RANCANG BANGUN SISTEM SMART HIDROPONIK UNTUK MONITORING DETEKSI HAMA PADA TUMBUHAN BERBASIS INTERNET OF THINGS

Sub Judul:

ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

SKRIPSI

Muhammad Khairu Mufid - 2207421031

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

DEPOK

2026

ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

SKRIPSI

Dibuat untuk Melengkapi Syarat-Syarat yang Diperlukan untuk Memperoleh Diploma Empat Politeknik

Muhammad Khairu Mufid - 2207421031

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

DEPOK

2026

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertanda tangan di bawah ini:

Nama : Muhammad Khairu Mufid

NIM : 2207421031

Jurusan/Program Studi : T.Informatika dan Komputer / T. Multimedia dan Jaringan

Judul Skripsi : ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

Menyatakan dengan sebenarnya bahwa skripsi ini benar-benar merupakan hasil karya saya sendiri, bebas dari peniruan terhadap karya dari orang lain. Kutipan pendapat dan tulisan orang lain ditunjuk sesuai dengan cara-cara penulisan karya ilmiah yang berlaku

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa dalam skripsi ini terkandung cirri-ciri plagiat dan bentuk-bentuk peniruan lain yang dianggap melanggar peraturan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Depok, 28 Juni 2026

Yang membuat pernyataan

(Muhammad Khairu Mufid)

NIM.2207421031

LEMBAR PENGESAHAN

Skripsi diajukan oleh:

Nama : Muhammad Khairu Mufid

NIM : 2207421031

Program Studi : Teknik Multimedia dan Jaringan

Judul Skripsi : ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

Telah diuji oleh tim penguji dalam Sidang Skripsi pada hari Rabu, Tanggal 1, Bulan Juli, Tahun 2026 dan dinyatakan **LULUS**

Disahkan oleh,

Pembimbing I : Maria Agustin, S.Kom., M.Kom. ( )

Penguji I : Dr. Prihatin Oktivasari S.Si.,M.Si ( )

Penguji II : Susana Dwi Yulianti M.Kom. ( )

Penguji III : Chandra Wirawan, S.Kom, M.Kom. ( )

Mengetahui :

Jurusan Teknik Informatika dan Komputer

Ketua

Dr. Anita Hidayati, S.Kom., M.Kom.

NIP.197908032003122003

KATA PENGANTAR

Puji dan syukur ke hadirat Allah SWT atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi berjudul "Analisis Perbandingan Kinerja Protokol HTTP, WebSocket, dan MQTT pada Platform Deteksi Hama Berbasis Arsitektur *Microservices*". Skripsi ini disusun sebagai syarat penyelesaian Program Studi Sarjana Terapan Teknik Multimedia dan Jaringan di Politeknik Negeri Jakarta. Penulis menyadari bahwa penyusunan skripsi ini tidak lepas dari bimbingan, bantuan, dan dukungan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT, atas segala kemudahan, kelancaran, dan petunjuk-Nya selama proses penelitian.
2. Ibu Maria Agustin, S.Kom., M.Kom., selaku dosen pembimbing yang telah meluangkan waktu serta memberikan arahan teknis yang berharga dari awal hingga akhir.
3. Kedua orang tua dan keluarga, yang senantiasa memberikan doa, cinta, dan dukungan moral tanpa henti.
4. Rekan-rekan tim penelitian hidroponik, atas kerja sama, kolaborasi, dan diskusi teknis yang sangat membantu penyelesaian riset ini.

Penulis menyadari bahwa skripsi ini masih memiliki keterbatasan. Oleh karena itu, kritik dan saran yang membangun diharapkan untuk penyempurnaan di masa mendatang. Semoga penelitian ini dapat memberikan manfaat nyata bagi pengembangan rekayasa jaringan dan *Internet of Things* (IoT).

Depok, 28 Juni 2026

Muhammad Khairu Mufid

SURAT PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Politeknik Negeri Jakarta, saya bertanda tangan di bawah ini:

Nama : Muhammad Khairu Mufid

NIM : 2207421031

Jurusan/ProgramStudi : T.Informatika dan Komputer / T. Multimedia dan Jaringan

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Politeknik Negeri Jakarta Hak Bebas Royalti Non-Eksklusif atas karya ilmiah saya yang berjudul :

ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksklusif ini Politeknik Negeri Jakarta Berhak menyimpan, mengalihmediakan/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan skripsisaya tanpa meminta izin darisaya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Depok, 28 Juni 2026

Yang Menyatakan

(Muhammad Khairu Mufid)

NIM.2207421031

ANALISIS PERBANDINGAN KINERJA PROTOKOL HTTP, WEBSOCKET, DAN MQTT PADA PLATFORM DETEKSI HAMA BERBASIS ARSITEKTUR MICROSERVICES

ABSTRAK

Penelitian ini dilatarbelakangi oleh beban ganda perangkat edge computing dalam mengeksekusi model kecerdasan buatan sekaligus mengirimkan gambar secara terus-menerus pada sistem pemantauan hidroponik. Kondisi ini rentan memicu hambatan pada perangkat keras. Di sisi lain, studi perbandingan sebelumnya umumnya masih terbatas pada pengujian data sensor berskala ringan, sehingga efisiensi protokol jaringan untuk pengiriman data gambar berukuran besar belum teruji secara pasti. Oleh karena itu, penelitian ini bertujuan mengevaluasi dan membandingkan kinerja protokol HTTP, WebSocket, dan MQTT pada arsitektur microservices. Metode yang digunakan adalah kuantitatif eksperimental melalui ekstraksi parameter Quality of Service (QoS) berstandar TIPHON dan pengujian keandalan operasional. Hasil pengujian menunjukkan Packet Loss seluruh protokol sangat ideal di bawah 1%. MQTT terbukti sebagai protokol paling optimal dengan Throughput tertinggi sebesar 153.92 Kbps dan Latency yang bersaing di angka 26.10 ms. Sebaliknya, HTTP menjadi yang paling lambat (27.71 ms) akibat pembengkakan lapisan aplikasi dan mekanisme sinkron, sedangkan WebSocket memicu fluktuasi Jitter tertinggi mencapai 78.22 ms akibat beban komputasi enkripsi XOR Masking pada tingkat mikrokontroler. Infrastruktur microservices terbukti kebal terhadap kegagalan berantai dengan waktu pemulihan murni berkisar antara 5.019 hingga 8.044 detik. Kesimpulannya, integrasi MQTT dan arsitektur microservices menghasilkan platform smart farming yang memiliki toleransi kesalahan tinggi dan andal untuk operasional greenhouse hidroponik.

Kata kunci: Arsitektur Microservices, Edge Computing, Hidroponik, Internet of Things, MQTT, Quality of Service.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah

Pertanian modern, khususnya sistem hidroponik, menuntut pengawasan untuk menjaga kualitas dan kuantitas hasil panen. Lingkungan tumbuh pada hidroponik sangat bergantung pada stabilitas parameter buatan. Pengawasan berkelanjutan menjadi penting karena sedikit saja fluktuasi pada kondisi lingkungan atau masuknya anomali biologis dapat langsung berdampak pada pertumbuhan sayuran dan memicu gagal panen (Kartika dkk., 2025).

Dari berbagai variabel risiko yang ada, salah satu ancaman terbesar dalam budidaya hidroponik adalah serangan hama. Hama seperti lalat buah, kutu daun, hingga ulat grayak sering kali memanfaatkan lingkungan *greenhouse* hidroponik sebagai sumber makanan dan lokasi berkembang biak. Berdasarkan penelitian yang dilakukan oleh Zulgani dkk. (2023), intensitas kerusakan tanaman sayur akibat hama yang tidak tertangani dapat menyebar dengan sangat cepat dan mencapai tingkat kerusakan signifikan hingga 60,7% dari total populasi tanaman.

Urgensi penanganan hama tersebut terkonfirmasi melalui hasil observasi dan wawancara langsung dengan pihak pengelola *greenhouse* hidroponik Lokatani. Pada operasionalnya, *greenhouse* hidroponik masih menghadapi serangan hama secara terus-menerus, khususnya kutu kebul (*whitefly*) dan kutu daun (*aphids*). Serangan hama tersebut secara spesifik menyerang dan merusak komoditas utama kebun, yakni tanaman pakcoy dan bayam. Kemunculan hama yang terus berulang di *greenhouse* hidroponik Lokatani membuktikan bahwa ancaman ini menuntut adanya integrasi teknologi deteksi hama.

Dalam menghadapi serangan hama di *greenhouse* hidroponik seperti Lokatani, metode pemantauan konvensional yang masih mengandalkan pengawasan secara manual oleh pekerja dinilai sangat tidak efisien dan rentan terhadap keterlambatan penanganan (Iman, 2026). Keterbatasan penglihatan mata manusia membuat gejala awal serangan hama sering kali terlewatkan. Oleh karena itu, integrasi teknologi

Internet of Things (IoT) dan kecerdasan buatan (*Computer Vision*) menjadi solusi krusial. Seperti yang dijelaskan oleh Tiwari dkk. (2020), penerapan IoT dan otomatisasi pengambilan keputusan mampu mengurangi ketergantungan pada pemantauan manual, sekaligus menggantikannya dengan sistem peringatan dini yang beroperasi secara *real-time*. Pada implementasinya, sistem pendeteksi hama ini membutuhkan arsitektur protokol komunikasi yang andal agar dapat mengirimkan data tanpa hambatan.

Integrasi teknologi IoT dan pengolahan citra pada sistem hidroponik telah diteliti secara spesifik melalui *prototype* deteksi hama YOLO berbasis Raspberry Pi (Dinata, 2023). Penelitian tersebut mengeksekusi model kecerdasan buatan secara lokal pada *Edge Device*, yang terbukti memicu *hardware bottleneck* dengan penurunan *frame rate* hingga mencapai 1,14 FPS. Namun, arsitektur pada penelitian tersebut hanya mengelola data di dalam *server* lokal tanpa adanya beban transmisi internet jarak jauh. Ketika infrastruktur ini diekspansi untuk mendistribusikan data gambar menuju *server Cloud Microservices* melalui jaringan publik, perangkat *Edge* dipaksa memikul *dual workload*, yakni mempertahankan komputasi visual dan memproses enkapsulasi jaringan secara serentak. Kondisi ini memunculkan tantangan arsitektural baru, yakni urgensi pemilihan protokol komunikasi yang paling ringan dan memiliki *Quality of Service* (QoS) tinggi, agar beban jaringan tidak memperparah kemacetan pemrosesan yang dapat melumpuhkan sistem.

Sebagai solusi latensi pada jaringan IoT, penerapan protokol komunikasi *persistent* seperti WebSocket terbukti mampu menekan waktu respons transmisi hingga di bawah 1 detik (Kojansow dkk., 2024). Di sisi lain, Tsaqief dan Sutopo (2025) membuktikan bahwa MQTT (QoS 0) secara fundamental jauh lebih unggul dalam meminimalkan latensi dan Jitter dibandingkan koneksi WebSocket, yang rentan mengalami hambatan pemrosesan di tingkat mikrokontroler. Analisis komparatif secara langsung juga telah dilakukan oleh Amirkhanov dkk. (2025), yang mengonfirmasi bahwa MQTT mencetak latensi rata-rata tercepat (290,5 ms) dibandingkan WebSocket (341,9 ms) maupun HTTP. Akan tetapi, seluruh penelitian tersebut memiliki satu celah empiris yang fundamental, yaitu pengujian hanya memikul muatan telemetri lingkungan berskala sangat ringan. Belum ada

pembuktian mengenai batas kegagalan protokol dan lonjakan latensi saat dihadapkan pada muatan data gambar berukuran besar secara dinamis, yang menjadi titik fokus komparatif pada penelitian ini.

Berdasarkan celah pada penelitian terdahulu, kebutuhan distribusi data gambar ke *server cloud* memunculkan dilema arsitektural pada pemilihan protokol komunikasi di perangkat *edge*. Setiap protokol lapisan aplikasi, yakni HTTP, WebSocket, dan MQTT, memiliki karakteristik bawaan yang berisiko menciptakan *bottleneck*. Mengingat infrastruktur jaringan modern seperti Wi-Fi 5G telah meniadakan batasan kecepatan fisik saluran, ancaman hambatan transmisi kini murni bergeser pada inefisiensi lapisan aplikasi dari protokol itu sendiri serta penalti pemrosesan pada perangkat keras berspesifikasi terbatas. Oleh karena itu, penelitian ini bertujuan memecahkan dilema tersebut dengan mengevaluasi dan membandingkan kinerja ketiga protokol secara empiris pada skenario transmisi gambar yang dikirim secara kontinu mengikuti pergerakan aktuator kamera. Evaluasi difokuskan pada ekstraksi parameter *Quality of Service* (QoS) berdasarkan standar TIPHON, meliputi *Latency*, *Throughput*, *Packet Loss*, dan *Jitter*, untuk merumuskan standar arsitektur telemetri visual yang paling efisien, stabil, dan optimal.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, rumusan masalah dalam penelitian ini adalah:

1. Bagaimana merancang bangun dan memvalidasi fungsionalitas sistem pemantauan deteksi hama berbasis arsitektur *microservices* menggunakan metode pengujian *Blackbox* serta *User Acceptance Test* (UAT)?
2. Bagaimana mengimplementasikan protokol lapisan aplikasi (HTTP, WebSocket, dan MQTT) di dalam lingkungan *microservices* terisolasi untuk melayani transmisi data citra dari perangkat *Edge* menuju *server*?
3. Bagaimana tingkat keandalan sistem saat dihadapkan pada skenario anomali jaringan dan arsitektural, seperti pemutusan koneksi, beban pengiriman tinggi, hingga gangguan layanan *broker*?

4. Bagaimana perbandingan kinerja jaringan berdasarkan parameter QoS meliputi *Throughput*, *Packet Loss*, *End-to-End Latency*, dan *Jitter* antara protokol HTTP, WebSocket, dan MQTT pada transmisi data gambar?
5. Bagaimana relevansi empiris antara degradasi metrik QoS yang terekam dengan batasan arsitektural dari setiap protokol, khususnya terkait inefisiensi pembengkakan lapisan aplikasi (*Application-Layer Bloat*) dan mekanisme sinkron (*Half-Duplex*) pada HTTP, serta penalti komputasi XOR Masking pada WebSocket?

1.3. Batasan Masalah

Agar penelitian lebih terarah dan parameter pengujian tetap objektif, ruang lingkup dibatasi pada:

1. Perancangan dan pengujian dibatasi pada evaluasi platform arsitektur *microservices* (frontend, backend, dan komunikasi data) menggunakan metode Blackbox dan *User Acceptance Test* (UAT). Pengembangan model kecerdasan buatan (YOLO) dan perakitan *hardware* aktuator dikecualikan karena dikerjakan terpisah dalam riset payung yang sama.
2. Implementasi transmisi data dibatasi pada karakteristik bawaan protokol HTTP, WebSocket, dan MQTT (QoS Level 0). Protokol alternatif seperti CoAP, AMQP, atau gRPC tidak diuji. Keamanan tingkat lanjut seperti enkripsi TLS/SSL tidak diterapkan agar pengujian murni menilai beban komputasi bawaan dari masing-masing protokol.
3. Pengujian *reliability* sistem dibatasi pada simulasi operasi selama 6 jam untuk mengukur stabilitas memori, kemampuan *auto-reconnect* perangkat edge saat jaringan internet terputus, serta pemulihan *recovery* otomatis saat dependensi *microservices* seperti database dan broker dimatikan secara paksa.
4. Komparasi kinerja jaringan dieksekusi menggunakan infrastruktur Wi-Fi 5G untuk memisahkan parameter Throughput, Packet Loss, End-to-End Latency, dan Jitter dari gangguan sinyal seluler. Pengiriman data menggunakan gambar 640x640 piksel berformat Base64 dengan interval pengiriman tetap 3 detik. Interval 3 detik ini ditetapkan berdasarkan batas waktu yang dibutuhkan

perangkat edge untuk mengeksekusi model AI, konversi *payload*, dan waktu pergerakan motor slider kamera secara kontinu dari satu ujung ke ujung lainnya.

5. Analisis penurunan kualitas QoS dibatasi pada evaluasi inefisiensi akibat header yang membengkak (*Application-Layer Bloat*) dan mekanisme transmisi sinkron pada HTTP, serta beban komputasi enkripsi XOR *Masking* bawaan WebSocket pada mikrokontroler.

1.4. Tujuan dan Manfaat

Berdasarkan rumusan masalah yang telah ditetapkan, penelitian ini memiliki tujuan spesifik sebagai berikut:

1. Merancang bangun dan memvalidasi fungsionalitas sistem pemantauan deteksi hama berbasis arsitektur *microservices* melalui metode pengujian *Blackbox* serta *User Acceptance Test* (UAT).
2. Mengimplementasikan protokol lapisan aplikasi (HTTP, WebSocket, dan MQTT) di dalam lingkungan *microservices* terisolasi untuk melayani transmisi data citra secara *persistent*.
3. Mengevaluasi tingkat keandalan operasional sistem saat dihadapkan pada skenario anomali melalui uji ketahanan durasi operasi, pengujian *Recovery* koneksi jaringan otomatis (*auto-reconnect*), dan resiliensi interupsi layanan *backend*.
4. Mengekstraksi dan membandingkan kinerja jaringan berdasarkan parameter QoS—meliputi *Throughput*, *Packet Loss*, *End-to-End Latency*, dan *Jitter*—dari ketiga protokol tersebut di atas batas kapasitas infrastruktur nirkabel.
5. Menganalisis relevansi empiris antara degradasi metrik QoS yang terekam dengan batasan arsitektural bawaan protokol, untuk mengetahui dampak inefisiensi dari pembengkakan lapisan aplikasi (*Application-Layer Bloat*) dan mekanisme sinkron (*Half-Duplex*) pada HTTP, serta penalti komputasi XOR Masking pada WebSocket.

Adapun manfaat yang diharapkan dari hasil penelitian ini adalah:

1. Menghasilkan sistem pemantauan deteksi hama yang fungsional dan telah teruji penggunaannya, sehingga dapat langsung membantu pengelola *greenhouse* hidroponik dalam mengambil tindakan penanganan hama.
2. Menyediakan rancangan arsitektur perangkat lunak berbasis *microservices* yang stabil, sehingga dapat dijadikan panduan praktis bagi *developer* lain yang ingin membangun aplikasi *Smart Farming* untuk kebutuhan pengiriman data gambar secara terus-menerus.
3. Memberikan sistem yang *reliable* dan mampu memulihkan koneksinya sendiri secara otomatis ketika terjadi pemutusan jaringan internet atau gangguan mendadak pada *server*.
4. Menyediakan tolok ukur berupa data nyata yang memperlihatkan perbandingan kecepatan dan kestabilan kinerja antara protokol HTTP, WebSocket, dan MQTT.
5. Memberikan wawasan teknis baru yang mengetahui penyebab pasti mengapa sebuah protokol bisa menjadi lambat atau membebani perangkat keras, sehingga dapat mencegah *developer* sistem IoT dari kesalahan dalam memilih protokol komunikasi.

1.5. Sistematika Penulisan

1. BAB I PENDAHULUAN

   Bab ini menguraikan fondasi penelitian, dimulai dari latar belakang masalah terkait *bottleneck* performa transmisi HTTP pada sistem deteksi hama sebelumnya. Selain itu, bab ini memaparkan rumusan masalah yang berfokus pada komparasi arsitektur jaringan, penetapan batasan masalah untuk mengunci variabel pengujian (*payload* citra dan isolasi protokol), serta menjabarkan tujuan, manfaat penelitian, dan sistematika penulisan laporan.
2. BAB II TINJAUAN PUSTAKA

   Bab ini berisi kajian literatur dan landasan teori yang mendasari rekayasa sistem. Topik utama mencakup arsitektur protokol komunikasi dasar (HTTP *Request-Response*, WebSocket *Full-Duplex*, dan MQTT Pub/Sub), teori pengukuran parameter *Quality of Service* (QoS) pada jaringan, karakteristik

format encoding data citra (Base64), serta tinjauan pustaka dari penelitian terdahulu.

3. BAB III METODE PENELITIAN

   Bab ini menjabarkan *blueprint* dan metodologi yang digunakan untuk mengeksekusi penelitian. Pembahasan meliputi perancangan topologi jaringan yang memuat node IoT, MQTT Broker, *Server Backend*, dan *Client Frontend*. Lebih lanjut, bab ini merinci skenario pengujian secara terisolasi untuk ketiga protokol, standarisasi variabel kontrol (seperti resolusi dan ukuran *payload* citra), serta instrumen dan tata cara pengambilan data metrik QoS (*Latency, Throughput, Jitter*, dan *Packet Loss*).

4. BAB IV HASIL DAN PEMBAHASAN

   Bab ini menjelaskan hasil implementasi dan analisis komparatif dari sistem yang telah dibangun. Fokus utama bab ini adalah penyajian data empiris hasil testing transmisi citra dari ketiga protokol secara *apple-to-apple*, analisis teknis terhadap *overhead* atau anomali jaringan yang terjadi, serta implementasi dashboard web yang terintegrasi dengan fitur sistem pendukung keputusan (rekomendasi otomatis tindakan preventif dan kuratif hama).

5. BAB V PENUTUP

   Bab ini merangkum seluruh hasil penelitian ke dalam kesimpulan yang bersifat objektif dan terukur, secara spesifik menjawab protokol mana yang terbukti paling optimal secara empiris berdasarkan pengujian di Bab IV. Selain itu, bab ini memberikan saran atau rekomendasi teknis bagi pengembangan arsitektur sistem pada penelitian di masa mendatang.

BAB II

TINJAUAN PUSTAKA

2.1. Penelitian Terdahulu

Tinjauan terhadap penelitian terdahulu dilakukan untuk memetakan posisi penelitian dan mengidentifikasi *gap* yang akan diselesaikan. Berikut adalah beberapa sumber referensi penelitian terdahulu:

Tabel 2.1. Penelitian Terdahulu

No	Peneliti & Tahun	Judul Penelitian	Hasil & Temuan	Keterkaitan & Perbedaan (*Gap*)
1.	Dinata, D. C. (2023)	Rancang Bangun Sistem Penyiraman Pestisida dan Deteksi Hama Menggunakan Metode YOLO	Mengeksekusi model deteksi hama secara lokal pada perangkat *Edge* (Raspberry Pi). Eksekusi lokal ini memicu *hardware bottleneck*, membebani memori, dan menjatuhkan *frame rate* komputasi hingga 1,14 FPS.	Penelitian ini menggunakan komputasi *Edge AI* lokal yang serupa. Keterbaruan penelitian penulis terletak pada pengembangan arsitektur telemetri visual, sistem mendistribusikan data citra pasca-inferensi dari perangkat *Edge* menuju *server Cloud Microservices*, serta mengevaluasi efisiensi protokol lapisan aplikasi (HTTP, WebSocket, MQTT) untuk mengatasi risiko hambatan perangkat keras (*Hardware Bottleneck*) akibat beban ganda pada mikrokontroler.

2	Kojanso w, C. W., et al. (2024)	*Internet Of Things (IoT) Platform Development using WebSocket communication*	Membuktikan bahwa penggunaan arsitektur komunikasi *persistent* (WebSocket) mampu menangani komunikasi data telemetri *real-time* dan menekan waktu respons hingga di bawah 1 detik.	Penelitian ini membuktikan efektivitas WebSocket untuk data sensor ringan. Penelitian penulis mengujinya pada batas ekstrem, yakni transmisi muatan citra (Base64) berukuran besar di mana enkripsi aplikasi WebSocket rentan memicu hambatan komputasi.
3	Rofians yah, W., et al. (2025)	*IoT-Based Control and Monitoring System for Hydroponic Plant Growth Using Image Processing*	Sistem berhasil memantau kondisi hidroponik secara visual via aplikasi seluler, namun masih mengalami delay transmisi data 5-7 detik akibat limitasi performa perangkat keras dan metode komunikasi.	Riset ini relevan pada agrikultur presisi, namun tidak mengeksplorasi optimasi protokol jaringan. Penelitian penulis mengisi celah tersebut dengan menguji dan membandingkan protokol MQTT, WebSocket, dan HTTP secara spesifik untuk mencari solusi transmisi citra teringan dan tercepat.
4	Tsaqief, M. F., & Sutopo, J. (2025)	*Comparative Performance Analysis Between the MQTT and WebSocket Protocols*	Mengeksekusi pengujian di lingkungan perangkat berspesifikasi terbatas. Hasil membuktikan MQTT memiliki latensi (11.040 ms) dan Jitter (0.201 ms) yang	Riset ini secara empiris mengungkap kelemahan arsitektur WebSocket pada perangkat IoT. Namun, pengujian batas *payload* terbesar hanya dilakukan pada skala 250 bita. Penelitian penulis mengeksekusi pengujian

			sangat rendah. Sebaliknya, koneksi full-duplex WebSocket memicu hambatan komputasi (bottleneck) di tingkat mikrokontroler yang mendegradasi performa seiring bertambahnya ukuran data.	pada skala data gambar Base64 berukuran puluhan kilobita (ratusan kali lipat lebih besar), untuk mengetahui dampak empiris penalti komputasi kriptografi (*seperti XOR Masking*).
5	Amirkhanov, B., et.al. (2025)	*Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on Latency, stability, and integration*	Analisis empiris membuktikan MQTT menghasilkan rata-rata latensi terendah (290,5 ms) dan paling andal, sementara HTTP sangat tidak efisien dan rentan dengan latensi tinggi (342,6 ms).	Penelitian ini mengonfirmasi kelemahan fundamental HTTP dibanding MQTT dan WebSocket. Namun, pengujian mereka hanya menggunakan data sensor berskala *byte*. Penelitian penulis mengadopsi struktur komparasi ini dan menerapkannya pada transmisi data citra.

