolor('black')

    x = list(range(len(time_history)))
    times = list(time_history)

    ax1.plot(x, list(temp_history), label='Suhu (°C)', color='red', marker='o', linestyle='--')
    ax2.plot(x, list(rh_history), label='Kelembaban (%)', color='cyan', marker='x', linestyle='--')

    ax1.set_title("Grafik Suhu", color='white')
    ax2.set_title("Grafik Kelembaban", color='white')
    ax1.set_ylabel("°C", color='white')
    ax2.set_ylabel("%", color='white')

    interval = 5
    tick_positions = x[::interval]
    tick_labels = times[::interval]

    ax1.set_xticks(tick_positions)
    ax2.set_xticks(tick_positions)
    ax1.set_xticklabels(tick_labels, rotation=45, ha="right", color='white')
    ax2.set_xticklabels(tick_labels, rotation=45, ha="right", color='white')

    for ax in [ax1, ax2]:
        ax.tick_params(axis='y', colors='white')
        ax.tick_params(axis='x', colors='white')
        ax.grid(True, linestyle='--', alpha=0.3, color='gray')
        for spine in ax.spines.values():
            spine.set_color('white')

    fig.tight_layout()
    canvas.draw()

def auto_refresh_table():
    rows = get_data_range("-5m", "now()")
    for i in table.get_children():
        table.delete(i)

    if rows:
        grouped = {}

```

```

for t, f, v in rows:
    if t not in grouped:
        grouped[t] = {}
    grouped[t][f] = v

for t in sorted(grouped.keys())[-20:]:
    row = grouped[t]
    temp = row.get("temperature", "--")
    rh = row.get("humidity", "--")
    table.insert("", "end", values=(t[11:19], f"{temp:.1f}", f"{rh:.1f}"))
root.after(5000, auto_refresh_table)

# GUI Setup
root = tk.Tk()
root.title("Monitor SHT20 dari InfluxDB")
root.geometry("900x700")

notebook = ttk.Notebook(root)
tab_realtime = ttk.Frame(notebook)
tab_table = ttk.Frame(notebook)
notebook.add(tab_realtime, text=" Realtime")
notebook.add(tab_table, text=" Tabel Historis")
notebook.pack(fill="both", expand=True)

# Tab Realtime
label_temp = tk.Label(tab_realtime, text="Suhu: -- °C", font=("Helvetica", 16))
label_temp.pack(pady=5)
label_rh = tk.Label(tab_realtime, text="Kelembaban: -- %", font=("Helvetica", 16))
label_rh.pack(pady=5)
status_label = tk.Label(tab_realtime, text="Status: ---", fg="blue")
status_label.pack(pady=5)

fig = Figure(figsize=(7, 4.5), dpi=100)
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)
canvas = FigureCanvasTkAgg(fig, master=tab_realtime)
canvas.get_tk_widget().pack(pady=10)

# Tab Tabel Historis
cols = ("Waktu", "Suhu (°C)", "Kelembaban (%)")
table = ttk.Treeview(tab_table, columns=cols, show="headings", height=20)
for col in cols:
    table.heading(col, text=col)
    table.column(col, width=150, anchor="center")
table.pack(padx=10, pady=10, fill="both", expand=True)

# Start background threads
threading.Thread(target=update_data, daemon=True).start()
auto_refresh_table()
root.mainloop()

```