Penelitian ini difokuskan untuk mengisi celah pada efisiensi transmisi *payload* citra dalam ekosistem perangkat berspesifikasi terbatas. Penelitian terdahulu oleh Dinata (2023) mengonfirmasi bahwa eksekusi model kecerdasan buatan secara lokal pada perangkat keras (*Edge AI*) memicu *hardware bottleneck*, menjatuhkan *frame rate* hingga 1,14 FPS. Di sisi lain, Rofiansyah et al. (2025) mendemonstrasikan pemrosesan citra pada domain agrikultur, namun arsitekturnya masih terkendala oleh *delay* transmisi jaringan hingga 5-7 detik. Berangkat dari permasalahan

tersebut, penelitian ini merancang ekspansi arsitektur telemetri visual. Tantangan krusialnya terletak pada keharusan perangkat *Edge* memikul beban ganda (*dual workload*), yakni mempertahankan komputasi inferensi kecerdasan buatan sekaligus mengirim data gambar secara kontinu menuju *server Cloud Microservices* melalui jaringan seluler publik.

Sebagai solusi, literatur terdahulu telah menawarkan optimasi protokol pada lingkungan IoT. Tsaqief dan Sutopo (2025) mengeksekusi komparasi empiris, membuktikan bahwa MQTT (QoS 0) secara fundamental jauh lebih unggul dalam meminimalkan latensi dan Jitter dibandingkan koneksi WebSocket, yang rentan mengalami hambatan pemrosesan di tingkat mikrokontroler. Di sisi lain, Kojansow et al. (2024) menawarkan soket *persistent* WebSocket untuk menekan latensi di bawah 1 detik. Analisis komparatif yang lebih tajam divalidasi oleh Amirkhanov et al. (2025), yang menemukan bahwa arsitektur HTTP memicu lonjakan latensi yang jauh lebih buruk dibandingkan soket *persistent* milik MQTT maupun WebSocket. Akan tetapi, seluruh pengujian pada penelitian tersebut memiliki satu celah empiris yang serupa, yaitu muatan yang ditransmisikan hanyalah data telemetri ringan berukuran *byte*.

Dengan mengintegrasikan temuan-temuan tersebut, penelitian ini mengambil posisi kebaruan dengan mengeksekusi analisis komparatif secara empiris antara HTTP, WebSocket, dan MQTT yang difokuskan pada transmisi data gambar visual (Base64). Evaluasi parameter *Quality of Service* (QoS) dilakukan untuk mengetahui secara spesifik akar masalah degradasi telemetri pemantauan, apakah murni bersumber dari inefisiensi *Application-Layer Bloat* dan mekanisme sinkron (*Half-Duplex*) pada HTTP, atau terkait penalti komputasi keamanan tingkat aplikasi seperti kewajiban enkripsi *XOR Masking* pada WebSocket. Melalui identifikasi komprehensif ini, dapat dirumuskan standar arsitektur telemetri yang paling efisien dan andal untuk menunjang keberlangsungan sistem otomasi hidroponik

2.2.Internet of Things (IoT) dan Edge Computing

Internet of Things (IoT) secara fundamental merupakan ekosistem perangkat fisik yang saling terhubung untuk mengumpulkan dan bertukar data melalui jaringan. Namun, seiring dengan banyaknya volume data yang dihasilkan, arsitektur IoT konvensional yang memusatkan seluruh pemrosesan di *server Cloud* menghadapi kendala berupa batas kapasitas *bandwidth* dan waktu respons. Sebagai solusi dari arsitektur tersebut, paradigma *Edge Computing* diperkenalkan untuk menggeser pusat komputasi, analitik, dan penyimpanan data mendekati lokasi sumber data (*edge device*) berada (Satyanarayanan, 2017).

Dalam implementasi sistem cerdas, perangkat *Edge* (seperti Raspberry Pi) difungsikan untuk mengeksekusi model kecerdasan buatan secara lokal (*Edge AI Inference*). Akan tetapi, arsitektur Edge AI memiliki limitasi komputasi yang ketat. Berbeda dengan *server Cloud*, perangkat Edge merupakan entitas berspesifikasi terbatas (*resource-constrained device*) yang umumnya hanya memiliki 1 hingga 4 inti prosesor (CPU) dan kapasitas RAM yang minimal (Merenda dkk., 2020).

Keterbatasan ini menjadi krusial ketika perangkat dipaksa memproses Beban Ganda, yakni mengeksekusi komputasi data gambar secara bersamaan dengan memproses jaringan berfrekuensi tinggi. Benturan komputasi antara proses I/O jaringan dan analitik lokal ini memicu saturasi perangkat keras (*Hardware Bottleneck*). Pada titik saturasi, perangkat akan mengalami penalti performa ekstrem yang bermanifestasi pada *thread blocking*, penurunan laju bingkai (FPS), pelebaran waktu tunda transmisi, dan lonjakan Jitter pada jaringan (Gao dkk., 2026).

2.3.Konsep *Smart Farming* dan Hidroponik

Smart farming mengintegrasikan teknologi seperti IoT untuk mengoptimalkan proses pertanian melalui pemantauan *real-time* dan otomatisasi, mengurangi intervensi manusia serta meningkatkan hasil panen. Hidroponik merupakan metode budidaya tanaman tanpa tanah menggunakan larutan nutrisi, yang sangat cocok untuk lingkungan terkendali di daerah perkotaan dengan keterbatasan lahan. Integrasi IoT pada sistem hidroponik memungkinkan pengukuran parameter seperti

pH, EC, suhu, dan cahaya secara otomatis, sehingga meningkatkan pertumbuhan tanaman hingga 1,775% pada tanaman sawi (Nugroho et al., 2023). Sistem ini mendukung pertanian berkelanjutan dengan penghematan air dan pengendalian presisi melalui aplikasi *mobile* (HydroFarm Project, 2025).

2.4. Protokol Komunikasi Data

Bagian ini menjelaskan mengenai dasar teori dan arsitektur setiap protocol komunikasi data yang dianalisa dalam penelitian ini, meliputi *Hypertext Transfer Protocol* (HTTP), WebSocket *Protocol,* dan MQTT *Protocol.*

2.4.1. *Hypertext Transfer Protocol* (HTTP)

Hypertext Transfer Protocol (HTTP) adalah protokol lapisan aplikasi standar berbasis teks yang beroperasi pada model komunikasi *client-server*. Secara arsitektural, HTTP menggunakan mekanisme perpesanan sinkron (*Half-Duplex*), di mana setiap siklus transmisi (seperti pengiriman *payload* melalui metode POST) mewajibkan *Client* untuk menunggu respons utuh dari *server* (misalnya kode status 200 OK) sebelum dapat memproses atau mengirimkan antrean paket berikutnya (Kurose & Ross, 2021).

Meskipun secara fundamental berkarakteristik *stateless* (tidak menyimpan informasi status dari interaksi sebelumnya), standar HTTP/1.1 telah mengimplementasikan mekanisme koneksi *persistent* atau *TCP Keep-Alive* secara bawaan. Fitur ini mengizinkan *Client* dan *server* untuk mempertahankan satu soket TCP agar tetap terbuka selama periode waktu tertentu (Kurose & Ross, 2021). Dengan mekanisme ini, kewajiban untuk melakukan *TCP Three-way Handshake* (SYN, SYN-ACK, ACK) pada setiap siklus pengiriman berhasil dieliminasi, sehingga efisiensi waktu tempuh pada lapisan transport dapat dioptimalkan.

Namun, tantangan utama arsitektur HTTP pada ekosistem transmisi matriks *Internet of Things* (IoT) terletak pada inefisiensi lapisan aplikasinya (*Application-Layer Bloat*). Sesuai dengan spesifikasi protokol standar, setiap bingkai transmisi HTTP wajib menyertakan tumpukan *header* berbasis teks

mentah (seperti Host, User-Agent, dan Content-Type) yang diulang-ulang pada setiap permintaan (Grigorik, 2013). Pada skenario transmisi frekuensi tinggi berukuran besar, tumpukan *header* teks ini memicu pembengkakan muatan yang mengonsumsi *bandwidth* secara *redundant*. Selain itu, mesin pemroses dituntut untuk membuang siklus *Central Processing Unit* (CPU) secara konstan untuk melakukan *parsing* karakter *string*, yang secara akumulatif menciptakan hambatan penyandian (*Application-Layer Bottleneck*) pada perangkat keras.

2.4.2. WebSocket *Protocol*

WebSocket Protocol dirancang untuk mengatasi keterbatasan *overhead* inisiasi koneksi pada HTTP. Protokol ini menyediakan jalur komunikasi dua arah (*full-duplex*) di atas satu soket TCP yang *persistent*. Setelah fase HTTP Upgrade handshake awal berhasil disepakati, koneksi akan tetap terbuka. Secara teoritis, *persistent*si ini menghilangkan beban waktu dari TCP Handshake berulang, sehingga ideal untuk memfasilitasi aliran data antar-sistem dengan tingkat latensi transmisi yang lebih rendah (Kojansow et al., 2024).

Meskipun unggul dalam efisiensi jaringan, WebSocket memiliki regulasi keamanan bawaan yang berdampak langsung pada beban komputasi perangkat keras. Berdasarkan spesifikasi standar RFC 6455, WebSocket mewajibkan penerapan mekanisme penyandian berupa XOR *Masking* untuk setiap *data frame* yang bergerak searah dari *Client* menuju server (*client-to-server*). Dalam ekosistem *Edge Computing* dengan unit pemroses (CPU) berspesifikasi terbatas, pemaksaan komputasi terhadap data gambar berukuran besar secara konstan rentan memicu antrean pemrosesan lokal (*hardware bottleneck*). Penalti komputasi di tingkat aplikasi ini berisiko menganulir keunggulan kecepatan yang ditawarkan oleh koneksi TCP *persistent*nya

2.4.3. MQTT Protocol

Message Queuing Telemetry Transport (MQTT) adalah protokol konektivitas mesin-ke-mesin (M2M) yang dirancang secara spesifik untuk ekosistem

perangkat IoT dengan keterbatasan sumber daya daya dan bandwidth. MQTT tidak menggunakan model *client-server* konvensional, melainkan mengimplementasikan arsitektur *Publish-Subscribe* yang dikelola oleh sebuah broker terpusat.

Layaknya WebSocket, MQTT beroperasi di atas soket TCP yang *persistent* sehingga membebaskan perangkat dari *overhead* TCP *Handshake* berulang saat mengirimkan data secara interval. Namun, berbeda dengan WebSocket, MQTT bertindak murni sebagai protokol pengangkut (*transport*) yang sangat ringan dan tidak mewajibkan mekanisme penyandian masking terhadap struktur isi muatannya di tingkat aplikasi (Amirkhanov dkk., 2025). Efisiensi ganda ini—ketiadaan *overhead* inisiasi jaringan dan ketiadaan penalti komputasi kriptografi—menjadikan MQTT diandalkan untuk transmisi telemetri berkecepatan tinggi, meskipun kapabilitasnya dalam memikul muatan citra visual berukuran besar secara stabil menuntut evaluasi kinerja jaringan yang lebih spesifik.

2.4.4. Overhead Komputasi dan Kriptografi

Kinerja transmisi data pada perangkat Internet of Things (IoT) berspesifikasi terbatas tidak hanya bergantung pada kapasitas saluran fisik (*bandwidth*), melainkan juga dipengaruhi secara signifikan oleh beban pemrosesan (*computational overhead*) di tingkat aplikasi. Salah satu mekanisme yang memberikan penalti komputasi tertinggi pada protokol berbasis koneksi *persistent* adalah XOR Masking pada WebSocket.

Berdasarkan spesifikasi protokol standar, XOR Masking adalah regulasi keamanan yang diwajibkan secara mutlak bagi seluruh data frame yang dikirimkan dari sisi *Client* menuju *server* (*client-to-server*). Mekanisme ini dirancang secara bawaan oleh RFC 6455 untuk mencegah serangan *cache poisoning attack* yang dapat mengeksploitasi infrastruktur *proxy* di jaringan perantara.

Secara matematis, sebelum data dapat dilepas ke antarmuka jaringan fisik, unit pemroses (CPU) pada perangkat *Client* harus menghasilkan kunci masking 32-bit acak. Kunci ini kemudian diaplikasikan ke seluruh susunan *byte* pada muatan data (*payload*) menggunakan operasi logika gerbang *Exclusive*-OR (XOR). Jika *payload* yang ditransmisikan berukuran besar, perangkat *Edge* dipaksa untuk mengeksekusi puluhan ribu operasi kriptografi secara sekuensial (Liu & Zhao, 2022). Beban penyandian tingkat aplikasi inilah yang mendegradasi performa WebSocket di ekosistem IoT, menciptakan hambatan (*bottleneck*) internal yang dapat memicu lonjakan latensi waktu tempuh (*End-to-End Latency*) secara drastis, terlepas dari kualitas sinyal jaringan.

2.5.Representasi Data Gambar (Base64 Encoding)

Standar protokol komunikasi pada lapisan aplikasi (seperti HTTP dan WebSocket murni) secara fundamental dirancang untuk mentransmisikan data berbasis teks, sehingga tidak dapat secara langsung membaca muatan biner mentah (seperti *file* citra .jpg atau .png). Untuk mengatasi limitasi arsitektural ini, algoritma penyandian Base64 (*Base64 Encoding*) digunakan untuk mengonversi data biner menjadi representasi teks ASCII (*American Standard Code for Information Interchange*).

Mekanisme algoritma Base64 beroperasi dengan memecah aliran bit biner menjadi blok-blok berukuran 6-bit. Setiap blok 6-bit tersebut kemudian dipetakan secara eksklusif ke dalam sebuah tabel alfabet cetak yang berisi 64 karakter aman (Josefsson, 2006). Konsekuensi arsitektural dari proses ini bersifat mutlak secara matematis, karena setiap 3 bita (*byte*) data asli (setara 24 bit) dipaksa untuk dikonversi menjadi 4 karakter ASCII (4 bita), penyandian Base64 akan selalu menghasilkan pembengkakan ukuran berkas (*size overhead*) sebesar 33,3% dari ukuran aslinya (Lemire, 2019).

Dalam skenario telemetri sistem yang mengirimkan data gambar beresolusi tinggi, pembengkakan ukuran ini menciptakan dua kerugian operasional. Pertama, beban kapasitas *bandwidth* yang harus diangkut oleh saluran seluler meningkat secara signifikan. Kedua, mesin pada lapisan *server* maupun perangkat lunak dituntut untuk mendedikasikan siklus *Central Processing Unit* (CPU) secara konstan untuk

melakukan *parsing* rentetan karakter teks (*string*) sebelum dapat merakitnya kembali (dekode) menjadi gambar biner (Grigorik, 2013). Akumulasi waktu proses pengodean dan pembacaan struktur teks inilah yang menyumbang hambatan waktu tempuh pada lapisan aplikasi (*Application-Layer Bottleneck*).

2.6.Arsitektur dan Infrastruktur Sistem

2.6.1. Arsitektur Microservices dan Toleransi Kesalahan

Arsitektur microservices adalah pendekatan rekayasa perangkat lunak yang menstrukturkan aplikasi sebagai kumpulan layanan yang terisolasi, beroperasi secara independen, dan berkomunikasi melalui antarmuka jaringan yang ringan seperti *Application Programming Interface* (API) atau *message broker* (Newman, 2015). Paradigma terdistribusi ini secara mendasar menggantikan arsitektur monolitik konvensional yang rentan terhadap kematian system .

Keunggulan arsitektural *microservices* pada ekosistem *Internet of Things* (IoT) terletak pada kapabilitas Toleransi Kesalahan (*Fault Tolerance*). Dengan memisahkan setiap komponen infrastruktur ke dalam kontainer yang independen (seperti layanan antarmuka *server*, pangkalan data, dan broker pesan), arsitektur ini menciptakan isolasi kegagalan (*Failure Isolation*). Ketika satu layanan dependensi mengalami *crash* akibat anomali internal maupun interupsi jaringan, sistem dirancang untuk mencegah terjadinya kegagalan berantai (*Cascading Failure*) yang dapat menjatuhkan utas eksekusi utama aplikasi (Dragoni dkk., 2017). Karakteristik resiliensi dan *self-healing* ini menjadikan *microservices* standar arsitektur untuk menjamin keberlangsungan operasional telemetri jangka panjang.

2.6.2. *Cloud Computing* (*Google Cloud Platform*)

Cloud Computing merupakan model yang memungkinkan akses jaringan secara terpusat, di mana saja, dan sesuai permintaan (*on-demand*) terhadap sekumpulan sumber daya komputasi yang dapat dikonfigurasi secara cepat, seperti *Server*, jaringan, dan ruang penyimpanan (Mell & Grance, 2011). Pada tingkat implementasi *Infrastructure as a Service* (IaaS), penyedia layanan

seperti Google Cloud Platform (GCP) mengalokasikan unit komputasi virtual (Virtual Machine) yang memiliki alamat IP publik statis. Ketersediaan *Server cloud* yang beroperasi secara terus-menerus (24/7) merupakan syarat mutlak dalam topologi telemetri agrikultur untuk menjamin ketersediaan penerimaan (*availability*) matriks data dari *edge device* yang berada di tempat terpisah.

2.6.3. Kontainerisasi (Docker dan Docker Compose)

Kontainerisasi adalah metode virtualisasi pada tingkat sistem operasi yang mendistribusikan perangkat lunak beserta seluruh dependensi, *libraries*, dan berkas konfigurasinya ke dalam unit standar yang disebut kontainer (Merkel, 2014). Docker menyediakan isolasi proses secara mandiri, sehingga kontainer yang mengeksekusi *Server database* tidak akan mengalami konflik porta atau dependensi dengan kontainer yang menjalankan broker pesan. Lebih lanjut, orkestrasi beberapa kontainer dapat dikendalikan menggunakan Docker Compose. Teknologi ini mengizinkan penciptaan *Bridge Network internal*, di mana komunikasi antarkontainer dilakukan melalui nama layanan (DNS *Resolving*) tanpa perlu mengekspos porta mereka ke jaringan internet publik, sehingga secara fundamental meningkatkan lapisan keamanan topologi sistem.

2.7.Teknologi Pengembangan Platform Web

2.7.1. Node JS dan Manajemen Memori

Node.js merupakan lingkungan eksekusi (*runtime*) JavaScript sisi *server* yang beroperasi pada arsitektur utas tunggal (*single-threaded*) dengan model *event-driven* dan *non-blocking I/O*. Arsitektur asinkron ini memampukan *server* untuk menangani ribuan antrean koneksi jaringan secara simultan, seperti lalu lintas telemetri sensor dan injeksi muatan HTTP, tanpa memblokir siklus eksekusi utama CPU (Tilkov & Vinoski, 2010).

Dalam skenario transmisi data gambar resolusi tinggi, mesin *server* dituntut untuk mengalokasikan ruang memori dengan ukuran yang besar untuk menampung dan memproses rentetan karakter penyandian Base64. Untuk mencegah terjadinya kebocoran memori (*Memory Leak*) yang dapat berujung

pada penghentian sistem (*Out-Of-Memory Crash*), Node.js ditenagai oleh mesin V8 (*V8 JavaScript Engine*) yang mengimplementasikan algoritma *Garbage Collection* otomatis (Ornbo, 2012). Mekanisme *Garbage Collection* beroperasi di latar belakang dengan memindai ruang memori secara berkala, mengidentifikasi objek atau variabel matriks yang tidak lagi direferensikan dalam siklus eksekusi, dan secara otonom membebaskan blok memori tersebut untuk digunakan kembali. Manajemen memori *persistent* ini memastikan penggunaan *Random Access Memory* (RAM) pada *server* tetap berada dalam kesetimbangan, meskipun terus-menerus dibombardir oleh beban data gambar berukuran besar.

2.7.2. Eclipse Mosquitto

Eclipse Mosquitto adalah perangkat lunak message broker *open-source* yang mengimplementasikan protokol *Message Queuing Telemetry Transport* (MQTT) versi 5.0, 3.1.1, dan 3.1. Ditulis dalam bahasa pemrograman C, Mosquitto dirancang secara spesifik dengan arsitektur memori yang sangat efisien dan ringan (*lightweight*), sehingga sangat ideal digunakan pada semua lingkungan, mulai dari perangkat berdaya rendah hingga *Server cloud* berkinerja tinggi (Light, 2017). Dalam topologi *Publish*/*Subscribe*, Mosquitto bertindak sebagai distributor sentral yang memvalidasi otorisasi, memfilter topik, dan mendistribusikan lalu lintas muatan data (*payload*) dari perangkat *Edge* menuju *Client Backend* maupun antarmuka pengguna akhir secara waktu nyata (*real-time*).

2.7.3. React.js

React.js adalah library *frontend* JavaScript untuk membangun antarmuka pengguna interaktif menggunakan komponen *reusable* dan Virtual DOM untuk update efisien. Hooks seperti useState memungkinkan manajemen state dinamis untuk tampilan data *real-time* dari WebSocket. React cocok untuk dashboard hidroponik karena membutuhkan data *real-time* serta ekosistemnya yang besar (Strapi, 2024).

2.7.4. PostgreSQL dan Connection Pooling

PostgreSQL merupakan sistem manajemen *database* relasional berbasis objek (*Object-Relational Database Management System* / ORDBMS) sumber terbuka yang dirancang untuk menangani beban kerja dengan tingkat konkurensi tinggi secara aman. Pada lingkungan *microservices* bervolume tinggi, prosedur membuka dan menutup soket TCP secara berulang untuk setiap eksekusi kueri pangkalan data (seperti injeksi matriks telemetri) akan menciptakan hambatan komputasi laten yang menguras kapasitas siklus *server* (Silberschatz dkk., 2020).

Untuk mereduksi inefisiensi tersebut, integrasi komunikasi antara mesin Node.js dan PostgreSQL diimplementasikan melalui mekanisme Pangkalan Koneksi (*Connection Pooling*). *Connection Pooling* adalah metode pemeliharaan sekumpulan soket koneksi aktif (*cache*) di dalam memori *server* yang dapat digunakan secara bergantian oleh utas aplikasi tanpa harus melakukan inisiasi *handshake database* dari awal pada tiap transaksi (Hussain dkk., 2022).

Selain mengoptimalkan kecepatan kueri, implementasi pangkalan koneksi ini mendongkrak ketahanan sistem (*resilience*). Ketika layanan *database* mengalami kelumpuhan paksa, pustaka *pooler* di sisi *server* aplikasi tidak akan hancur (*crash*). Pustaka tersebut akan menahan antrean kueri dan secara otonom mengaitkan ulang soket yang terputus di dalam jaringan lokal internal (seperti *Docker Bridge*) sesaat setelah porta *database* kembali terbuka. Mekanisme inilah yang menjamin *Recovery* otonom (*self-healing*) pada koneksi pangkalan data dapat dieksekusi hanya dalam hitungan detik.

2.8. Parameter Pengujian Quality of Service (QoS)

QoS merupakan metode pengukuran yang digunakan untuk menentukan kemampuan dan keandalan sebuah jaringan dalam menyediakan layanan pengiriman data. Dalam penelitian ini, evaluasi performa jaringan mengacu pada standardisasi yang ditetapkan oleh ETSI (*European Telecommunications Standards*

Institute) melalui dokumen standarisasi TIPHON (*Telecommunications and Internet Protocol Harmonization Over Networks*).

2.8.1. *Throughput*

Throughput merupakan kecepatan transfer data yang berhasil diukur pada suatu saluran transmisi dalam satuan waktu tertentu, biasanya dinyatakan dalam bit per detik (bps). Dalam ekosistem IoT, *Throughput* yang tinggi menjamin aliran data gambar dapat ditransmisikan secara kontinu. Parameter ini sangat dipengaruhi oleh kapasitas bandwidth nirkabel dan besaran overhead antrean bawaan protokol (Paessler, 2017). Persamaan untuk menghitung *Throughput* ditunjukkan pada Persamaan (1).

$$Throughput = \frac{Jumlah\ data\ yang\ dikirim}{Waktu\ pengiriman\ data} \quad (1)$$

Dalam standardisasi TIPHON, performa *Throughput* dapat diklasifikasikan berdasarkan persentase perbandingan antara *Throughput* terhadap bandwidth maksimum jaringan, seperti yang ditunjukkan pada Tabel 2.2.

Tabel 2.2 Klasifikasi Throughput Menurut TIPHON

Kategori Kualitas	Persentase *Throughput*	Indeks
Sangat Bagus	75% - 100%	4
Bagus	50% - < 75%	3
Sedang	25% - < 50%	2
Buruk	< 25%	1

(Sumber: ETSI TR 101 329)

2.8.2. Latensi

Latensi adalah waktu tempuh paket data dari sumber ke tujuan, termasuk delay propagasi dan pemrosesan. Latensi tinggi menyebabkan penundaan update dashboard, memengaruhi responsivitas sistem hidroponik. WebSocket mengurangi latensi dengan koneksi *persistent* (TEKTELIC, 2023).

$$Latency = Waktu_{Terima} - Waktu_{kirim} \quad (2)$$

$$Rata - rata\ Delay = \frac{Total\ Delay}{Total\ Paket\ yang\ Diterima} \quad (3)$$

Menurut standar TIPHON, tingkat kelayakan latensi untuk layanan yang membutuhkan komunikasi interaktif dan streaming diklasifikasikan pada Tabel 2.3.

Tabel 2.3 Klasifikasi Latensi Menurut TIPHON

Kategori Kualitas	Nilai Latensi	Indeks
Sangat Bagus	< 150 ms	4
Bagus	150 ms - 300 ms	3
Sedang	300 ms - 400 ms	2
Buruk	> 400 ms	1

(Sumber: ETSI TR 101 329)

2.8.3. *Packet Loss*

Packet Loss adalah persentase paket data hilang selama transmisi, idealnya <0,5-1% untuk VoIP dan streaming IoT agar menghindari gangguan. Penyebab utamanya termasuk kemacetan jaringan dan interferensi, yang dievaluasi dengan retransmisi protokol. Pada QoS hidroponik, *Packet Loss* rendah memastikan integritas data citra (Paessler, 2017).

$$Packet\ Loss\ (\%) = \frac{(Paket\ Dikirim - Paket\ Diterima)}{Paket\ Dikirim} \times 100\% \quad (4)$$

Standardisasi TIPHON menetapkan toleransi kehilangan paket dengan metrik klasifikasi seperti pada Tabel 2.3.

Tabel 2.4 Klasifikasi *Packet Loss* Menurut TIPHON

Kategori Kualitas	Persentase *Packet Loss*	Indeks
Sangat Bagus	0% - 2%	4
Bagus	3% - 14%	3
Sedang	15% - 24%	2
Buruk	≥ 25%	1

(Sumber: ETSI TR 101 329)

2.8.4. *Jitter*

Jitter atau variasi delay adalah fluktuasi ketidakstabilan waktu kedatangan antar paket data di sisi penerima. Pada transmisi berbasis urutan (stream), *Jitter* yang tinggi mengindikasikan antrean komputasi yang tidak konsisten pada lapisan transportasi, yang berisiko menciptakan distorsi penerimaan gambar beruntun. Standar TIPHON menetapkan bahwa untuk layanan video/streaming yang baik, nilai *Jitter* sebaiknya dijaga seminimal mungkin (mendekati 0 ms) agar rekonstruksi data gambar di sisi penerima dapat berjalan mulus (ETSI, 2020).

$$Jitter = \frac{Total\ variasi\ delay}{Total\ paket\ yang\ diterima} \quad (5)$$

Tabel 2.5 menguraikan klasifikasi *Jitter* berdasarkan batasan rekomendasi TIPHON untuk menjaga stabilitas penerimaan paket.

Tabel 2.5 Klasifikasi *Jitter* Menurut TIPHON

Kategori Kualitas	Nilai *Jitter*	Indeks
Sangat Bagus	0 ms - 75 ms	4
Bagus	76 ms - 125 ms	3
Sedang	126 ms - 225 ms	2
Buruk	> 225 ms	1

(Sumber: ETSI TR 101 329)

2.9. Metode Pengujian dan Keandalan Sistem Perangkat Lunak

Subbab ini menguraikan metodologi validasi komprehensif yang diaplikasikan untuk mengevaluasi parameter spesifikasi perangkat lunak. Pengujian tidak hanya ditujukan untuk mengukur kesesuaian fungsional, tetapi juga dirancang untuk menguji tingkat penerimaan pengguna akhir serta mengevaluasi resiliensi arsitektur saat dihadapkan pada kegagalan operasional (*fault-injection*).

2.9.1. Pengujian Fungsionalitas (Blackbox)

Pengujian *Blackbox* adalah metode evaluasi perangkat lunak yang berfokus pada spesifikasi fungsional masukan (*input*) dan luaran (*output*) tanpa menganalisis, membedah, atau mengetahui struktur logika kode internal dari aplikasi tersebut (Pressman, 2014). Metode ini bertujuan untuk memvalidasi apakah sistem terintegrasi telah memfasilitasi seluruh skenario operasional yang didefinisikan pada tahap awal analisis kebutuhan.

Dalam konteks pengembangan arsitektur *microservices* IoT, pengujian *Blackbox* difokuskan untuk mengevaluasi keberhasilan komunikasi antar-layanan (*inter-service communication*). Parameter uji mencakup ketepatan perutean lalu lintas gerbang aplikasi (*API Gateway Routing*), validasi mekanisme otorisasi keamanan seperti *JSON Web Token* (JWT), manajemen resolusi kebijakan akses lintas domain (*CORS*), hingga memastikan bahwa muatan telemetri dari perangkat *Client* berhasil disimpan ke dalam tabel *database* tanpa anomali (Sommerville, 2015). Pencapaian absolut pada pengujian ini menjadi bukti empiris bahwa perangkat lunak bebas dari cacat logika (*logical defect*) yang dapat menginterupsi aliran data.

2.9.2. Pengujian Penerimaan (UAT) dan Technology Acceptance Model

User Acceptance Test (UAT) mengevaluasi fungsionalitas sistem secara empiris saat dioperasikan langsung oleh pengguna akhir pada kondisi lapangan (Sommerville, 2015). Evaluasi hasil pengujian ini dilandaskan pada *Technology Acceptance Model* (TAM) yang menetapkan bahwa penerimaan

pengguna terhadap suatu teknologi sangat ditentukan oleh indikator kebermanfaatan dan kemudahan penggunaan (An dkk., 2023). Dalam konteks sistem pendukung keputusan, aspek kebermanfaatan sangat bergantung pada integritas data. Penggunaan data rekomendasi yang belum divalidasi oleh pakar secara alamiah akan menurunkan nilai fungsi praktis di mata pengguna. Hal tersebut menjadikan penilaian netral sebagai konsekuensi logis sekaligus indikator evaluasi yang valid, terlepas dari seberapa sempurna arsitektur perangkat lunak yang telah dibangun.

2.9.3. Pengujian Keandalan Operasional (*Reliability Testing*)

Pengujian Keandalan Operasional memvalidasi konsistensi performa sistem saat menghadapi tekanan durasi ekstrem dan anomali kegagalan (Myers dkk., 2011). Pengujian ini berfokus pada dua parameter: Pengujian Rendam (*Soak Testing*) untuk mendeteksi degradasi laten seperti *Thermal Throttling* pada perangkat fisik atau kebocoran memori (*Memory Leak*) di *server*, serta Injeksi Kegagalan (*Fault-Injection*) untuk memvalidasi kapabilitas Toleransi Kesalahan (*Fault Tolerance*). Parameter keberhasilan diukur dari kemampuan arsitektur perangkat lunak dalam mengeksekusi penanganan eksepsi (*Exception Handling*) untuk mencegah kelumpuhan total, dan keberhasilan melakukan *Recovery* jaringan otonom (*Auto-Reconnect*) dengan durasi jeda (*Recovery Time*) seminimal mungkin (Kuo dkk., 2001).

BAB III

METODE PENELITIAN

3.1. Rancangan Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan jenis riset eksperimental. Menurut Sugiyono (2022), riset eksperimental merupakan metode penelitian kuantitatif yang digunakan untuk mencari pengaruh perlakuan tertentu terhadap variabel lain dalam kondisi yang dikendalikan. Dalam rancangan eksperimen ini, protokol komunikasi lapisan aplikasi (HTTP, WebSocket, dan MQTT) ditetapkan sebagai variabel bebas. Kinerja sistem ditetapkan sebagai variabel terikat yang diukur melalui parameter standar QoS meliputi *End-to-End Latency*, *Throughput*, *Packet Loss*, dan *Jitter*. Eksperimen ini mengisolasi variabel kontrol tersebut, yaitu metode transmisi data dengan interval diskrit 3 detik. Pengujian dieksekusi dengan durasi observasi selama 15 menit per protokol (dibagi menjadi 3 sesi, 5 menit per sesi), membawa muatan citra statis berformat Base64 menggunakan infrastruktur jaringan Wi-Fi 5G berkapasitas tinggi untuk memastikan validitas komparasi yang setara

Teknik pengumpulan data dilaksanakan melalui metode observasi empirik menggunakan instrumen perangkat lunak penganalisis jaringan (*packet sniffer*). Pengumpulan data lalu lintas jaringan (*network traffic logging*) pada lapisan transport dieksekusi secara pasif menggunakan tcpdump di dalam lingkungan *Virtual Machine Google Cloud Platform* (GCP). Menurut Kurose dan Ross (2021), *packet sniffer* bertugas merekam salinan seluruh pesan yang dikirim dan diterima oleh simpul akhir jaringan tanpa menginterupsi atau memodifikasi interaksi protokol yang sedang berjalan.

Untuk melengkapi perekaman jaringan, pengumpulan data waktu tempuh aplikasi (*End-to-End Latency*) dilakukan melalui mekanisme perekaman *timestamp*. Data waktu pengiriman dari perangkat *Edge* (Raspberry Pi) disandingkan dengan waktu penerimaan pada Node.js, yang kemudian diekstraksi menggunakan kueri SQL

pada pangkalan data PostgreSQL. Seluruh pengumpulan data dilakukan secara langsung di area *greenhouse* hidroponik untuk menjamin representasi kondisi lingkungan lapangan aslinya.

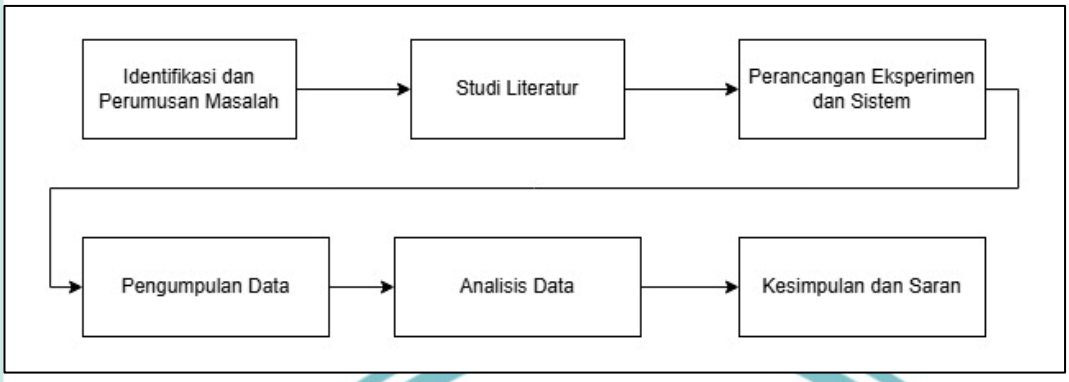
Data kuantitatif mentah berupa berkas rekaman jaringan (.pcap) diekstraksi ke dalam format *Comma Separated Values* (CSV) menggunakan penganalisis protokol Wireshark. Seluruh metrik jaringan yang terekam dievaluasi menggunakan teknik analisis statistik deskriptif komparatif, yang meliputi agregasi nilai rata-rata (*Throughput* dan *Latency*) serta simpangan baku untuk parameter *Jitter*. Analisis komparatif kuantitatif adalah metode yang digunakan untuk membandingkan keberadaan satu variabel atau lebih pada dua sampel atau lebih pada waktu yang berbeda (Sugiyono, 2022).

Proses analisis kuantitatif ini disandarkan pada standardisasi TIPHON (*Telecommunications and Internet Protocol Harmonization Over Network*) untuk menjustifikasi komparasi nilai metrik antarprotokol. Hasil analisis statistik ini digunakan untuk membuktikan secara empiris hubungan sebab akibat antara *Application-Layer Bloat* dan mekanisme sinkron *(Half-Duplex)* pada HTTP maupun penalti komputasi XOR *Masking* pada WebSocket terhadap degradasi efisiensi transmisi data di lingkungan *Edge* Computing berspesifikasi terbatas.

3.2. Tahapan Penelitian

Penelitian ini dilaksanakan dengan mengadopsi alur metode penelitian kuantitatif jenis eksperimental. Pendekatan kuantitatif eksperimental mensyaratkan adanya langkah-langkah yang terukur dan sistematis, mulai dari penetapan masalah hingga penarikan kesimpulan berdasarkan data numerik.

Merujuk pada kerangka kerja tersebut, alur operasional penelitian ini dirumuskan ke dalam enam tahapan utama yang direpresentasikan pada Gambar 3.1 berikut:



Gambar 3.1 Tahapan Penelitian

Rincian dari masing-masing tahapan eksperimental pada gambar 3.1 dijelaskan sebagai berikut:

1. Identifikasi dan Perumusan Masalah

   Tahap ini berfokus pada observasi kendala transmisi data gambar visual pada sistem pendeteksi hama sebelumnya. Berdasarkan identifikasi masalah tersebut, peneliti menetapkan rumusan dan batasan masalah yang difokuskan pada evaluasi komparatif efisiensi protokol komunikasi.

2. Studi Literatur

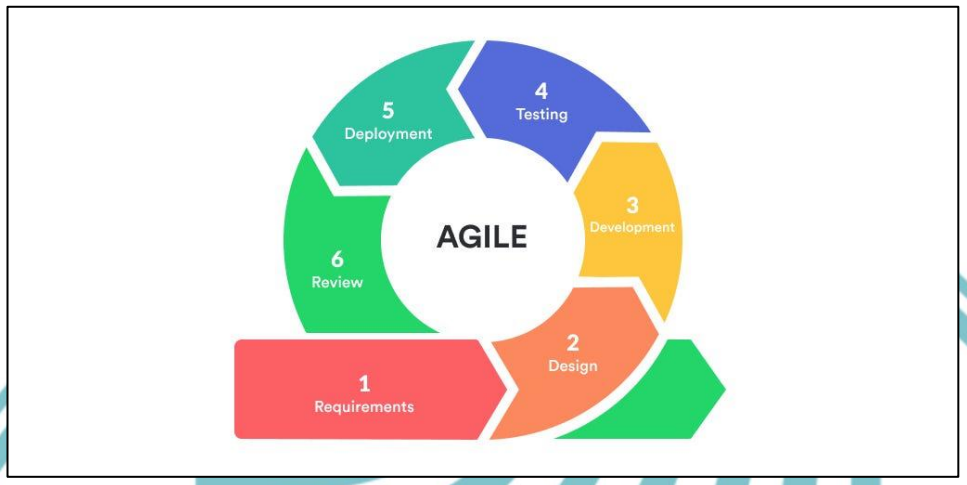
   Tahap ini melibatkan penelusuran literatur untuk membangun kerangka pemikiran eksperimen. Kajian utama mencakup dasar teori karakteristik protokol lapisan aplikasi, parameter standardisasi kualitas jaringan (TIPHON), serta identifikasi celah penelitian dari studi terdahulu.

3. Perancangan Eksperimen dan Sistem

   Tahap ini merupakan fase penetapan variabel penelitian dan perancangan infrastruktur sistem. Peneliti menentukan variabel bebas (protokol komunikasi), variabel terikat (metrik kinerja jaringan), dan variabel kontrol (resolusi citra dan interval transmisi), serta menyusun topologi jaringan yang memastikan pengujian berjalan *apple-to-apple*.

   Selain perancangan variabel, tahap ini mencakup proses pembangunan platform pemantauan secara teknis. Pengembangan perangkat lunak pada penelitian ini menggunakan metode *Agile*. Menurut Pressman dan Maxim (2020), *Agile* merupakan pendekatan iteratif yang berfokus pada

penyelesaian fungsionalitas sistem secara bertahap, kolaborasi tim yang adaptif, dan respons cepat terhadap perubahan. Siklus iterasi *Agile* yang diterapkan pada penelitian ini divisualisasikan pada Gambar 3.2.



Gambar 3.2. Siklus Pengembangan Perangkat Lunak Metode Agile

Berdasarkan Gambar 3.2, pengembangan sistem dilakukan melalui enam tahapan berurutan dalam setiap iterasinya:

1) *Requirements*

   Tahap pengumpulan dan analisis spesifikasi teknis, meliputi identifikasi kebutuhan protokol komunikasi, penentuan arsitektur *microservices*, serta penetapan parameter *Quality of Service* yang akan diukur pada fase eksperimen.

2) *Design*

   Tahap pembuatan cetak biru sistem. Peneliti merancang rute komunikasi di dalam docker bridge, menyusun skema pangkalan data PostgreSQL, dan menentukan topologi jaringan fisik antara perangkat edge dan server.

3) *Development*

   Tahap penulisan kode program secara modular. Implementasi dieksekusi secara bertahap, dimulai dari pembangunan *backend* Node.js, penyusunan antarmuka web, hingga penulisan skrip aktuator Python pada Raspberry Pi.

4) Testing

Tahap verifikasi fungsionalitas. Setiap komponen layanan yang selesai dikembangkan langsung dievaluasi melalui pengujian *Blackbox* untuk memastikan ketiadaan cacat logika pada manajemen otentikasi maupun perutean data.

5) Deployment

Tahap peluncuran sistem ke lingkungan operasional. Kontainer layanan didistribusikan ke infrastruktur *Google Cloud Platform*, sementara skrip pengirim data dipasang secara langsung pada perangkat edge di lapangan.

6) Review

Tahap evaluasi akhir terhadap sistem yang telah diluncurkan. Peneliti melaksanakan *User Acceptance Test* dan memastikan seluruh arsitektur perangkat lunak telah berjalan stabil sebelum mengeksekusi fase pengumpulan data empiris.

4. Pengumpulan Data

Pada tahap ini, perlakuan eksperimental dieksekusi secara langsung di lingkungan *greenhouse* menggunakan konektivitas seluler aktual. Data kuantitatif diekstraksi melalui metode perekaman lalu lintas jaringan (*network traffic logging*) dan pencatatan waktu transmisi sistem secara berkesinambungan.

5. Analisis Data

Data mentah hasil perekaman jaringan dikonversi dan diproses menggunakan metode analisis statistik deskriptif komparatif. Nilai metrik QoS dari masing-masing protokol dibandingkan langsung dengan indeks standardisasi TIPHON untuk mengidentifikasi tingkat performa dan anomali jaringan.

6. Kesimpulan dan Saran

Tahap akhir ini bertujuan untuk menyajikan temuan empiris secara objektif. Kesimpulan ditarik untuk menjawab arsitektur protokol mana yang terbukti

paling andal dan efisien, dilanjutkan dengan pemberian saran strategis untuk pengembangan sistem di masa mendatang.

3.3. Objek Penelitian

Objek utama yang menjadi sasaran dalam penelitian ini adalah metrik kinerja arsitektur protokol komunikasi jaringan pada lapisan aplikasi, yang secara spesifik mengomparasikan *Hypertext Transfer Protocol* (HTTP), WebSocket, dan *Message Queuing Telemetry Transport* (MQTT). Sasaran penelitian difokuskan pada evaluasi kemampuan, stabilitas jaringan, dan tingkat efisiensi pemrosesan dari ketiga protokol tersebut dalam mentransmisikan *payload* citra digital berformat Base64

Objek citra tersebut merupakan matriks data yang merepresentasikan luaran visual dari hasil komputasi model deteksi hama pada sistem pertanian hidroponik. Aliran data ini ditransmisikan menggunakan metode interval setiap 3 detik. Transmisi dieksekusi dari perangkat *Edge Computing* berspesifikasi terbatas (Raspberry Pi) menuju *Server backend* terpusat (Node.js) untuk dievaluasi, dan didistribusikan ke antarmuka *dashboard Client* (React.js). Evaluasi objek dititikberatkan pada kemampuan setiap arsitektur protokol dalam meminimalisasi *bottleneck*, baik yang bersumber dari akumulasi *overhead* inisiasi koneksi pada saluran jaringan maupun dari penalti komputasi kriptografi di tingkat perangkat keras.

BAB IV

HASIL DAN PEMBAHASAN

4.1.Analisis Kebutuhan

Analisis kebutuhan merupakan tahap fundamental yang menetapkan batasan operasional dan spesifikasi kapabilitas platform pemantauan hidroponik sebelum memasuki fase perancangan arsitektur. Tahap ini difokuskan pada standar kualitas layanan. Oleh karena itu, spesifikasi kebutuhan pada penelitian ini diklasifikasikan ke dalam dua domain utama, kebutuhan fungsional yang mendefinisikan fitur, layanan logika, dan aliran data antar-komponen, serta kebutuhan non-fungsional yang menetapkan batasan kendala arsitektural, termasuk metrik kinerja jaringan (*Quality of Service*) dan standar keandalan operasional.

4.1.1. Kebutuhan Fungsional

Kebutuhan fungsional mendefinisikan interaksi logis, fitur, dan kapabilitas spesifik yang harus mampu dieksekusi oleh sistem aplikasi (baik *backend* maupun *frontend*). Karena sistem ini dirancang untuk mengakomodasi transmisi gambar secara diskrit dan pemantauan jarak jauh, spesifikasi fungsionalitasnya diuraikan pada Tabel 4.1.

Tabel 4.1. Kebutuhan Perangkat Fungsional

ID	Fitur / Fungsionalitas	Deskripsi Kebutuhan
F-01	Autentikasi dan Otorisasi	Sistem *backend* harus memvalidasi perangkat *Edge* menggunakan API Key statis sebelum menerima transmisi data. Pada sisi *frontend*, sistem harus membatasi akses *dashboard* hanya untuk pengguna terdaftar melalui mekanisme otentikasi JSON *Web Token* (JWT).
F-02	Penerimaan Data Multi-Protokol	*Backend* harus memiliki *endpoints* dan *listeners* yang terisolasi untuk menerima

		payload JSON (berisi gambar Base64, jenis hama, dan tingkat kepercayaan) melalui protokol HTTP, WebSocket, dan MQTT secara independen.
F-03	*Dashboard* Ringkasan Pemantauan	*Frontend* harus menyajikan halaman *Dashboard* yang menampilkan akumulasi statistik deteksi, grafik tren serangan hama selama 7 hari terakhir, serta daftar peringatan terbaru.
F-04	Pemantauan Langsung (*Live Monitor*)	Sistem harus menyediakan halaman *Live Monitor* untuk me-render *feed* gambar yang dikirimkan oleh aktuator di kebun, lengkap dengan informasi latensi dan status deteksi hama yang sedang aktif.
F-05	Pelaporan dan Pencatatan Log	Sistem harus mampu menyimpan dan menyajikan riwayat *monitoring* pada halaman Laporan. Halaman ini dibagi menjadi ringkasan Sesi Monitoring dan Log Deteksi (*Raw*) yang memuat rincian waktu, jenis protokol, latensi spesifik, dan tautan untuk melihat bukti gambar.
F-06	Pengaturan Preferensi Sistem	Pengguna harus dapat memodifikasi preferensi aplikasi melalui halaman Pengaturan, termasuk mengubah batas ambang kepercayaan (*minimum confidence threshold*) secara dinamis untuk menyaring *motion blur* dari kamera slider.
F-07	Sistem Pendukung Keputusan (DSS)	Sistem harus secara otomatis memicu dan menampilkan rekomendasi tindakan mitigasi (preventif maupun kuratif) pada antarmuka pengguna hanya jika modul

		mendeteksi eksistensi hama pada frame gambar yang dikirimkan.

4.1.2. Kebutuhan Non-Fungsional

Kebutuhan non-fungsional mendefinisikan batasan arsitektural, standar kualitas layanan, keamanan, dan keandalan yang menjadi tolok ukur kelayakan sistem secara operasional. Kebutuhan ini menjadi parameter utama dalam tahap pengujian performa jaringan. Spesifikasinya diuraikan pada Tabel 4.2
.

Tabel 4.2. Kebutuhan Perangkat Fungsional

ID	Kategori Kinerja	Deskripsi Kebutuhan dan Target Standar
NF-01	Kinerja Jaringan (QoS)	Kualitas transmisi data antara perangkat *Edge* dan *server cloud* harus dapat dievaluasi menggunakan standardisasi TIPHON (*Telecommunications and Internet Protocol Harmonization Over Network*). Sistem harus mampu menjaga integritas pengiriman gambar Base64 di bawah kondisi jaringan seluler yang fluktuatif, dengan parameter ukur meliputi *Throughput*, *Packet Loss*, Delay, dan *Jitter*.
NF-02	Kinerja Latensi Aplikasi	Waktu tempuh *End-to-End Latency*, yang diukur dari saat perangkat *Edge* memulai transmisi hingga *backend* merekamnya ke dalam PostgreSQL, harus berada pada rentang yang dapat ditoleransi untuk pemantauan diskrit interval 3 detik (target toleransi latensi rata-rata < 1000 ms).

NF-03	Ketersediaan (*Availability*)	Arsitektur *microservices* pada *Virtual Machine* harus memiliki kebijakan auto-restart melalui Docker Daemon untuk memastikan layanan (seperti Mosquitto Broker dan Node.js) segera pulih secara otomatis jika terjadi interupsi daya atau *crash* pada tingkat sistem operasi.
NF-04	Responsivitas Antarmuka	Aplikasi web *Client* (React.js) harus dibangun menggunakan paradigma *Single Page Application* (SPA) agar perutean antarhalaman (seperti perpindahan dari *Dashboard* ke Laporan) terjadi seketika tanpa perlu memuat ulang keseluruhan dokumen HTML dari *Server* Nginx.

4.2. Perancangan Sistem

Perancangan sistem merupakan fase pemodelan arsitektural sebelum implementasi dilakukan. Perancangan platform pemantauan hidroponik ini mencakup spesifikasi perangkat keras, spesifikasi perangkat lunak, pemetaan topologi jaringan mikro (*microservices*), alur komunikasi data antarblok, struktur relasional *database*, serta *layout* antarmuka web.

4.2.1. Spesifikasi Perangkat Keras

Perangkat keras yang dibutuhkan mencakup infrastruktur *server cloud*, perangkat keras di sisi *Client* pengirim (kebun), serta perangkat pengolahan data. Kebutuhan perangkat keras ini diuraikan pada Tabel 4.3.

Tabel 4.3. Spesifikasi Perangkat Keras

No	Perangkat Keras	Deskripsi/Fungsi	Spesifikasi

1.	*Virtual Machine* (*Google Cloud Platform*)	Menyediakan infrastruktur server terpusat untuk menjalankan orkestrasi kontainer seluruh layanan.	Ubuntu Linux 22.04 LTS, e2-medium, 2 vCPU, 4GB RAM, 40GB SSD
2.	Raspberry Pi 4 Model B	*Single Board Computer* (SBC) yang mengeksekusi inferensi YOLO, kompresi gambar (JPEG), enkoding Base64, dan transmisi TCP/IP.	• 4GB RAM, Quad-Core Cortex-A72
3.	Modul Kamera IoT	Menangkap visual dari tanaman hidroponik untuk diproses oleh algoritma deteksi.	• Raspberry Pi Camera Module v2 (Sensor 8MP)
4.	ESP32 & Modul Motor Slider	Menggerakkan modul kamera pada lintasan rel.	32-bit MCU, Motor Stepper / Driver
5.	Router WiFi	Menyediakan konektivitas internet nirkabel dari lokasi hidroponik menuju *Server* GCP.	Jaringan 5G

4.2.2. Spesifikasi Perangkat Lunak

Infrastruktur perangkat lunak difokuskan pada pemenuhan *tech stack* yang mendukung pengembangan API, *database* relasional, *message brokering*, serta alat perekaman paket jaringan tingkat sistem operasi. Spesifikasi perangkat lunak yang didefinisikan dapat dilihat pada Tabel 4.4.

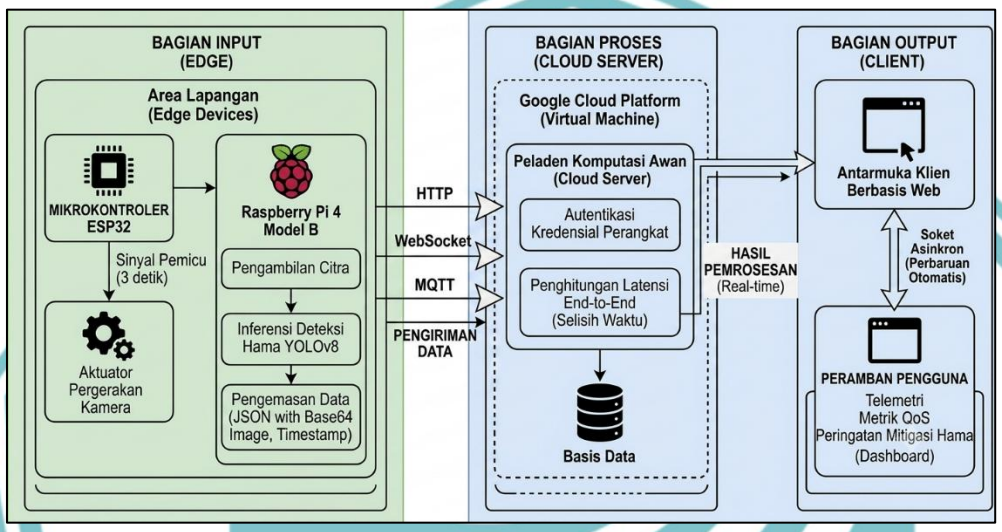
Tabel 4.4. Spesifikasi Perangkat Lunak

No	Perangkat Lunak	Deskripsi/Fungsi

1.	Ubuntu Server	Sistem operasi Linux pada VM untuk menjamin stabilitas eksekusi *Server* dan tcpdump.
2.	Docker & Docker Compose	Melakukan isolasi dan menghubungkan layanan (DB, API, Broker, *Frontend*) ke dalam jaringan bridge internal.
3.	Node.js	Lingkungan *runtime* eksekusi untuk membangun aplikasi *backend*.
4.	Eclipse Mosquitto	*Message broker* independen yang menangani lalu lintas Pub/Sub MQTT dan fitur konversi ke MQTT over WebSockets.
5.	React.js	Pustaka *frontend* untuk membangun dashboard UI yang menampilkan visual secara dinamis dari ketiga jalur protokol.
6.	PostgreSQL	*Database* relasional untuk menyimpan metrik latensi dan riwayat deteksi.
7.	Python & Pustaka IoT	Bahasa pemrograman pada Raspberry Pi untuk menangani pengolahan model (format TFLite/NCNN) dan transmisi data.
8.	Wireshark & tcpdump	Utilitas packet sniffer. tcpdump merekam paket di VM, Wireshark menganalisis metrik *Throughput*, *Jitter*, dan *Packet Loss* di PC.
9.	Nginx	Bertindak sebagai web server dan reverse proxy untuk lalu lintas data masuk.
8.	Visual Studio Code	Integrated Development Environment (IDE) untuk menyusun kode sumber seluruh arsitektur perangkat lunak.

4.2.3. Diagram Blok Sistem

Diagram blok mendeskripsikan topologi makro dari infrastruktur sistem secara keseluruhan. Sistem dapat dikelompokkan menjadi tiga bagian, yaitu input, process, dan output. Gambar 4.1 menunjukkan diagram blok dari sistem.



Gambar 4.1. Diagram Blok Sistem

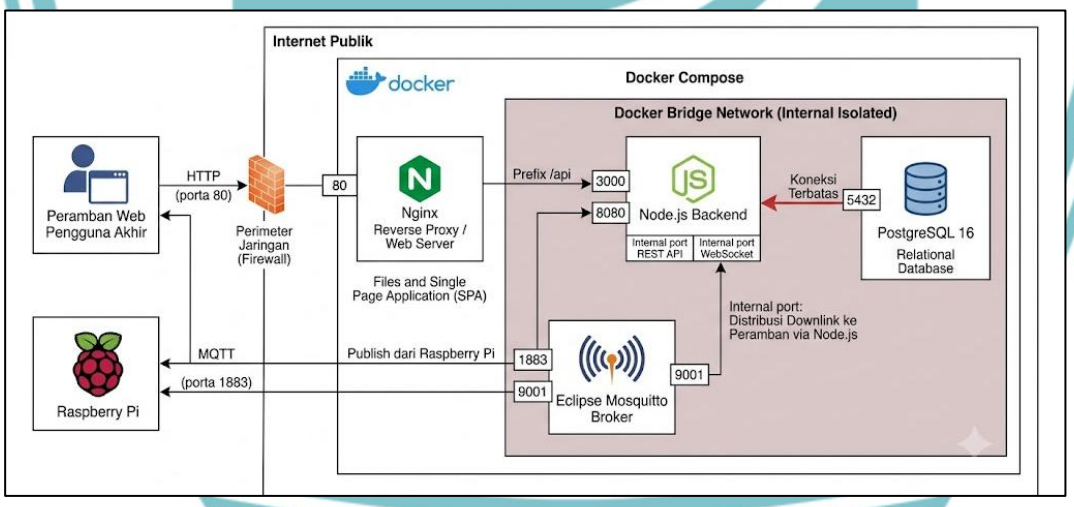
Pada bagian input di gambar 4.1, sistem memanfaatkan perangkat di area lapangan (*Edge*) yang terdiri dari mikrokontroler ESP32 dan Raspberry Pi 4 Model B. ESP32 berfungsi sebagai aktuator yang mengatur pergerakan kamera pada rel. lalu, Raspberry Pi menjalankan inferensi deteksi hama menggunakan YOLOv8, mengemas hasil deteksi ke dalam format JSON yang berisi matriks gambar Base64 beserta stempel waktu pengiriman, dan melakukan pengiriman citra setiap interval 3 detik.

Pada bagian proses, data yang dikirimkan oleh *Edge* akan diterima oleh *Server cloud* yang di-hosting pada *Virtual Machine Google Cloud Platform*. *Server* ini bertugas mengautentikasi kredensial perangkat, menghitung selisih waktu untuk mendapatkan metrik latensi *End-to-End*, dan menyimpan data riwayat tersebut ke dalam *database*. Proses penerimaan ini dirancang untuk mampu menangani tiga jalur protokol secara bersamaan, yakni HTTP, WebSocket, dan MQTT.

Pada bagian output, sistem mengirim hasil pemrosesan ke antarmuka *Client* berbasis web. *Client* web ini diakses oleh pengguna melalui browser untuk melihat pembaruan data telemetri, metrik QoS, dan peringatan mitigasi hama. Komunikasi antara *Server* dan *Client* dipertahankan menggunakan soket asinkron agar tampilan *dashboard* diperbarui secara otomatis tanpa perlu memuat ulang halaman.

4.2.4. Arsitektur Microservices dan Jaringan

Infrastruktur *Server* dibangun menggunakan paradigma *microservices* yang diorkestrasi oleh Docker Compose. Mekanisme ini memastikan seluruh layanan inti beroperasi di dalam satu jaringan internal yang terisolasi (*Docker Bridge Network*). Isolasi jaringan ini mencegah akses langsung dari internet publik ke komponen rentan seperti pangkalan data, sekaligus menghilangkan penalti latensi perutean eksternal antarwadah.



Gambar 4.2. Diagram Arsitektur *Microservices*

Seperti yang dapat dilihat pada gambar 4.2, pada lapisan antarmuka dan perutean, Nginx dikonfigurasi untuk beroperasi pada port 80 (HTTP). Layanan ini bertugas menyajikan berkas *Single Page Application* kepada pengguna akhir. Nginx juga bertindak sebagai *reverse proxy* yang meneruskan

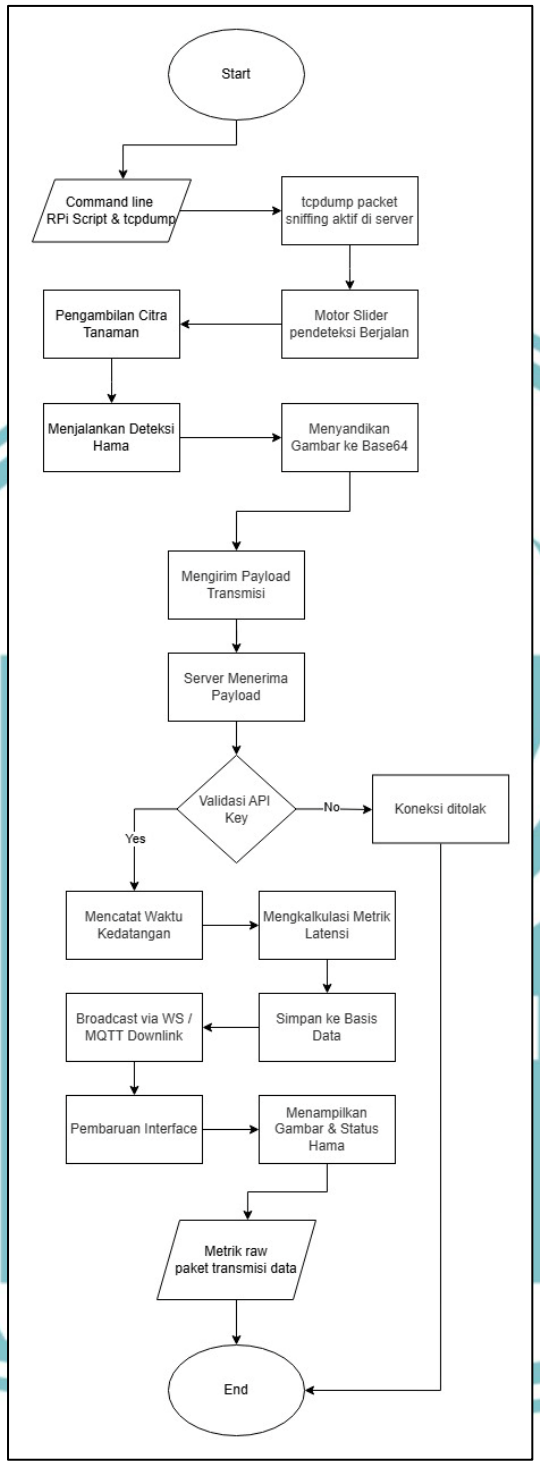
permintaan dengan awalan tertentu (seperti /api) langsung menuju layanan *backend* di dalam jaringan internal.

Pada lapisan logika bisnis dan pertukaran pesan, layanan Node.js bertindak sebagai *backend* utama yang berjalan pada port internal 3000 (REST) dan 8080 (WebSocket). Untuk menangani lalu lintas protokol ringan, Eclipse Mosquitto digunakan sebagai *message broker* yang membuka port publik 1883 untuk menerima publish dari Raspberry Pi, dan port internal 9001 untuk mendistribusikan *downlink* ke *browser*.

Pada lapisan penyimpanan, PostgreSQL 16 dioperasikan sebagai wadah *database* relasional. Layanan ini dikonfigurasi pada port internal 5432 dan sama sekali tidak mengekspos titik masuk publik. PostgreSQL hanya akan memproses kueri yang datang dari kontainer Node.js yang berada dalam satu jaringan bridge.

4.2.5. Flowchart Sistem

Alur kerja sistem berjalan berdasarkan mekanisme transmisi interval tanpa pemrosesan *redundant* di sisi *backend,* seperti yang dapat dilihat pada gambar 4.3 di bawah. Siklus dimulai ketika motor slider ESP32 berjalan, lalu Raspberry Pi melakukan pengambilan citra tanaman, menjalankan deteksi hama, menyandikan gambar ke format Base64, dan mengirimkan payload melalui salah satu protokol yang sedang aktif setiap interval waktu yang telah ditentukan (3 detik).



Gambar 4.3. Flowchart Sistem

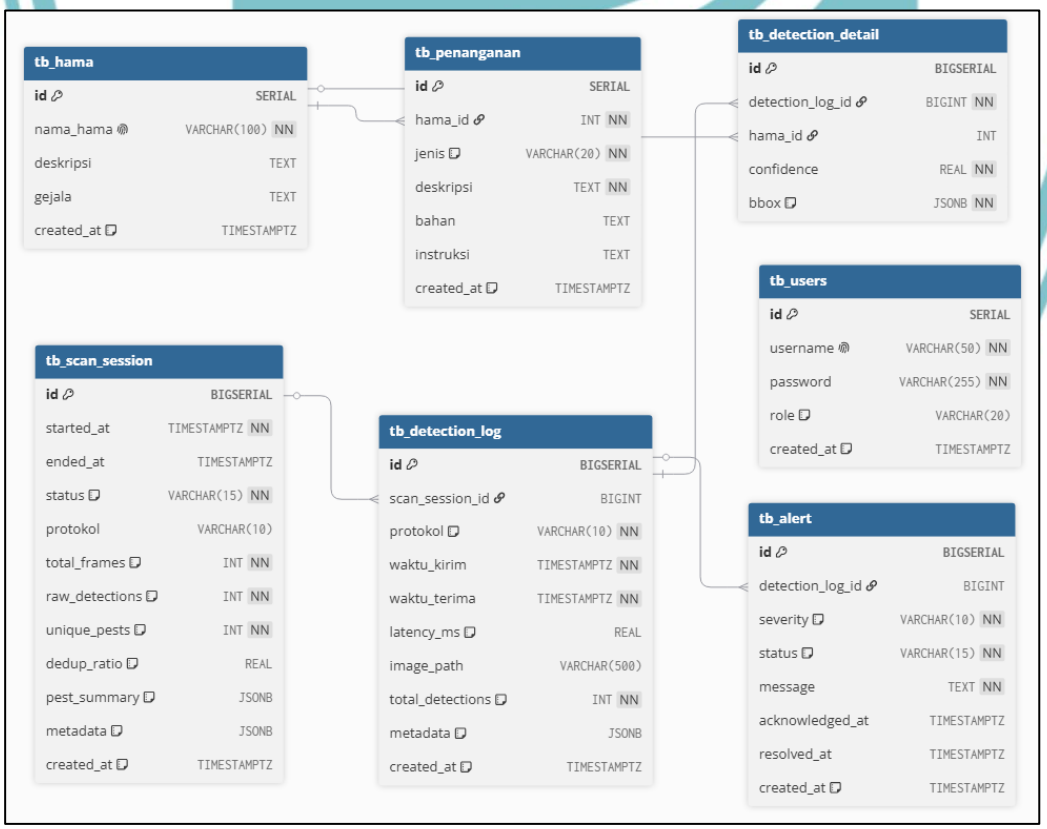
Setelah data ditransmisikan melintasi jaringan internet, *backend* pada *Server* menerima *payload* tersebut dan langsung memvalidasi API Key. Jika kredensial tidak sah, koneksi akan langsung ditolak. Jika sah, *backend* akan

mencatat stempel waktu kedatangan paket. Sistem kemudian mengkalkulasi waktu kedatangan tersebut dengan waktu pengiriman yang tertera pada *payload* untuk menghasilkan metrik latensi aplikasi.

Setelah kalkulasi latensi selesai, *backend* menyimpan data operasional tersebut ke dalam *database* relasional. Sebagai langkah terakhir, *backend* mengirimkan sinyal siaran (broadcast) melalui jalur WebSocket atau MQTT *downlink* menuju *Client* web. Sinyal ini memicu pembaruan antarmuka secara instan sehingga pengguna dapat memantau gambar dan status hama terbaru.

4.2.6. Perancangan *Database*

Skema *database* dirancang menggunakan model relasional yang dinormalisasi untuk menjaga integritas rekam jejak telemetri dan meminimalkan redundansi data. Arsitektur data ini direpresentasikan melalui *Entity-Relationship Diagram* (ERD) pada Gambar 4.4.



Gambar 4.4 Entity-Relationship Diagram (ERD) Sistem

Berdasarkan Gambar 4.4, rancangan *database* dibangun di atas beberapa entitas yang saling berelasi. Untuk menjaga keteraturan sesi pemindaian alat aktuator di kebun, log telemetri dikelompokkan menggunakan entitas tb_scan_session. Tabel ini memiliki relasi *One-to-Many* dengan tabel sentral tb_detection_log yang bertugas mencatat performa jaringan tingkat transport. Tabel tb_detection_log menyimpan parameter utama transmisi, meliputi jenis protokol yang digunakan, stempel waktu kirim dari perangkat keras, stempel waktu terima di *Server*, serta hasil komputasi metrik latensi untuk setiap frame gambar.

Data telemetri jaringan tersebut kemudian dikorelasikan dengan hasil inferensi kecerdasan buatan melalui relasi *One-to-Many* menuju tabel tb_detection_detail. Tabel sekunder ini menyimpan presisi matriks visual berupa koordinat kotak pembatas (*bounding box*) dan tingkat akurasi (*confidence*) dari setiap objek hama yang berhasil ditangkap pada satu bingkai gambar yang sama.

Untuk mendukung kapabilitas klasifikasi dan mitigasi otomatis, sistem mengimplementasikan entitas *master data*. Tabel tb_hama berperan menyimpan deskripsi spesifik jenis hama. Entitas ini memiliki relasi struktural *One-to-Many* dengan tabel tb_penanganan, yang menyimpan instruksi mitigasi preventif maupun kuratif untuk didistribusikan sebagai *output* Sistem Pendukung Keputusan (DSS) pada antarmuka pengguna.

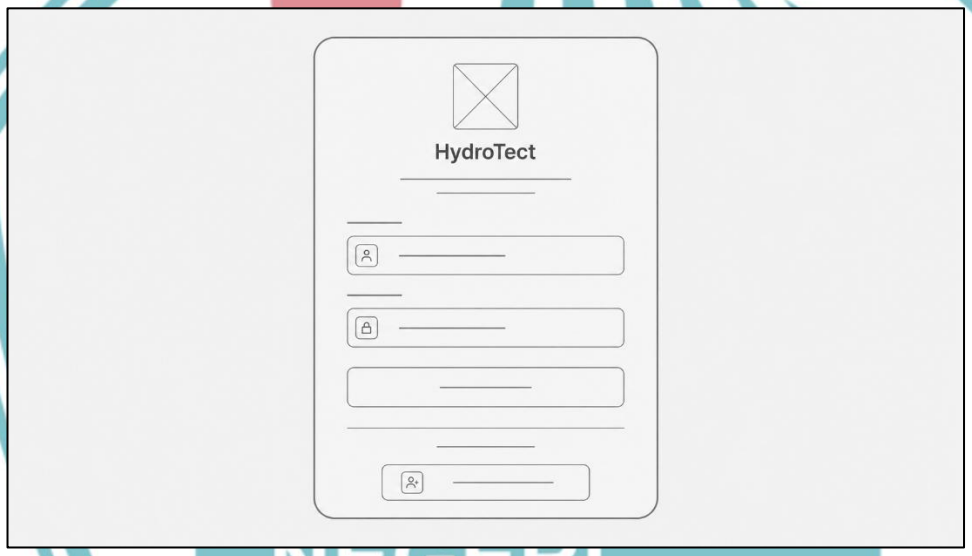
Sementara itu, data otentikasi administrator dikelola secara terisolasi dan independen di dalam entitas tb_users. Pemisahan entitas pengguna dari skema log telemetri yang berukuran besar ini direkayasa secara sengaja untuk memangkas beban kueri relasional (JOIN) dan memastikan efisiensi pemuatan data pada titik akhir antarmuka *dashboard* tetap optimal.

4.2.7. Perancangan Antarmuka Pengguna

Antarmuka pengguna (UI) dibangun sebagai *Single Page Application* (SPA) interaktif yang meniadakan proses *reload* halaman. Tata letak desain *wireframe* dirancang spesifik untuk merepresentasikan data metrik dan citra secara ergonomis, yang dibagi ke dalam enam halaman fungsional:

a) Halaman Login

Halaman pertama yang diakses oleh pengguna untuk proses autentikasi awal. Terdapat form input ID dan password untuk masuk ke dashboard aplikasi. Gambar 4.5 menampilkan gambaran rancangan halaman login.



Gambar 4.5. Rancangan Halaman Login

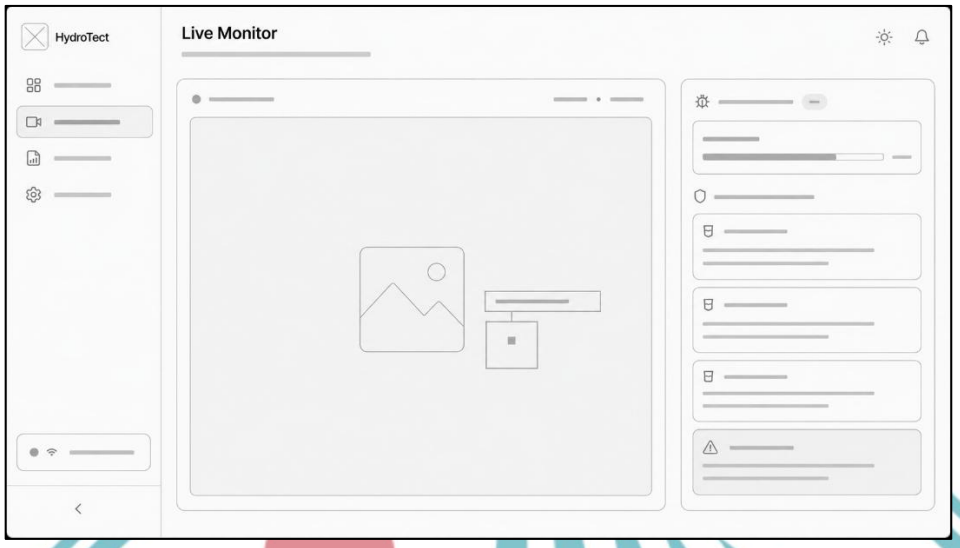
b) Halaman *Dashboard* (Dashboard)

Halaman utama yang memberikan ringkasan status perkebunan. *Dashboard* memuat kartu metrik statistik (total deteksi dan waktu aktif sistem), serta menyajikan grafik deret waktu (*time-series chart*) yang memvisualisasikan tren serangan hama selama 7 hari terakhir untuk mendukung analisis prediktif pengelola kebun. Gambar 4.6 di bawah ini menampilkan gambaran rancangan halaman *dashboard.*

Gambar 4.6. Rancangan Halaman Dashboard

c) Halaman Pemantauan Langsung (*Live Monitor*)

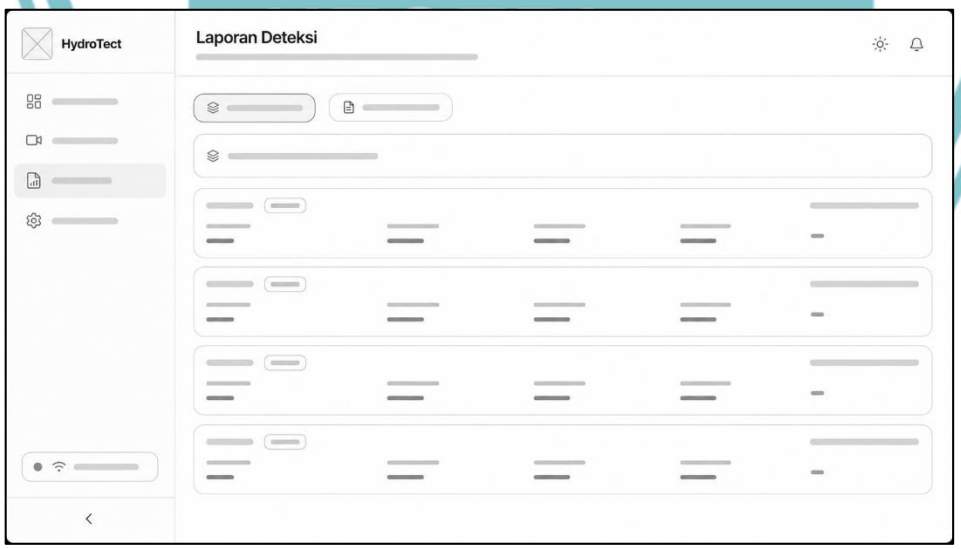
Halaman analitik *real-time* ini mendedikasikan porsi antarmuka utamanya untuk me-*render* citra matriks Base64 dari perangkat *Edge* ke dalam elemen kanvas HTML5 setiap 3 detik. Halaman ini terdapat fitur UI *Protocol Switcher* yang memberikan otonomi bagi pengguna untuk mengalihkan pemantauan parameter QoS di antara ketiga protokol secara dinamis. Peringatan mitigasi DSS akan tampil secara adaptif pada panel jika hama terdeteksi pada frame aktif. Gambar 4.7 di bawah ini menampilkan gambaran rancangan halaman *live monitor*.



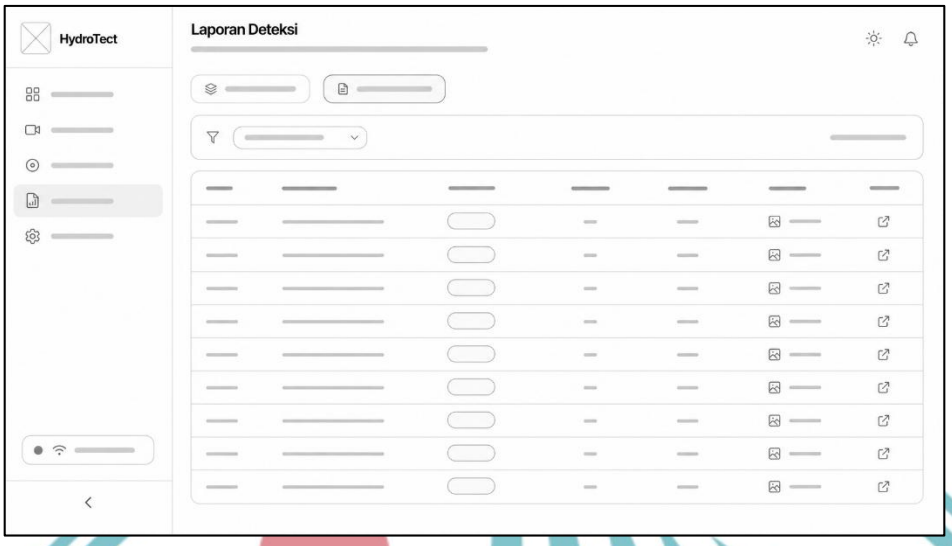
Gambar 4.7. Rancangan Halaman Live Monitor

d) Halaman Laporan Teragregasi

Halaman ini berfungsi sebagai arsip histori telemetri yang dipecah menjadi dua tabel, Sesi Monitoring (rekam jejak durasi penyisiran alat rel) dan Log Deteksi Mentah. Log memuat tabel kronologis yang menyajikan waktu kedatangan, ukuran muatan, nama protokol (HTTP/WS/MQTT), dan kalkulasi latensi dalam satuan milidetik per frame. Gambar 4.8 dan 4.9 di bawah ini menampilkan gambaran rancangan halaman Laporan



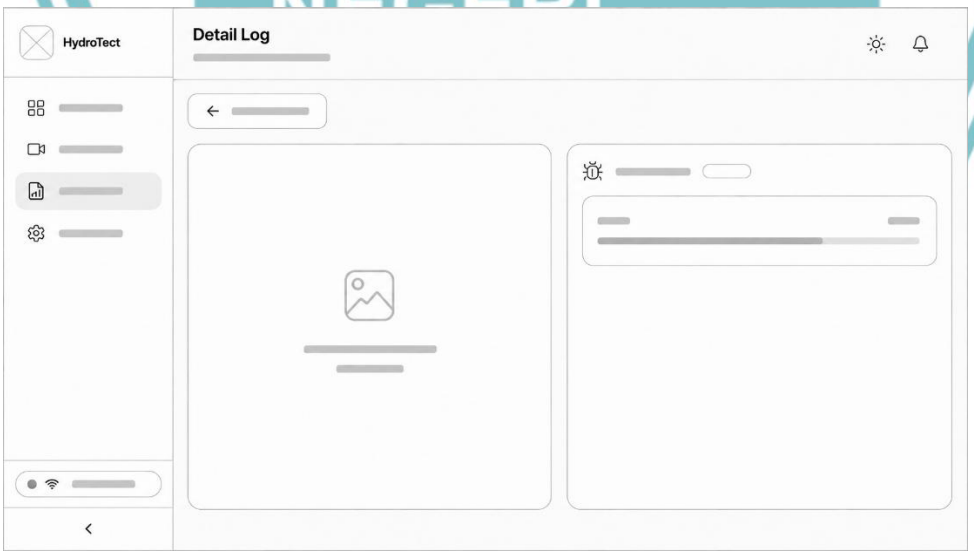
Gambar 4.8. Rancangan Halaman Laporan (Sesi *Monitoring*)



Gambar 4.9. Rancangan Halaman Laporan (Log Deteksi *Raw*)

e) Halaman Detail Log

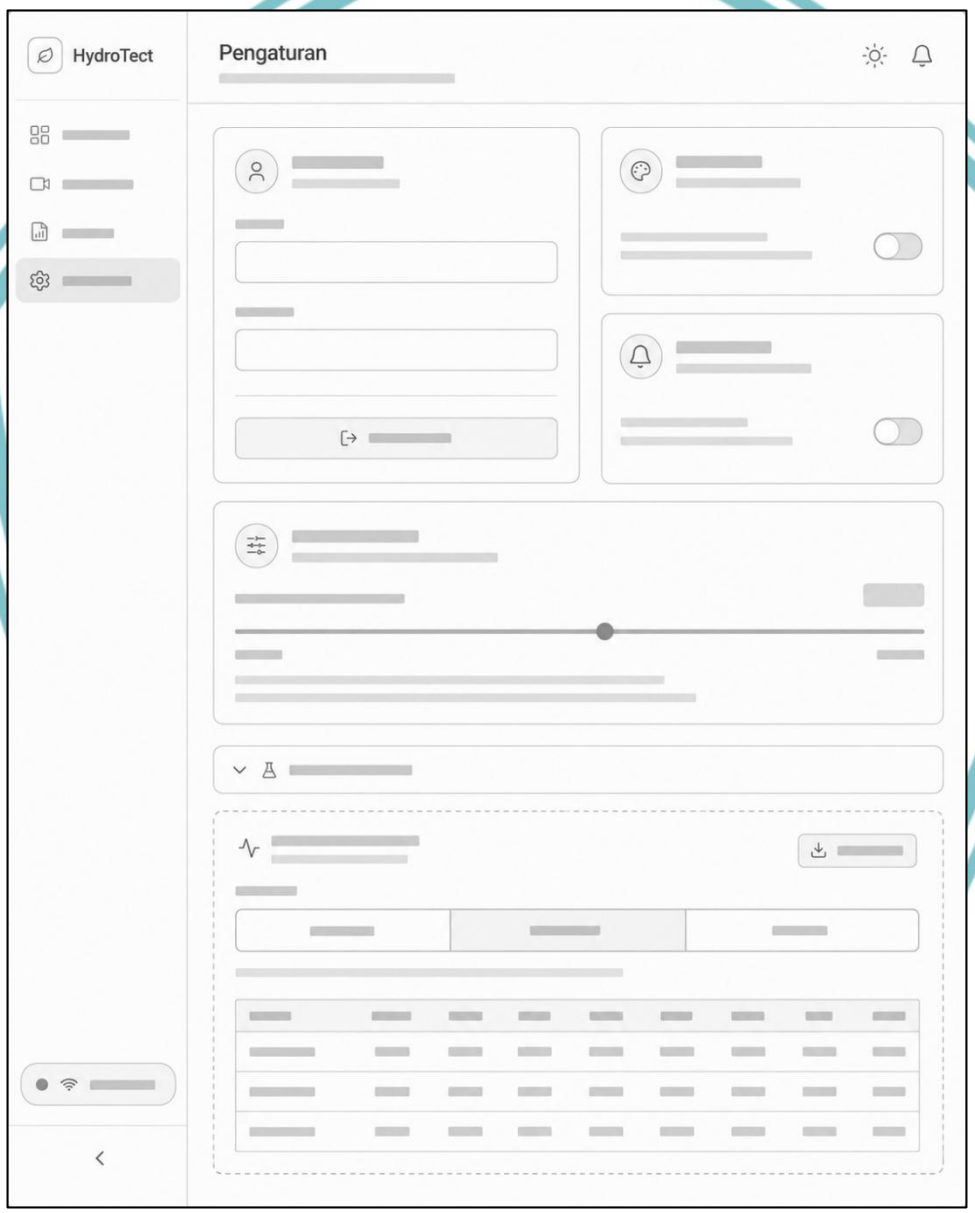
Antarmuka ekstensi dari tabel laporan yang menampilkan rincian presisi dari sebuah insiden deteksi tunggal. Halaman ini memvalidasi data tabular dengan menayangkan gambar fisik tangkapan kamera beserta kotak pembatas (*bounding box*) dan tingkat kepercayaan (*confidence score*) hasil inferensi. Gambar 4.10 di bawah ini menampilkan gambaran rancangan halaman detail log.



Gambar 4.10. Rancangan Halaman Detail Log

f) Halaman Pengaturan (*Settings*)

Pada halaman ini, pengguna dapat memodifikasi parameter parameter filter sistem *backend*. Pengguna memiliki kontrol untuk mengubah batas ambang (*minimum confidence threshold*) untuk memblokir inferensi model yang bernilai rendah akibat efek *motion blur* pada kamera aktuator. Gambar 4.11 di bawah ini menampilkan gambaran rancangan halaman *settings*



Gambar 4.11. Rancangan Halaman Pengaturan

4.3. Impelementasi Sistem

Subbab ini memaparkan tahapan realisasi dari perancangan arsitektur sistem ke dalam lingkungan produksi. Implementasi disusun berdasarkan alur pengembangan perangkat lunak yang terstruktur, dimulai dari lapisan penyimpan data, pengolahan logika aplikasi dan perpesanan (*backend* dan broker), penyajian antarmuka visual (*frontend*), hingga penyebaran sistem pada infrastruktur *cloud*.

4.3.1. Implementasi *Database*

Database diimplementasikan menggunakan PostgreSQL versi 16 yang diisolasi di dalam kontainer Docker. Pangkalan data ini tidak memiliki eksposur porta ke internet publik, sehingga seluruh operasi Input/Output (I/O) hanya dapat dilakukan oleh lapisan *backend* yang berada pada satu jaringan virtual.

Gambar 4.12 dan Tabel 4.5 memperlihatkan keseluruhan daftar tabel yang digunakan pada sistem, diambil melalui antarmuka Command Line Interface (CLI) PostgreSQL.

```
muhammadkhairumufid@hydrotect-server:~/hydrotect-platform$ docker exec -it lokatani-db psql -U postgres -d lokat
ani
psql (16.14)
Type "help" for help.

lokatani=# \dt
                List of relations
 Schema |        Name          | Type  |  Owner
--------+----------------------+-------+----------
 public | tb_alert             | table | postgres
 public | tb_detection_detail  | table | postgres
 public | tb_detection_log     | table | postgres
 public | tb_hama              | table | postgres
 public | tb_penanganan        | table | postgres
 public | tb_scan_session      | table | postgres
 public | tb_users             | table | postgres
(7 rows)

lokatani=#
```

Gambar 4.12 Daftar Tabel Sistem pada Antarmuka CLI PostgreSQL

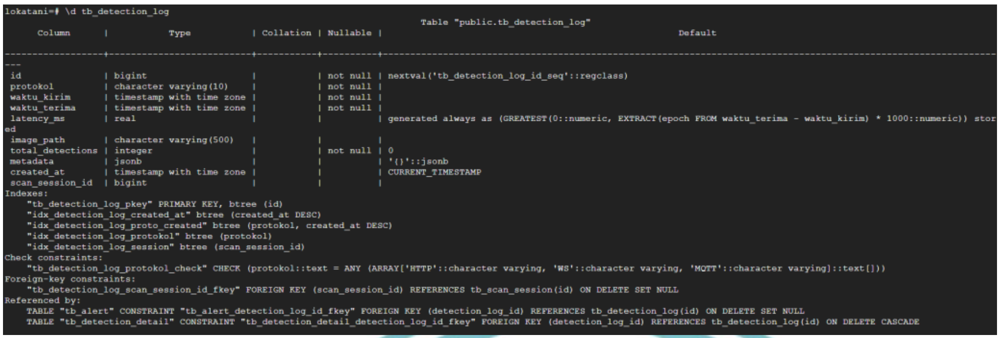
Tabel 4.5 Daftar Tabel Sistem

Nama Tabel	Deskripsi
tb_users	Menyimpan data akun pengguna pengelola (admin atau operator).

tb_hama	Menyimpan data utama (master data) jenis hama, gejala, dan deskripsinya.
tb_penanganan	Menyimpan tindakan mitigasi (preventif atau kuratif) untuk masing-masing hama.
tb_scan_session	Menyimpan log sesi pemindaian alat yang bergerak dari awal hingga akhir lintasan aktuator.
tb_detection_log	Merupakan tabel sentral yang menyimpan setiap pengiriman data visual dan log metrik jaringan antar-protokol.
tb_detection_detail	Menyimpan detail spesifik dari objek hama yang terdeteksi di dalam satu gambar (koordinat dan nilai confidence).
tb_alert	Menyimpan data notifikasi dan sistem peringatan otomatis untuk pengguna.

Tabel sentral yang diimplementasikan untuk mengakomodasi kebutuhan riset metrik QoS jaringan pada penelitian ini adalah tb_detection_log. Pada tabel ini, implementasi tipe data difokuskan untuk menjaga presisi waktu. Kolom waktu_kirim dan waktu_terima menggunakan tipe data TIMESTAMPTZ yang secara otomatis mengunci dan melacak zona waktu pada *timestamp*. Hal ini diterapkan untuk mencegah bias perhitungan latensi yang mungkin terjadi akibat perbedaan pengaturan zona waktu antara perangkat *client* (Raspberry Pi) dan *server cloud*.

Selain presisi waktu, tabel ini juga dilengkapi dengan fitur penghitungan latensi otomatis di tingkat database melalui kolom latency_ms. Rincian struktur atribut dan tipe data pada tabel tb_detection_log ditunjukkan pada Gambar 4.13 dan Tabel 4.6.



```
lokatani=# \d tb_detection_log
                                        Table "public.tb_detection_log"
     Column      |           Type           | Collation | Nullable |                          Default
-----------------+--------------------------+-----------+----------+------------------------------------------------------------
---
 id              | bigint                   |           | not null | nextval('tb_detection_log_id_seq'::regclass)
 protokol        | character varying(10)    |           | not null |
 waktu_kirim     | timestamp with time zone |           | not null |
 waktu_terima    | timestamp with time zone |           | not null |
 latency_ms      | real                     |           |          | generated always as (GREATEST(0::numeric, EXTRACT(epoch FROM waktu_terima - waktu_kirim) * 1000::numeric)) stor
ed
 image_path      | character varying(500)   |           |          |
 total_detections | integer                 |           | not null | 0
 metadata        | jsonb                    |           |          | '{}'::jsonb
 created_at      | timestamp with time zone |           |          | CURRENT_TIMESTAMP
 scan_session_id | bigint                   |           |          |
Indexes:
    "tb_detection_log_pkey" PRIMARY KEY, btree (id)
    "idx_detection_log_created_at" btree (created_at DESC)
    "idx_detection_log_proto_created" btree (protokol, created_at DESC)
    "idx_detection_log_protokol" btree (protokol)
    "idx_detection_log_session" btree (scan_session_id)
Check constraints:
    "tb_detection_log_protokol_check" CHECK (protokol::text = ANY (ARRAY['HTTP'::character varying, 'WS'::character varying, 'MQTT'::character varying]::text[]))
Foreign-key constraints:
    "tb_detection_log_scan_session_id_fkey" FOREIGN KEY (scan_session_id) REFERENCES tb_scan_session(id) ON DELETE SET NULL
Referenced by:
    TABLE "tb_alert" CONSTRAINT "tb_alert_detection_log_id_fkey" FOREIGN KEY (detection_log_id) REFERENCES tb_detection_log(id) ON DELETE SET NULL
    TABLE "tb_detection_detail" CONSTRAINT "tb_detection_detail_detection_log_id_fkey" FOREIGN KEY (detection_log_id) REFERENCES tb_detection_log(id) ON DELETE CASCADE
```

Gambar 4.13 Spesifikasi Atribut dan Tipe Data Tabel tb_detection_log

Tabel 4.6 Spesifikasi Atribut Tabel tb_detection_log

Nama Kolom	Tipe Data	Keterangan
id	BIGSERIAL	Primary Key, pengidentifikasi unik baris data.
protokol	VARCHAR(10)	Menyimpan jenis protokol yang digunakan (HTTP, WS, atau MQTT).
waktu_kirim	TIMESTAMPTZ	Waktu paket dikirim oleh Raspberry Pi (timestamp).
waktu_terima	TIMESTAMPTZ	Waktu paket diterima oleh server backend Node.js.
latency_ms	REAL	Nilai End-to-End Latency (dihitung otomatis dari selisih waktu terima dan waktu kirim).
image_path	VARCHAR(500)	Lokasi penyimpanan gambar Base64 yang telah dikonversi menjadi berkas fisik di server.
total_detections	INT	Jumlah total objek hama yang terdeteksi dalam satu gambar.
is_empty_detection	BOOLEAN	Penanda apakah gambar yang dikirim memiliki deteksi hama atau sekadar pemindaian kosong.

metadata	JSONB	Data tambahan terstruktur mengenai transmisi.
created_at	TIMESTAMPTZ	Waktu baris data dibuat di database.
scan_session_id	BIGINT	Foreign Key yang menghubungkan log ini dengan ID sesi pemindaian di tb_scan_session.

Gambar 4.13 dan Tabel 4.y menguraikan struktur tabel tb_detection_log yang merekam seluruh aktivitas transmisi data. Selain mencatat presisi waktu untuk perhitungan latensi, tabel ini menyimpan rincian hasil inferensi seperti lokasi gambar dan jumlah hama. Atribut is_empty_detection digunakan untuk membedakan deteksi hama dari pemindaian kosong, mengingat alat terus mengirimkan data secara kontinu selama bergerak. Seluruh log ini kemudian diikat melalui kolom scan_session_id agar setiap pengiriman data dapat dilacak kembali ke siklus pergerakan kamera yang bersesuaian.

4.3.2. Implementasi *Backend* dan Broker

Logika aplikasi, orkestrasi jalur komunikasi, dan perekaman telemetri pada platform ini diimplementasikan menggunakan *runtime* Node.js, sementara sistem distribusi *message* dikelola oleh broker Eclipse Mosquitto. Arsitektur perangkat lunak dibangun menggunakan pola *layered architecture* untuk memisahkan logika rute jaringan, logika bisnis, dan akses *database*.

```
backend-app/
├── Dockerfile
├── package.json
└── src/
    ├── config/
    │   ├── database.js          # Konfigurasi koneksi Pool PostgreSQL
    │   ├── env.js               # Parser environment variables
    │   └── mqtt.js              # Factory untuk koneksi MQTT Client & daftar topik
    │
    ├── controllers/
    │   ├── alertController.js   # Menangani logika respons alert peringatan
    │   ├── authController.js    # Autentikasi dashboard (JWT)
    │   ├── detectionController.js # Entry point deteksi dari semua protokol
    │   ├── dssController.js     # Menangani Decision Support System (Sistem Pakar)
    │   └── telemetryController.js
    │
    ├── db/
    │   ├── schema.sql           # Skema tabel database (termasuk latency_ms)
    │   └── seed.sql
    │
    ├── gateways/                # Layer Routing Protocol-Specific
    │   ├── httpGateway.js       # Router Express.js (REST API Endpoint)
    │   ├── mqttGateway.js       # Router MQTT (Menerima pesan dari topik)
    │   └── wsGateway.js         # Router WebSocket (Action-based router)
    │
    ├── middleware/
    │   └── authMiddleware.js    # Pengecekan API Key & JWT
    │
    ├── repositories/            # Layer Akses Database (Query SQL Langsung)
    │   ├── alertRepo.js
    │   ├── dashboardRepo.js
    │   ├── detectionRepo.js     # Query insert log deteksi & perhitungan QoS
    │   └── pestRepo.js
    │
    ├── server-http.js           # Entry point aplikasi HTTP Server
    ├── server-mqtt.js           # Entry point aplikasi MQTT Subscriber
    ├── server-ws.js             # Entry point aplikasi WebSocket Server
    │
    ├── services/                # Layer Business Logic
    │   ├── dedupService.js      # Algoritma Deduplikasi Centroid Euclidean
    │   ├── detectionService.js  # Pipeline validasi, dedup, dan alert
    │   └── sessionManager.js    # State machine sesi kamera
    │
    └── utils/
        ├── imageProcessor.js    # Penyimpanan Base64 menjadi file statis JPEG
        ├── logger.js            # Format log console
        └── timestampHelper.js   # Kalkulasi waktu
```

Gambar 4.14 Struktur Direktori Arsitektur *Backend* Node.js

Sebagaimana direpresentasikan pada Gambar 4.14, *source code* dibagi menjadi beberapa lapisan (*layer*). Direktori *gateways* bertugas mengisolasi *entry point* untuk masing-masing protokol (HTTP, WebSocket, dan MQTT). Lapisan

services menangani logika bisnis sistem pendukung keputusan, sedangkan lapisan *repositories* berinteraksi langsung dengan kueri PostgreSQL.

A. Implementasi Broker Pesan (Eclipse Mosquitto)

Pada layanan Mosquitto, implementasi konfigurasi difokuskan pada pembukaan jalur asinkron ganda untuk melayani dua jenis *Client* yang berbeda. Konfigurasi ini didefinisikan secara eksplisit di dalam berkas mosquitto.conf.

```
# ── Persistentce ──────────────────────────────────────
persistentce true
persistentce_location /mosquitto/data/

# ── Logging ───────────────────────────────────────────
log_dest file /mosquitto/log/mosquitto.log
log_type all

# ── Listener 1: Standard MQTT TCP (port 1883) ────────
# Digunakan oleh Backend Node.js dan Edge Device (Raspberry Pi)
listener 1883
protocol mqtt
allow_anonymous true

# ── Listener 2: MQTT over WebSocket (port 9001) ──────
# Digunakan oleh aplikasi React Frontend (Dashboard Live) di
browser
listener 9001
protocol websockets
allow_anonymous true
```

Gambar 4.15 Konfigurasi *Listener* pada Eclipse Mosquitto

Berdasarkan Gambar 4.15, broker mengonfigurasi port 1883 untuk menerima aliran paket MQTT TCP *native* dari *edge device* di perkebunan (Raspberry Pi) dan layanan *backend* Node.js. Selain itu, port 9001 dikonfigurasi untuk mengekspos protokol MQTT berbasis WebSocket (MQTT *over* WebSockets). *Exposure* port 9001 ini diwajibkan karena *Client* antarmuka pengguna pada *browser* web tidak memiliki dukungan bawaan untuk membuka soket TCP murni.

Parameter allow_anonymous *true* diaktifkan secara sengaja pada kedua porta tersebut. Secara arsitektural, hal ini tidak menimbulkan celah keamanan karena broker Mosquitto diisolasi secara tertutup di dalam *Docker Bridge Network*. Keamanan sistem tidak ditangani di tingkat broker, tetapi ditugaskan ke lapisan aplikasi Node.js (melalui validasi API Key pada setiap *payload* yang masuk), sehingga berhasil menghilangkan *overhead* dari proses otentikasi ganda saat perangkat IoT mentransmisikan gambar berukuran besar.

B. Implementasi *Endpoint Gateways* Multi-Protokol

Node.js mengimplementasikan *gateway* yang terpisah untuk masing-masing protokol transmisi.

```
// src/server-http.js (Inisiasi Express)
import express from 'express';
import httpGateway from './gateways/httpGateway.js';

const app = express();
app.use(express.json({ limit: '10mb' }));  // Menerima payload
gambar besar
app.use(httpGateway);                      // Menyambungkan
semua rute API
const server = app.listen(3000, () => {
  console.log(`[HTTP] Server listening on port 3000`);
});

// src/gateways/httpGateway.js (Definisi Endpoint HTTP Deteksi)
import { Router } from 'express';
import { handleDetection } from
'../controllers/detectionController.js';
import { apiKeyAuth } from '../middleware/authMiddleware.js';

const router = Router();
```

Gambar 4.16 Implementasi *Gateway* HTTP dan *Client* MQTT pada Node.js

Pada implementasi HTTP (Gambar 4.16), *framework* Express.js dikonfigurasi menggunakan express.json({ limit: '10mb' }) untuk menaikkan ambang batas ukuran *request*. Hal ini penting untuk mencegah penolakan paket ketika *edge device* mengirimkan *payload* matriks gambar Base64 yang

berukuran besar. Permintaan HTTP POST diterima pada rute /api/detect yang dilindungi oleh *middleware* apiKeyAuth.

```
// src/config/mqtt.js (Membuat Instansiasi Client)
import mqtt from 'mqtt';

export const TOPICS = Object.freeze({
  DETECT_UP:       'lokatani/detect/up',
  DETECT_DOWN:     'lokatani/detect/down',
  SESSION_START:   'lokatani/session/start',
  // ... topik lainnya
});

export function createMqttClient() {
  const client = mqtt.connect('mqtt://mosquitto:1883', {
    clientId: 'lokatani-backend',
    clean: true,
  });
  return client;
}
```

Gambar 4.17. Pembuatan Instans MQTT Client

Di sisi lain, Node.js juga diimplementasikan sebagai MQTT *Client* yang *subscribe* ke topik lokatani/detect/up, seperti yang ditampilkan pada gambar 4.17. Ketika Raspberry Pi melakukan *publish* gambar ke topik tersebut di Mosquitto, fungsi mqttGateway.js di dalam Node.js akan mencegat pesan tersebut, meneruskannya ke siklus pemrosesan deteksi, dan *publish* laporan mitigasi hama ke topik lokatani/detect/down menuju *Client* antarmuka pengguna.

C. Komputasi Latensi QoS

Perekaman parameter QoS, khususnya latensi *End-to-End*, ditangani di dalam lapisan akses data pada berkas detectionRepo.js.

```
// src/repositories/detectionRepo.js

import { query } from '../config/Database.js';

/**
 * Menyimpan entry payload deteksi ke tabel tb_detection_log.
```

```
 * PostgreSQL secara internal mengkalkulasikan selisih `waktu_terima` dan
`waktu_kirim`
 * ke dalam kolom `Latency_ms` yang kemudian dikembalikan menggunakan
RETURNING.
 */
export async function insertLog({
  protokol,
  waktuKirim,
  waktuTerima,
  imagePath,
  totalDetections,
  metadata,
  scanSessionId
}) {
  const sql = `
    INSERT INTO tb_detection_log
      (protokol, waktu_kirim, waktu_terima, image_path, total_detections,
metadata, scan_session_id)
    VALUES ($1, $2, $3, $4, $5, $6, $7)
    RETURNING id, Latency_ms, created_at
  `;

  const { rows } = await query(sql, [
    protokol,
    waktuKirim,     // Timestamp ISO dari Raspberry Pi
    waktuTerima,    // Timestamp ISO Server (Node.js) saat request masuk
    imagePath,
    totalDetections,
    metadata ? JSON.stringify(metadata) : '{}',
    scanSessionId || null,
  ]);

  return rows[0]; // Mereturn objek { id, Latency_ms, created_at } untuk
broadcast frontend
}
```

Gambar 4.18 Logika Penyisipan Telemetri dan Perekaman Latensi pada detectionRepo.js

Gambar 4.18 mendemonstrasikan metode penyimpanan log telemetri yang dieksekusi melalui satu kueri INSERT tunggal. Sistem menyandingkan stempel waktu saat data ditangkap kamera (waktuKirim) dengan waktu aktual data tiba di *Server* (waktuTerima).

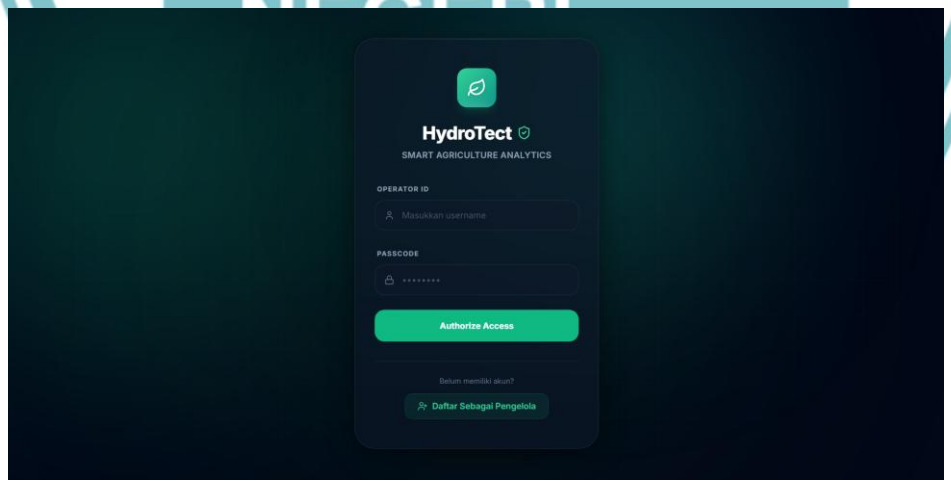
Alih-alih melakukan operasi pengurangan waktu di dalam Node.js, komputasi latensi didelegasikan sepenuhnya ke mesin *database* PostgreSQL. PostgreSQL secara otomatis mengkalkulasi selisih waktu tersebut ke dalam kolom *Latency*_ms (melalui pemicu *database* internal), yang kemudian langsung dikembalikan ke lapisan aplikasi menggunakan klausa RETURNING id, *Latency*_ms. Pendekatan arsitektural ini secara efektif membebaskan antrean komputasi (*Event Loop*) pada *thread* Node.js dari operasi aritmatika yang repetitif, sehingga stabilitas dan skalabilitas *Server* dalam menerima burst traffic citra tetap terjaga.

4.3.3. Implementasi *Frontend*

Antarmuka pengguna diimplementasikan menggunakan pustaka React.js yang beroperasi sebagai *Single Page Application* (SPA). Aplikasi *Client* yang telah melalui tahap kompilasi akhir (statik) disajikan oleh *Server* Nginx. Implementasi antarmuka dipetakan ke dalam beberapa modul halaman fungsional untuk mengakomodasi kebutuhan pemantauan dan analisis, yaitu:

a) Halaman Login

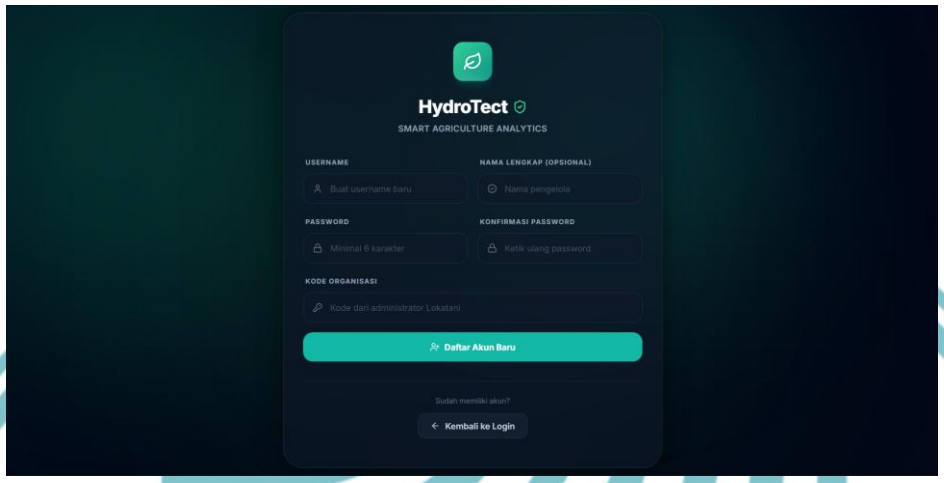
Gambar 4.19 menampilkan halaman login. Halaman ini merupakan halaman yang pertama kali muncul saat *user* mengakses web, dan dapat menginputkan ID dan password untuk login.



Gambar 4.19 Implementasi Halaman Login

Pada halaman login, pengguna baru dapat mendaftarkan akun dengan klik *button* "Daftar Sebagai Pengelola". Pada form registrasi ini,

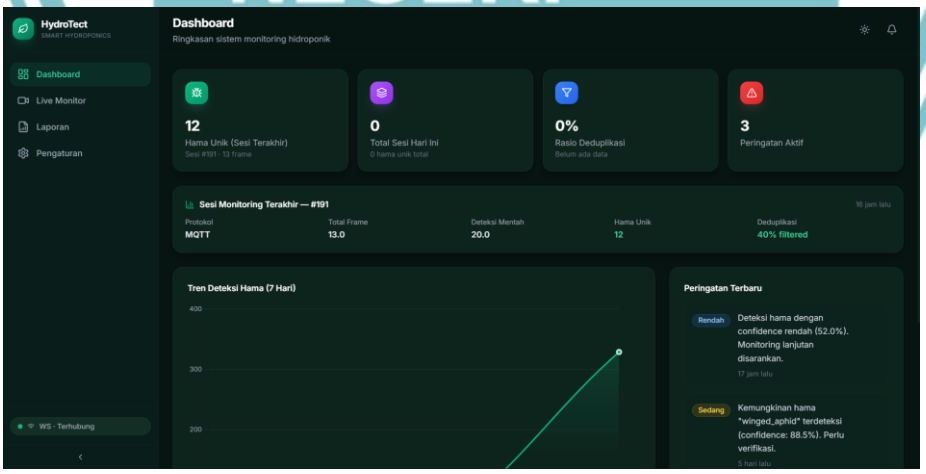
pengguna dapat memasukkan data seperti *Username*, nama lengkap, Password, konfirmasi password, dan kode organisasi seperti terlihat pada gambar 4.20 di bawah ini.



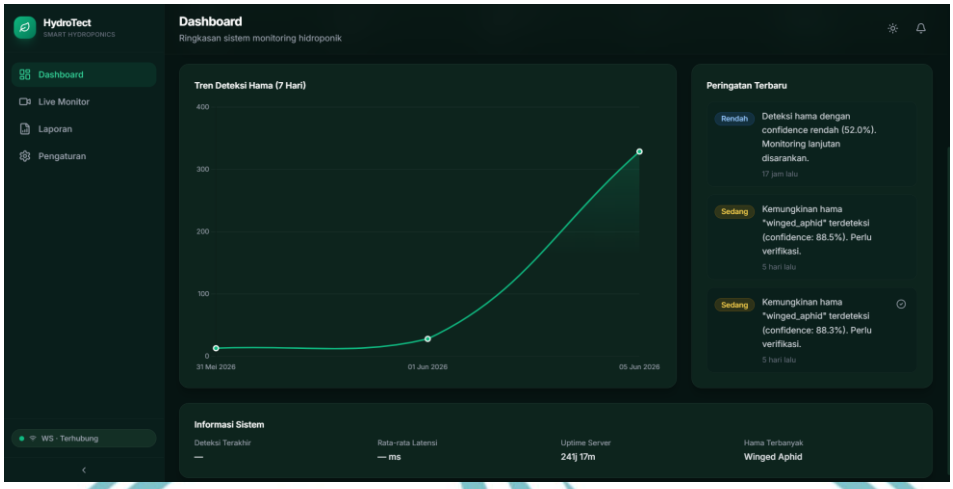
Gambar 4.20 Implementasi Halaman Registrasi Pengguna

b) Halaman *Dashboard* (Dashboard)

Gambar 4.21 dan 4.22. menampilkan Halaman *Dashboard*. Halaman ini diimplementasikan untuk memberikan agregasi data tingkat tinggi, memuat indikator statistik deteksi hama kumulatif, serta menayangkan grafik deret waktu tren hama selama 7 hari operasional terakhir untuk mempermudah identifikasi anomali oleh administrator.



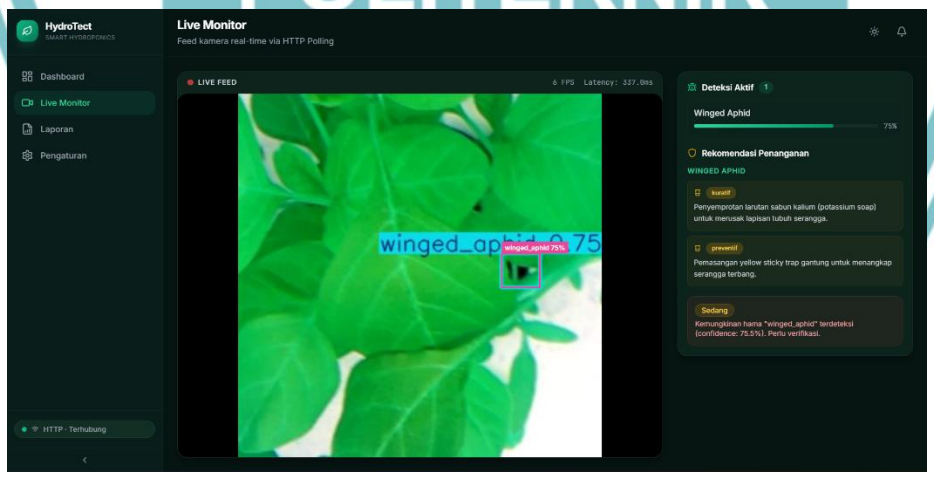
Gambar 4.21 Implementasi Halaman *Dashboard* (1)



Gambar 4.22 Implementasi Halaman *Dashboard* (2)

c) Halaman Live Monitor

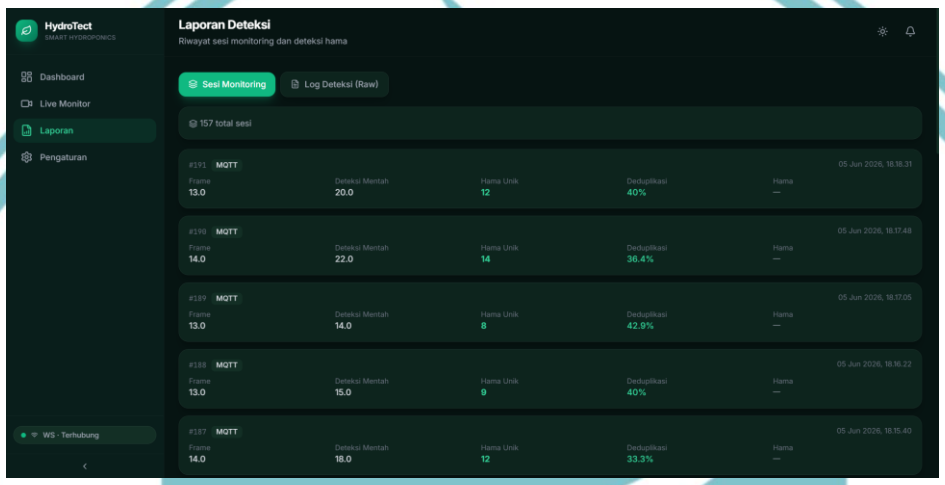
Gambar 4.23 memperlihatkan Halaman *Live Monitor*. Implementasi halaman ini difokuskan pada perenderan elemen matriks gambar secara diskrit yang diterima dari sistem perpesanan asinkron (WebSockets/MQTT). Pada halaman ini juga tersemat komponen Protocol Switcher untuk memberikan kemampuan uji alih protokol secara langsung pada fase evaluasi kinerja metrik jaringan.



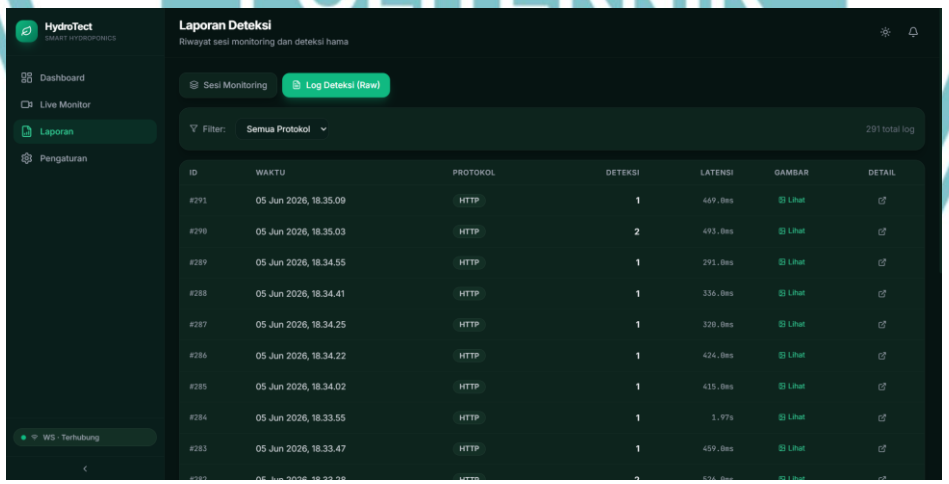
Gambar 4.23 Implementasi Halaman *Live Monitor* dengan *Protocol Switcher*

d) Halaman Laporan Sesi dan Log Deteksi

Gambar 4.24 dan 4.25 menyajikan struktur data pada Halaman Laporan. Implementasi dipecah menjadi tabel Sesi Monitoring dan tabel Log Deteksi mentah. Tabel ini bertugas memvisualisasikan parameter kunci dari setiap bingkai (*frame*) gambar, mulai dari ukuran muatan (*payload size*), stempel waktu absolut, hingga nilai fluktuasi latensi jaringan (delay).



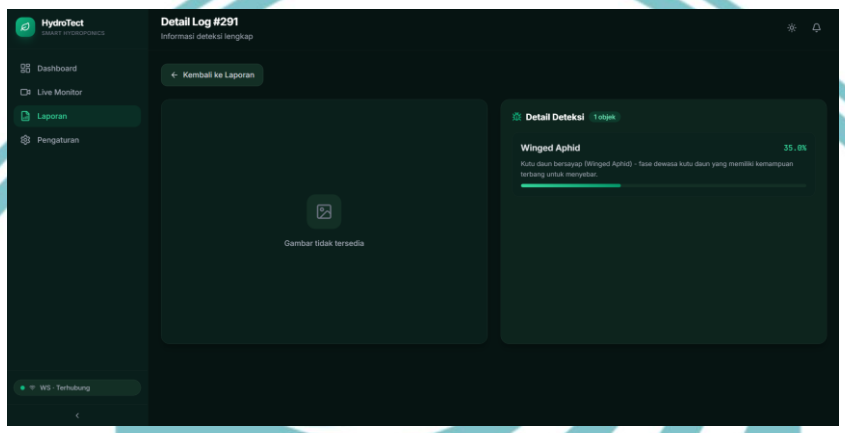
Gambar 4.24 Implementasi Halaman Laporan Log Telemetri



Gambar 4.25 Implementasi Halaman Laporan Log Telemetri

e) Halaman Detail Log dan Mitigasi

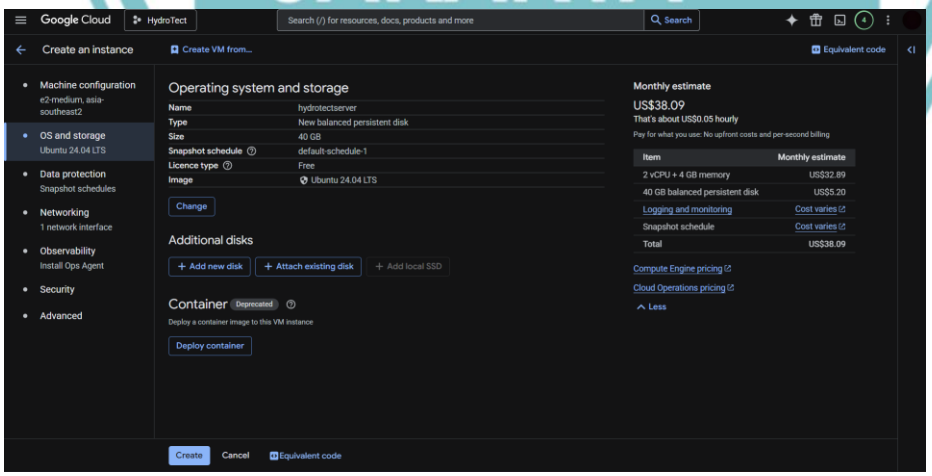
Gambar 4.26 memvisualisasikan detail log dari Halaman Laporan. Halaman Detail Log diimplementasikan untuk memfasilitasi audit visual melalui penyajian citra kamera fisik, representasi *bounding box*, skor *confidence*, beserta rekomendasi penanganan Sistem Pendukung Keputusan (DSS) yang berelasi dengan hama tersebut.



Gambar 4.26 Implementasi Halaman Detail Log Visual

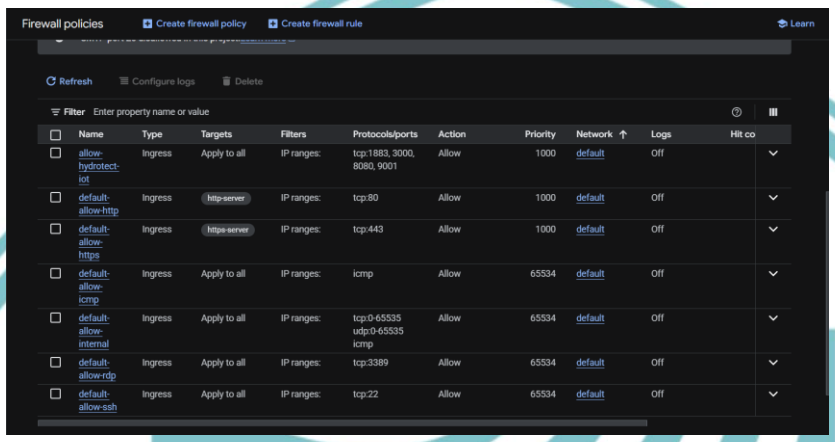
4.3.4. Implementasi Infrastruktur Cloud

Keseluruhan kode dan *microservices* yang telah diimplementasikan pada tahap sebelumnya di-*deploy* ke dalam lingkungan produksi VM. Google Cloud Platform (GCP) digunakan dengan mengalokasikan spesifikasi *Virtual Machine* instans e2-medium berbasis sistem operasi Ubuntu Server 22.04 LTS.



Gambar 4.27. Overview Pembuatan Instans VM GCP

Agar *Server* dapat berkomunikasi secara selektif dengan *Edge* Device dan *Client* web, implementasi keamanan diatur menggunakan kapabilitas VPC Firewall Rules. Gambar 4.28 mendemonstrasikan panel konfigurasi GCP *Ingress Traffic* yang menetapkan pembukaan porta spesifik (Port 80, 3000, 8080, 1883, dan 9001) untuk jalur masuknya data dari *edge device*.



Gambar 4.28. Hasil Konfigurasi *Firewall Rule* VM

4.4.Pengujian Sistem

Tahap pengujian merupakan fase validasi empiris untuk mengukur tingkat keberhasilan sistem dalam memenuhi analisis kebutuhan yang telah ditetapkan pada subbab sebelumnya. Pengujian sistem ini dilaksanakan di area perkebunan hidroponik untuk merepresentasikan kondisi operasional nyata.

4.4.1. Deskripsi Pengujian

Pengujian pada platform pemantauan hidroponik ini merupakan fase validasi empiris untuk mengukur tingkat keberhasilan sistem dalam memenuhi spesifikasi kebutuhan yang telah ditetapkan. Agar hasil evaluasi komprehensif dan terukur, pengujian diklasifikasikan ke dalam tiga domain utama:

A. Pengujian Fungsionalitas dan Penerimaan Pengguna

Pengujian ini bertujuan untuk memvalidasi kelayakan operasional aplikasi dari dua sudut pandang yang berbeda, yakni secara teknis infrastruktur dan secara praktis oleh pengguna akhir. Pengujian ini dibagi menjadi dua instrumen:

1) Pengujian *Blackbox*

Berfokus pada validasi teknis aliran data, memastikan seluruh rute aplikasi (API), logika antarmuka pengguna berbasis SPA (*Single Page Application*), dan fungsi Sistem Pendukung Keputusan (DSS) dapat memproses masukan serta menghasilkan keluaran yang akurat sesuai rancangan, tanpa meninjau struktur kode internal.

2) *User Acceptance Test* (UAT)

Pengujian penerimaan yang dieksekusi secara langsung oleh pihak pengelola kebun hidroponik Lokatani. Pengujian ini menggunakan kuesioner terstruktur berbasis skenario tugas untuk mengevaluasi tingkat kepuasan, kemudahan penggunaan (*usability*), stabilitas akses, dan relevansi platform terhadap kebutuhan operasional nyata di lapangan.

B. Pengujian Kinerja QoS Antarprotokol

Pengujian ini bertujuan untuk melakukan komparasi analitis terhadap efisiensi lapisan aplikasi dari protokol HTTP, WebSocket, dan MQTT. Evaluasi merujuk pada standar TIPHON yang mencakup empat parameter metrik jaringan, yakni *Throughput*, *Packet Loss*, *End-to-End Latency*, dan *Jitter*. Pengujian dieksekusi menggunakan skenario transmisi data gambar visual (Base64) berukuran besar secara kontinu. Untuk menjaga variabel kontrol agar perbandingan bandwidth lebar bersifat setara dan adil, perangkat *Edge* dikonfigurasi untuk mengirim data secara diskrit murni pada interval statis 3 detik selama total durasi 15 menit per protokol (dibagi menjadi 3 sesi masing-masing 5 menit).

C. Pengujian Keandalan Operasional Sistem

Pengujian ini dirancang untuk membuktikan ketangguhan dan daya tahan arsitektur sistem perangkat lunak terdistribusi saat dihadapkan pada skenario operasional jangka panjang maupun anomali pemutusan jaringan. Pengujian dieksekusi melalui tiga skenario:

1) Uji Ketahanan Operasi Sistem

Mengevaluasi stabilitas pemrosesan memori *server* (*Server* GCP) dan ketahanan suhu perangkat *Edge* dengan membiarkan sistem beroperasi secara terus-menerus tanpa henti selama durasi 6 jam menggunakan protokol MQTT. Skenario ini bertujuan untuk membuktikan ketiadaan kebocoran memori (*memory leak*) pada soket *persistent* dan kelangsungan penggunaan CPU di bawah beban kerja konstan.

2) Uji *Recovery* Jaringan Perangkat *Edge* (*Auto-Reconnect*)

Membuktikan keandalan skrip jaringan pada *edge device* dengan melakukan simulasi interupsi koneksi internet secara fisik. Sistem dituntut untuk mampu menahan kehilangan jaringan tanpa mengalami kelumpuhan program dan secara otonom menyambung kembali ke *server* saat sinyal jaringan seluler dipulihkan.

3) Uji Ketahanan *Microservices*

Menguji ketangguhan arsitektur isolasi kontainer di *cloud*. Skenario dilakukan dengan mematikan paksa layanan dependensi secara spesifik di tengah proses transmisi data, yaitu mematikan kontainer broker MQTT (Eclipse Mosquitto) dan mematikan kontainer *database* (PostgreSQL). Pengujian ini memvalidasi kapabilitas *self-healing*, kemampuan *server* utama (Node.js) dalam mencegah *cascading failure*, dan keandalan perangkat *Edge* dalam merespons penolakan sistem tanpa merusak integritas arsitektur.

4.4.2. Prosedur Pengujian

Pengujian sistem dilaksanakan melalui prosedur operasional baku yang sistematis dan terisolasi untuk memastikan validitas dan konsistensi data empiris. Pengujian dipetakan ke dalam tiga domain utama dengan persiapan dan eksekusi sebagai berikut:

A. Lingkungan dan Instrumen Pengujian

Sebelum prosedur eksekusi dilakukan, lingkungan fisik dan instrumen ukur ditetapkan sebagai variabel kontrol:

a) Pengujian kinerja jaringan dilaksanakan di area *greenhouse* hidroponik Lokatani dalam kondisi cuaca cerah untuk meniadakan atenuasi sinyal akibat cuaca.
b) Perangkat *Edge* (Raspberry Pi 4) dihubungkan ke internet menggunakan jaringan WifI 5G yang diisolasi murni untuk keperluan telemetri IoT (tidak digunakan untuk aplikasi pihak ketiga untuk mencegah *bufferbloat*).
c) Sinkronisasi waktu (*Network Time Protocol* / NTP) pada *server* GCP dan Raspberry Pi diselaraskan ke time.google.com dengan toleransi *clock offset* di bawah ±50 ms untuk menjamin validitas kalkulasi stempel waktu (*timestamp*).

B. Prosedur Pengujian Fungsionalitas dan Penerimaan Pengguna

Domain ini memvalidasi kelayakan teknis aplikasi dan tingkat penerimaan operasional oleh pengguna akhir (UAT).

1) Prosedur Pengujian *Blackbox*:
   - Validasi Otentikasi: Penguji mengakses antarmuka SPA React melalui *browser* web dan mengeksekusi proses masuk (login) untuk mendapatkan JSON Web Token (JWT).
   - Validasi Penerimaan Data & UI: Perangkat *Edge* (Raspberry Pi) diaktifkan secara fisik untuk melakukan inferensi dan mentransmisikan payload JSON (berisi citra Base64, koordinat *bounding box*, dan tingkat *confidence*) ke titik akhir *backend*. Penguji mengobservasi pembaruan pada elemen HTML5 Canvas di Halaman *Live Monitor* secara asinkron.
   - Validasi Rute Protokol: Penguji mengaktifkan *Protocol Switcher* untuk mengganti rute data antara HTTP, WebSocket, dan MQTT secara bergantian tanpa memuat ulang (*reload*) halaman.
   - Validasi DSS & Laporan: Penguji memvalidasi kemunculan instruksi mitigasi hama dari Sistem Pendukung Keputusan (DSS) pada panel informasi, lalu menavigasi Halaman Laporan untuk memastikan log

deteksi dan stempel waktu terekam secara *persistent* di pangkalan data.

- Validasi Konfigurasi Dinamis: Penguji memodifikasi batas ambang deteksi (*Minimum Confidence Threshold*) pada Halaman Pengaturan untuk memastikan filter penyaringan objek pada *backend* merespons perubahan tersebut secara instan.

2) Prosedur *User Acceptance Test* (UAT):
   - Pengujian dilakukan oleh representasi pengelola kebun Lokatani menggunakan kuesioner terstruktur (*Google Form*).
   - Pengguna diinstruksikan menyelesaikan 6 tugas eksplorasi (Task-Based) yang mencakup login, akses Live Monitor, observasi pemicu DSS, penelusuran riwayat log, modifikasi sensitivitas, dan responsibilitas aplikasi.
   - Setelah eksekusi tugas, pengguna mengevaluasi sistem melalui 7 indikator kualitas berbasis skala Likert (1-5) yang mengukur kemudahan UI/UX, stabilitas performa, keterbacaan data historis, dan efektivitas DSS dalam konteks operasional kebun.

C. Prosedur Pengujian Kinerja QoS Antarprotokol

Pengujian ini bertujuan untuk mengekstrak metrik performa jaringan berdasarkan standardisasi TIPHON. Eksekusi dilakukan menggunakan interval pengiriman statis 3 detik selama total durasi 15 menit per protokol (dibagi menjadi 3 sesi). Ekstraksi data tingkat transport dan *application* dilakukan dengan prosedur berikut:

1) Penguji menggunakan fitur *SSH remote capture* pada perangkat lunak Wireshark untuk merekam lalu lintas jaringan secara langsung dari antarmuka *Virtual Machine* GCP. Penangkapan paket difilter secara spesifik berdasarkan *port* masing-masing protokol (3000 untuk HTTP, 8080 untuk WebSocket, dan 1883 untuk MQTT).
2) Skrip transmisi pada Raspberry Pi dijalankan untuk mengirimkan *payload* matriks gambar Base64 menuju *server* selama batas waktu 5 menit per sesi.

3) Data *End-to-End Latency* diekstraksi dari tabel tb_detection_log pada *database* PostgreSQL ke dalam format CSV menggunakan kueri SQL melalui terminal GCP.
4) Untuk perhitungan *Throughput*, filter ip.src == [IP_RASPBERRY_PI] diterapkan pada Wireshark guna mengisolasi arah unggahan (*upload*). Kalkulasi dilakukan menggunakan alat *I/O Graphs* pada interval 1 detik, lalu diekspor ke format CSV untuk dikonversi menjadi rata-rata keseluruhan dalam satuan Kbps.
5) Pemindaian *Packet Loss* diisolasi menggunakan filter ip.src == [IP_RASPBERRY_PI] && (tcp.analysis.retransmission || tcp.analysis.fast_retransmission). Parameter ini digunakan untuk memastikan tidak terjadi perhitungan ganda akibat mekanisme perbaikan otomatis TCP. Persentase dihitung dengan membagi jumlah paket retransmisi tersebut dengan total keseluruhan paket yang ditransmisikan.
6) Variasi kedatangan paket (*Jitter*) ditarik menggunakan filter ip.src == [IP_RASPBERRY_PI] && tcp.len > 0 untuk mengambil paket yang memiliki muatan. Data diekspor ke CSV untuk dilakukan *data cleansing*. Baris paket dengan *time delta* 0 (proses rekonstruksi TCP) dan *time delta* di atas 1,0 detik (jeda internal aplikasi) dihapus. Nilai *Jitter* akhir diperoleh melalui kalkulasi Standar Deviasi Sampel (STDEV.S) pada perangkat lunak *spreadsheet*.

D. Prosedur Pengujian Keandalan Operasional Sistem

Pengujian ini dirancang untuk membuktikan daya tahan arsitektur sistem menghadapi anomali jaringan dan kegagalan layanan kontainer, dieksekusi melalui tiga skenario:

1) Skenario 1: Uji Ketahanan Operasi Sistem (*Soak Testing*)
   - Skenario ini menguji stabilitas soket *persistent* dan manajemen memori. Sistem dikonfigurasi menggunakan protokol MQTT dan dijalankan secara tanpa henti (*non-stop*) selama durasi 6 jam melalui multiplexer terminal (tmux).
   - Perekaman sumber daya CPU dan RAM pada Server GCP diotomasi setiap 5 detik menggunakan bash script (server_logger.sh), sementara

metrik suhu dan utilitas Raspberry Pi direkam melalui *background* skrip Python.

2) Skenario 2: Uji *Recovery* Jaringan Perangkat *Edge* (*Auto-Reconnect*)
   - Skenario ini menguji *Exception Handling* pada sistem IoT tingkat lapangan. Saat transmisi beroperasi, jaringan internet diputus secara fisik.
   - Penguji memvalidasi keluaran log terminal Raspberry Pi untuk memastikan utas program menolak terminasi paksa (*Crash*) dan masuk ke fase *Recovery*. Saat jaringan diaktifkan kembali, sistem harus berhasil merestorasi *handshake* soket TCP dan melanjutkan transmisi matriks tanpa perintah *restart* manual.
3) Skenario 3: Uji Resiliensi *Microservices* (Kegagalan *Backend*)
   - Saat komunikasi berjalan via MQTT, kontainer *broker* dimatikan secara paksa menggunakan perintah docker stop mosquitto. Node.js diuji agar tidak mengalami *crash*, sementara perangkat *edge* divalidasi kemampuannya dalam menangani penolakan transmisi.
   - Pada pengujian protokol HTTP, kontainer *database* dimatikan secara paksa (docker stop postgres). Node.js diuji untuk menangkap *error* koneksi dan secara proaktif mengembalikan status HTTP 500 kepada *client*. Skrip pada Raspberry Pi divalidasi keberhasilannya dalam mengabaikan respons galat tersebut tanpa membekukan memori antrean kamera, lalu secara mandiri mencoba mengirim data kembali pada siklus 3 detik berikutnya.
   - Durasi *recovery time* pada kedua pengujian di atas diukur secara otomatis oleh sistem Node.js. Saat terputus, sistem mencatat *timestamp* kegagalan. Ketika kontainer pangkalan layanan tersebut dihidupkan ulang, kode sistem akan menghitung selisih waktu secara matematis dan langsung menampilkannya pada log terminal dalam satuan milidetik, sebagai bukti empiris kecepatan sistem memulihkan aliran data secara otonom.

4.4.3. Data Hasil Pengujian

Subbab ini menyajikan agregasi data mentah yang diperoleh selama prosedur pengujian fungsionalitas aplikasi dan kinerja transmisi jaringan.

A. Hasil Pengujian Fungsionalitas dan Penerimaan Pengguna

Hasil pengujian fungsionalitas dan penerimaan pengguna mendokumentasikan validasi aplikasi dari dua perspektif yang saling melengkapi. Perspektif pertama merupakan pengujian teknis infrastruktur secara tertutup (*Blackbox* Testing) untuk memastikan ketiadaan cacat logika pada rute data. Perspektif kedua merupakan pengujian penerimaan pengguna akhir (User Acceptance Test / UAT) yang dieksekusi secara langsung oleh pihak pengelola kebun untuk mengevaluasi stabilitas, kegunaan antarmuka, dan relevansi sistem terhadap skenario operasional di lapangan.

1) Hasil Pengujian *Blackbox*

Hasil validasi fungsional terhadap interaksi antara antarmuka *frontend* dan lapisan *backend* didokumentasikan pada Tabel 4.7.

Tabel 4.7. Hasil Pengujian Fungsionalitas (*Blackbox*)

No	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan	Hasil
1.	Login dengan kredensial valid	1. Memasukkan ID Operator dan Passcode yang valid. 2. Menekan tombol otorisasi akses.	Sistem melakukan validasi, menyimpan token JWT dengan aman, dan mengarahkan pengguna ke halaman *Dashboard* Utama.	Berhasil
2.	Login dengan kredensial tidak valid	1. Memasukkan kredensial (ID/Passcode) yang salah atau tidak terdaftar.	Sistem memblokir akses dan menampilkan peringatan (kredensial tidak valid) melalui	Berhasil

		2. Menekan tombol otorisasi akses.	komponen toast/alert pada antarmuka.	
3.	Perlindungan rute privat tanpa JWT	Mengakses tautan rute privat secara langsung (misal: /live atau /logs) melalui kolom URL *browser*.	Fungsi interceptor sistem mendeteksi ketiadaan token, mencegat permintaan (HTTP 401), dan memaksa pengalihan (redirect) ke halaman Login.	Berhasil
4.	Proses terminasi sesi (*Logout*)	Menekan tombol "Keluar" atau "Logout" pada panel navigasi.	Sistem menghilangkan token dari penyimpanan lokal *browser* dan mengalihkan pengguna kembali ke halaman Login.	Berhasil
5.	Inisiasi sesi pemindaian (*Start Session*)	Perangkat *edge* mengirimkan sinyal awal (start_session) yang dilampiri API Key statis milik mesin.	*Backend* mengotentikasi perangkat, merekam sesi pemindaian baru berstatus *active* di *database*, dan mengembalikan session_id.	Berhasil
6.	Transmisi interval deteksi (*Burst Frame*)	Perangkat *edge* mengirim payload JSON berisi gambar Base64, daftar koordinat (bounding box), tingkat *confidence*, dan scan_session_id.	*Backend* memvalidasi scan_session_id, mengeliminasi objek di bawah ambang batas (*filter confidence*), menyimpannya, lalu menyiarkan frame ke *browser*.	Berhasil

7.	Terminasi sesi pemindaian (*End Session*)	Perangkat *edge* mengirimkan sinyal end_session beserta session_id.	*Backend* menandai status sesi menjadi completed, menghasilkan kompilasi ringkasan hama, lalu membebaskan alokasi memori pemrosesan.	Berhasil
8.	Penolakan telemetri di luar sesi (*Strict State*)	Perangkat *edge* mencoba mendistribusikan payload deteksi tanpa menyertakan scan_session_id yang valid.	*Backend* secara paksa menolak paket transmisi tersebut dan memberikan respons galat HTTP 400/403 demi menjaga integrasi (*state compliance*).	Berhasil
9.	Render agregasi dan matriks data	Memuat halaman *Dashboard* Utama.	Sistem berhasil melakukan permintaan ke REST API untuk mengambil total akumulasi deteksi, status alerts, serta parameter lingkungan, lalu merendernya pada kartu matriks.	Berhasil
10.	Visualisasi grafik tren deteksi	Memeriksa komponen bagan deteksi.	*Client* berhasil memproses agregasi waktu dari JSON yang diterima dan memvisualisasikannya secara presisi ke dalam struktur bar/line chart.	Berhasil

11.	Penerimaan telemetri via HTTP *Polling*	1. Memilih opsi "HTTP" pada *Protocol Switcher* antarmuka. 2. Mengamati panel live monitor.	Modul antarmuka mengirimkan polling asinkron ke REST API untuk mengambil dan menggambar matriks bounding box beserta gambar Base64 dari rekam data terbaru.	Berhasil
12.	Penerimaan telemetri via WebSocket (WS)	Memilih opsi "WebSocket" pada *Protocol Switcher*.	*Client* membentuk koneksi soket dua arah. Modul langsung menampilkan render gambar, *bounding box*, dan daftar peringatan setiap kali menerima *event broadcast* secara instan.	Berhasil
13.	Penerimaan telemetri via MQTT	Memilih opsi "MQTT" pada *Protocol Switcher*.	*Client* terhubung (subscribe) melalui MQTT over WebSockets. Frame beserta bounding box berhasil di-render seketika setiap kali payload masuk di topik yang sesuai.	Berhasil
14.	Tampilan riwayat sesi (*Sessions*) dan detail (*Raw Logs*)	1. Mengakses halaman "Log" atau "Laporan". 2. Menggunakan navigasi daftar (tabel) dan fitur halaman (*pagination*).	Sistem menampilkan tabel riwayat sesi operasional (kompilasi hama terdeteksi) beserta tabel sub-log yang mengurutkan jejak perekaman	Berhasil

			berdasarkan waktu aktual (WIB).	
15.	Akses Modal Detail View dan Rekomendasi Penanganan	Mengklik tombol aksi "Detail" pada salah satu entri deteksi yang spesifik.	Sistem membuka overlay/modal berisi gambar asli yang diperbesar dengan bounding box, informasi tingkat keyakinan (akurasi), beserta langkah mitigasi (*Decision Support System*) khusus hama tersebut.	Berhasil
16.	Perubahan parameter deteksi (*Minimum Confidence Threshold*)	1. Mengakses panel Settings. 2. Menggeser parameter *Minimum Confidence Threshold* (misal: dari 0.50 menjadi 0.70). 3. Menekan tombol "Simpan".	Sistem mengeksekusi update konfigurasi. Pemindaian sesi selanjutnya otomatis memfilter dan mengabaikan objek hama yang memiliki nilai prediktif di bawah 0.70.	Berhasil

2) Hasil Pengujian UAT

Pengujian UAT melibatkan perwakilan pengguna dari Lokatani, yakni Pengelola Kebun. Pengujian ini difokuskan pada validasi fungsional *Task-Based* dan evaluasi kualitas perangkat lunak. Hasil eksekusi tugas fungsional yang dilakukan oleh responden dirangkum pada Tabel 4.8.

Tabel 4.8 Hasil Eksekusi Tugas Fungsional UAT

No	Skenario Tugas Fungsional (Task-Based)	Status Penyelesaian

1.	Autentikasi Pengguna: Masuk ke dalam dasbor menggunakan kredensial.	Berhasil
2.	Akses *Live Monitor*: Membaca aliran data gambar secara real-time.	Berhasil
3.	Validasi DSS: Memvalidasi kemunculan otomatis rekomendasi mitigasi.	Berhasil
4.	Penelusuran Riwayat: Mengakses log deteksi dan detail *database*.	Berhasil
5.	Modifikasi Parameter: Mengubah batas sensitivitas (*Confidence*) ke 0.70.	Berhasil
6.	Responsivitas Perangkat: Mengakses dasbor melalui ponsel pintar.	Berhasil

Berdasarkan Tabel 4.6, seluruh skenario tugas yang diberikan berhasil diselesaikan oleh pengguna dengan tingkat keberhasilan sebesar 100%. Fakta ini secara empiris memvalidasi bahwa arsitektur antarmuka dan perutean *backend* merespons instruksi pengguna (termasuk pada perangkat bergerak) tanpa mengalami galat fungsional (*functional defect*).

Setelah tugas fungsional diselesaikan, responden memberikan penilaian terhadap kualitas operasional sistem menggunakan skala Likert (1 hingga 5). Hasil evaluasi tersebut didokumentasikan pada Tabel 4.9.

Tabel 4.9 Hasil Evaluasi Kualitas Perangkat Lunak (Skala 1-5)

Parameter Evaluasi Kualitas (UI/UX & Performa)	Skor Penilaian
Kinerja Jaringan: Kecepatan pembaruan data tanpa tundaan yang mengganggu.	5 (Sangat Baik)
Stabilitas & Penanganan Galat: Sistem beroperasi tanpa *crash* atau pembekuan.	5 (Sangat Baik)
Efektivitas Visual: Bounding Box membantu mengenali posisi hama secara instan.	4 (Baik)

Implikasi Operasional: Sistem meningkatkan efisiensi waktu manajemen kebun.	4 (Baik)
Arsitektur Informasi Historis: Kemudahan interpretasi grafik tren hama.	4 (Baik)
Kemudahan Navigasi: Tampilan dasbor dan menu yang intuitif.	3 (Cukup)
Kegunaan Data DSS: Eksistensi panel mitigasi membantu pengambilan keputusan.	3 (Cukup)

Data evaluasi pada Tabel 4.7 menunjukkan dominasi skor tinggi (4 dan 5) pada parameter inti rekayasa perangkat lunak. Pengelola *greenhouse* memberikan skor maksimal (5) pada aspek "Kinerja Jaringan" dan "Stabilitas Sistem". Hal ini menjadi bukti kuat bahwa arsitektur *microservices* mampu menyajikan aliran gambar tanpa membebani memori *client browser*.

Validasi terhadap keandalan perangkat lunak ini juga diperkuat melalui catatan umpan balik kualitatif responden. Responden tidak melaporkan adanya *bug* perangkat lunak, melainkan memberikan saran perbaikan pada tingkat mekanik (perangkat keras), yakni *"Perlu adanya rotasi kamera 360"* dan *"Penyesuaian tinggi kamera"*. Umpan balik ini menegaskan bahwa fungsionalitas arsitektur TI dan jaringan pada platform ini telah dinyatakan matang, andal, dan siap dioperasikan.

B. Hasil Pengujian QoS

Pengumpulan data kinerja jaringan tingkat transport dan aplikasi dieksekusi di area *greenhouse* Lokatani dalam kondisi cuaca cerah menggunakan konektivitas seluler 5G yang diisolasi. Skenario transmisi difokuskan pada pengiriman matriks gambar berformat Base64 secara *persistent*. Untuk memastikan tingkat validitas statistik yang tinggi dan menghindari anomali jaringan sesaat, pengumpulan data dilakukan sebanyak 3 sesi pengujian independen dengan durasi masing-masing 5 menit untuk setiap protokol.

Total durasi observasi untuk setiap protokol adalah 15 menit, menghasilkan himpunan populasi sampel yang ekuivalen untuk dikomparasikan.

a. Data Waktu Tempuh Aplikasi (*End-to-End Latency*)

Data metrik latensi diekstraksi dari *database* PostgreSQL untuk merepresentasikan total waktu tempuh pengiriman data dari perangkat *edge* hingga selesai diproses oleh *backend*. Pengujian dilakukan sebanyak tiga sesi independen untuk setiap protokol menggunakan infrastruktur Wi-Fi 5G.

Hasil ekstraksi stempel waktu (*timestamp*) untuk protokol HTTP direkapitulasi pada Tabel 4.10.

Tabel 4.10 Hasil Kalkulasi Latensi Protokol HTTP

Sesi Pengujian	Total Sampel	Latensi Min (ms)	Latensi Max (ms)	Rata-rata Sesi (ms)
1	95	20	167	28.62
2	97	18	154	27.26
3	97	19	145	27.25
Keseluruhan	287	18	167	27.71

Data pada Tabel 4.10 menunjukkan bahwa pengiriman data menggunakan HTTP mencatatkan rata-rata latensi keseluruhan sebesar 27.71 ms, dengan nilai puncak tertinggi mencapai 167 ms.

Selanjutnya, hasil kalkulasi latensi untuk protokol WebSocket didokumentasikan pada Tabel 4.11.

Tabel 4.11 Hasil Kalkulasi Latensi Protokol WebSocket

Sesi Pengujian	Total Sampel	Latensi Min (ms)	Latensi Max (ms)	Rata-rata Sesi (ms)
1	96	16	150	25.87
2	96	17	146	24

3	96	18	144	25.47
Keseluruhan	287	81	150	25.11

Tabel 4.11 menunjukkan WebSocket memiliki rata-rata latensi keseluruhan sebesar 25.11 ms, serta mencatatkan nilai latensi minimum paling rendah, yakni 16 ms.

Kalkulasi latensi untuk protokol MQTT disajikan pada Tabel 4.12.

Tabel 4.12 Hasil Kalkulasi Latensi Protokol MQTT

Sesi Pengujian	Total Sampel	Latensi Min (ms)	Latensi Max (ms)	Rata-rata Sesi (ms)
1	96	21	34	25
2	97	19	37	26
3	97	20	110	27.30
Keseluruhan	290	19	110	26.10

Data pada Tabel 4.12 menunjukkan MQTT memiliki nilai latensi maksimum yang jauh lebih rendah dibandingkan kedua protokol sebelumnya, yakni hanya menyentuh angka 110 ms.

Untuk melihat perbandingan secara langsung, nilai keseluruhan dari ketiga protokol tersebut ditarik menjadi satu tabel rekapitulasi akhir yang merepresentasikan kinerja absolut setiap protokol.

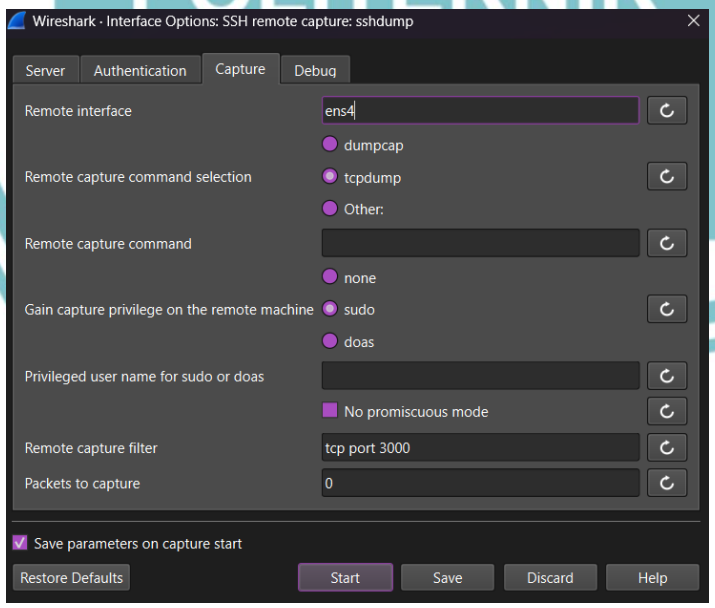
Tabel 4.13 Agregasi Komparatif Latensi Antarprotokol

Protokol	Total Sampel Populasi	Rentang Latensi (Min - Max)	Latensi Rata-rata Absolut
WebSocket	288 Paket	16 ms - 150 ms	25.11 ms
MQTT	290 Paket	19 ms - 110 ms	26.10 ms
HTTP	289 Paket	18 ms - 167 ms	27.71 ms

Data komparatif pada Tabel 4.13 memperlihatkan secara objektif urutan kecepatan transmisi dari ketiga protokol yang diuji. WebSocket mencatatkan rata-rata latensi paling cepat yakni 25.11 ms. MQTT menyusul dengan rata-rata 26.10 ms dan memiliki rentang latensi paling stabil dengan nilai maksimum terkecil. Di sisi lain, HTTP mencatatkan latensi rata-rata paling lambat sebesar 27.71 ms, serta memiliki anomali lonjakan nilai maksimum tertinggi yang mencapai 167 ms. Rekapitulasi angka ini secara murni menunjukkan hasil performa setiap protokol saat dihadapkan pada skenario pengiriman gambar secara kontinu di atas infrastruktur Wi-Fi 5G.

b. Data Analitik Jaringan (*Throughput, Packet Loss*, dan *Jitter*)

Data analitik jaringan ditarik dari lalu lintas tingkat *transport* yang direkam secara langsung melalui antarmuka *Virtual Machine* di GCP. Proses perekaman dilakukan menggunakan fitur *SSH remote capture* pada perangkat lunak Wireshark yang dispesifikasikan pada *port* masing-masing protokol untuk menghasilkan *file* perekaman berformat pcap. Gambar 4.29 mendemonstrasikan jendela konfigurasi penangkapan jarak jauh pada antarmuka Wireshark.



Gambar 4.29 Konfigurasi Penangkapan Jaringan Menggunakan SSH Remote Wireshark

File hasil perekaman tersebut kemudian dibedah menggunakan instrumen penganalisis protokol Wireshark untuk mengekstraksi metrik *Jitter*, *Throughput*, dan *Packet Loss*.

1) *Jitter*

Jitter merepresentasikan variasi jeda kedatangan paket yang terjadi selama transmisi berlangsung. Nilai metrik Jitter diperoleh dari kalkulasi standar deviasi pada data selisih waktu di Wireshark setelah melalui proses pembersihan data untuk membuang anomali jeda interval diam aplikasi.

Hasil ekstraksi dan kalkulasi *Jitter* untuk protokol HTTP selama tiga sesi pengujian disajikan pada Tabel 4.14.

Tabel 4.14 Hasil Kalkulasi *Jitter* Protokol HTTP

Sesi Pengujian	Total Durasi Sesi	*Jitter* (ms)
Sesi 1	5 Menit	23.17
Sesi 2	5 Menit	14.02
Sesi 3	5 Menit	10.80
Rata-rata Keseluruhan	15 Menit	16.00

Data pada Tabel 4.14 menunjukkan protokol HTTP memiliki fluktuasi jeda paket dengan rata-rata keseluruhan sebesar 16.00 ms.

Selanjutnya, hasil kalkulasi *Jitter* untuk protokol WebSocket didokumentasikan pada Tabel 4.15.

Tabel 4.15 Hasil Kalkulasi *Jitter* Protokol WebSocket

Sesi Pengujian	Total Durasi Sesi	*Jitter* (ms)
Sesi 1	5 Menit	77.65
Sesi 2	5 Menit	85.13
Sesi 3	5 Menit	71.87

Rata-rata Keseluruhan	15 Menit	78.22

Tabel 4.15 memperlihatkan variasi jeda kedatangan paket pada WebSocket dengan rata-rata fluktuasi keseluruhan mencapai 78.22 ms.

Kalkulasi *Jitter* untuk protokol MQTT disajikan pada Tabel 4.16.

Tabel 4.16 Hasil Kalkulasi *Jitter* Protokol MQTT

Sesi Pengujian	Total Durasi Sesi	*Jitter* (ms)
Sesi 1	5 Menit	43.61
Sesi 2	5 Menit	44.93
Sesi 3	5 Menit	30.55
Rata-rata Keseluruhan	15 Menit	39.70

Untuk membandingkan stabilitas transmisi antarprotokol secara objektif, nilai keseluruhan ditarik menjadi satu agregasi komparatif akhir.

Tabel 4.17 Agregasi Komparatif *Jitter* Antarprotokol

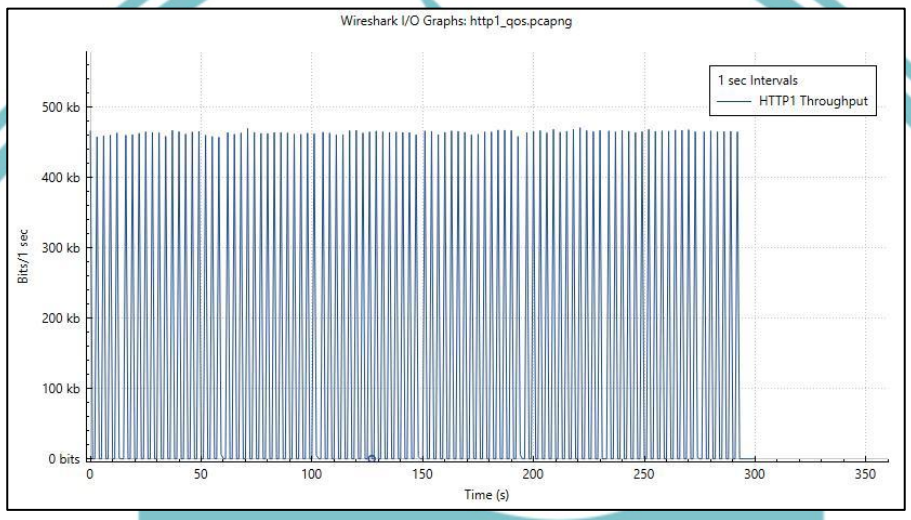
Protokol	Total Durasi Populasi	Rata-rata *Jitter*
HTTP	15 Menit	16.00 ms
MQTT	15 Menit	39.70 ms
Websocket	15 Menit	78.22 ms

Data komparatif pada Tabel 4.17 memperlihatkan bahwa HTTP mencatatkan tingkat stabilitas kedatangan paket tertinggi dengan rata-rata Jitter sebesar 16.00 ms. MQTT menyusul di urutan kedua dengan rata-rata 39.70 ms. Di sisi lain, WebSocket mengalami fluktuasi jeda pengiriman paling besar di antara ketiga protokol tersebut, yakni mencapai 78.22 ms.

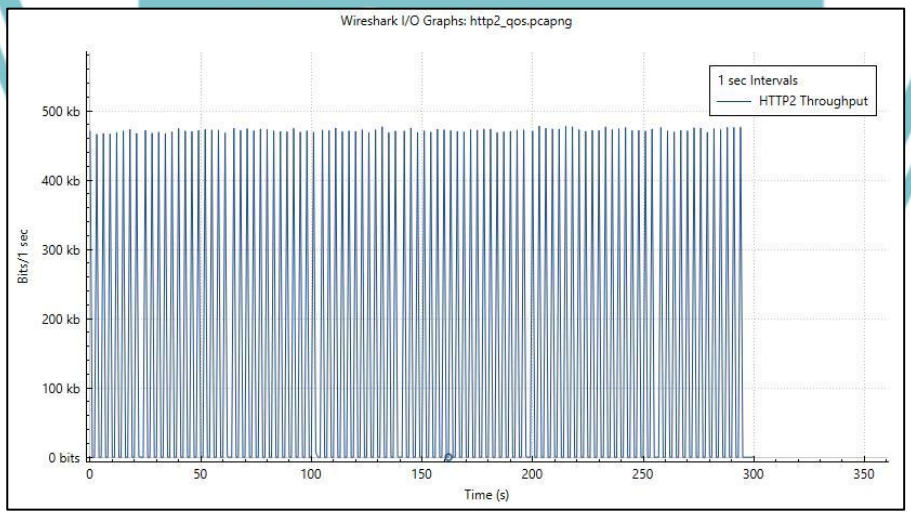
2) *Throughput*

Throughput merepresentasikan laju pemakaian jaringan selama proses transmisi berlangsung. Nilai ini mencakup muatan data aplikasi berupa gambar Base64 ditambah seluruh beban bawaan dari lapisan jaringan.

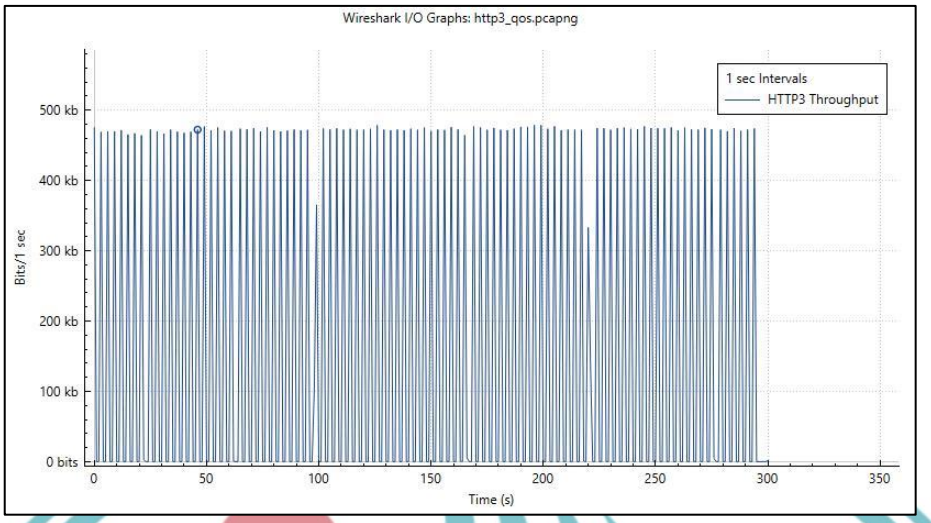
Distribusi laju data sepanjang periode transmisi 5 menit untuk protokol HTTP divisualisasikan melalui grafik I/O dari Wireshark pada Gambar 4.30, Gambar 4.31, dan Gambar 4.32.



Gambar 4.30 Grafik *Throughput* Protokol HTTP (Sesi 1)



Gambar 4.31 Grafik *Throughput* Protokol HTTP (Sesi 2)



Gambar 4.32 Grafik *Throughput* Protokol H TTP (Sesi 3)

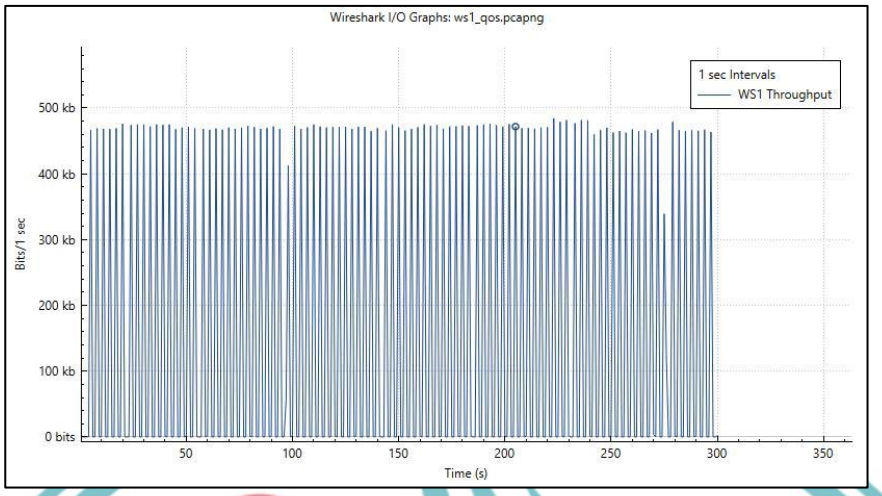
Berdasarkan Gambar 4.30, Gambar 4.31, dan Gambar 4.32, grafik memperlihatkan pemakaian jaringan pada protokol HTTP selama tiga sesi pengujian. Hasil kalkulasi Throughput untuk keseluruhan sesi pengujian HTTP disajikan pada Tabel 4.18.

Tabel 4.18 Hasil Kalkulasi *Throughput* Protokol HTTP

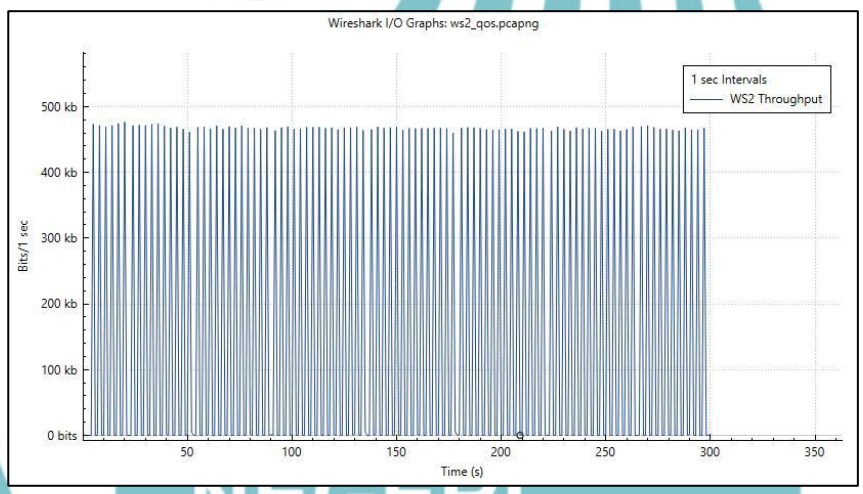
Sesi Pengujian	Total Durasi Sesi	*Throughput* (Kbps)
Sesi 1	5 Menit	148.15
Sesi 2	5 Menit	152.12
Sesi 3	5 Menit	152.36
Rata-rata Keseluruhan	15 Menit	150.88

Data pada Tabel 4.18 menunjukkan bahwa protokol HTTP mencatatkan rata-rata pemakaian jaringan sebesar 150.88 Kbps selama pengujian berlangsung.

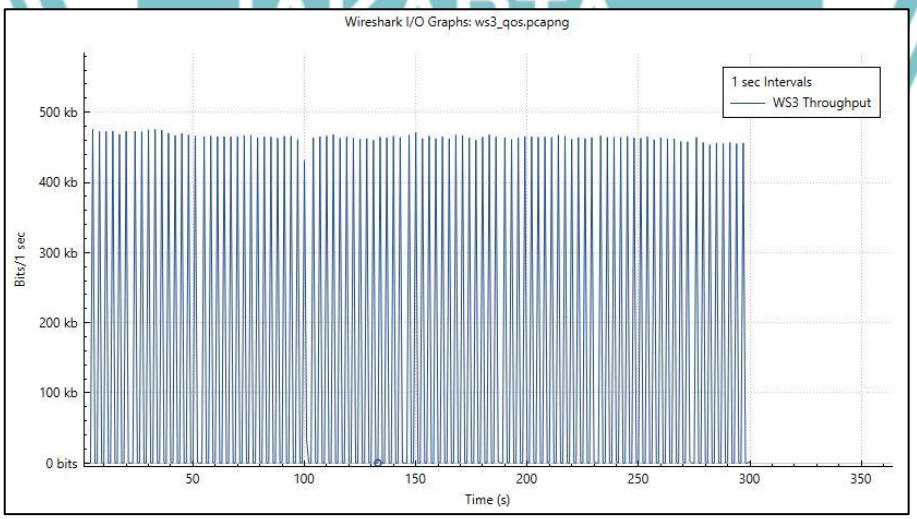
Selanjutnya, visualisasi grafik laju data untuk sesi pengujian protokol WebSocket disajikan pada Gambar 4.33, 4.34, dan 4.35.



Gambar 4.33 Grafik *Throughput* Protokol WebSocket (Sesi 1)



Gambar 4.34 Grafik *Throughput* Protokol WebSocket (Sesi 2)



Gambar 4.35 Grafik *Throughput* Protokol WebSocket (Sesi 3)

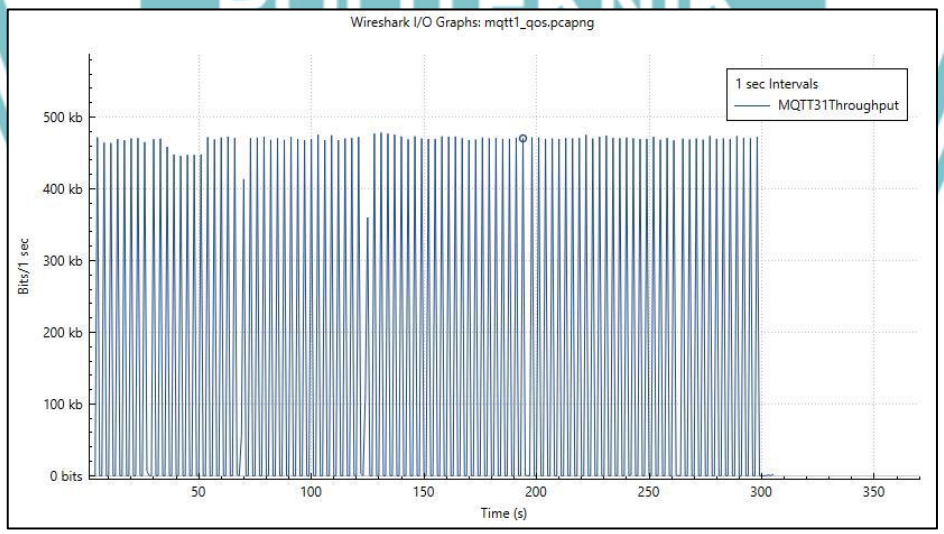
Pada Gambar 4.33, Gambar 4.34, dan Gambar 4.35, grafik menunjukkan pergerakan transmisi data untuk protokol WebSocket. Rekapitulasi nilai laju data untuk protokol WebSocket didokumentasikan pada Tabel 4.19.

Tabel 4.19 Hasil Kalkulasi *Throughput* Protokol WebSocket

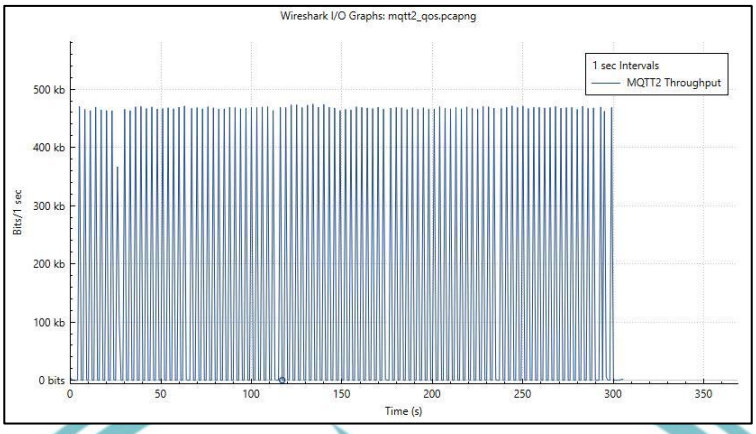
Sesi Pengujian	Total Durasi Sesi	*Throughput* (Kbps)
Sesi 1	5 Menit	154.25
Sesi 2	5 Menit	152.71
Sesi 3	5 Menit	151.88
Rata-rata Keseluruhan	15 Menit	152.95

Data pada Tabel 4.19 memperlihatkan bahwa WebSocket mencatatkan rata-rata penggunaan jaringan sebesar 152.95 Kbps.

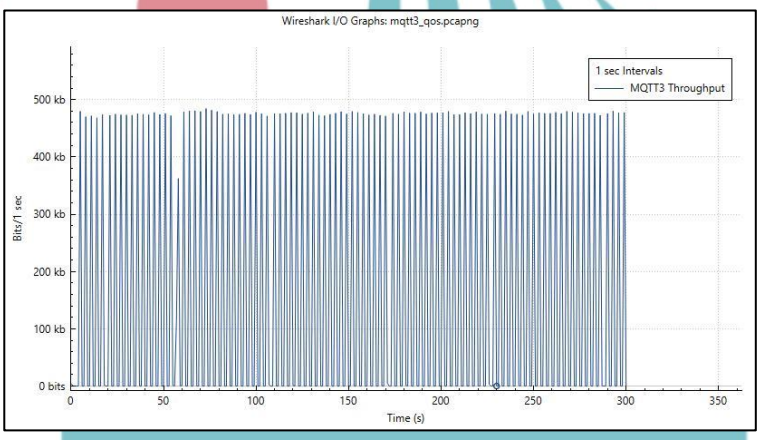
Terakhir, distribusi laju data transmisi untuk protokol standar IoT, yakni MQTT, divisualisasikan melalui Gambar 4.36, 4.37, dan 4.38.



Gambar 4.36 Grafik *Throughput* Protokol MQTT (Sesi 1)



Gambar 4.37 Grafik *Throughput* Protokol MQTT (Sesi 2)



Gambar 4.38 Grafik *Throughput* Protokol MQTT (Sesi 3)

Gambar 4.36, Gambar 4.37, dan Gambar 4.38 menampilkan visualisasi aliran transmisi data untuk protokol MQTT. Hasil kalkulasi dari grafik arus data MQTT tersebut direkapitulasi pada Tabel 4.20.

Tabel 4.20 Hasil Kalkulasi *Throughput* Protokol MQTT

Sesi Pengujian	Total Durasi Sesi	*Throughput* (Kbps)
Sesi 1	5 Menit	152.31
Sesi 2	5 Menit	153.40
Sesi 3	5 Menit	156.04
Rata-rata Keseluruhan	15 Menit	153.92

Data pada Tabel 4.20 menunjukkan rata-rata penggunaan jaringan pada protokol MQTT mencapai 153.92 Kbps.

Untuk membandingkan kapabilitas dan penggunaan Throughput dari ketiga protokol, nilai rata-rata keseluruhan disintesis ke dalam satu agregasi matriks komparatif akhir.

Tabel 4.21 Agregasi Komparatif *Throughput* Antarprotokol

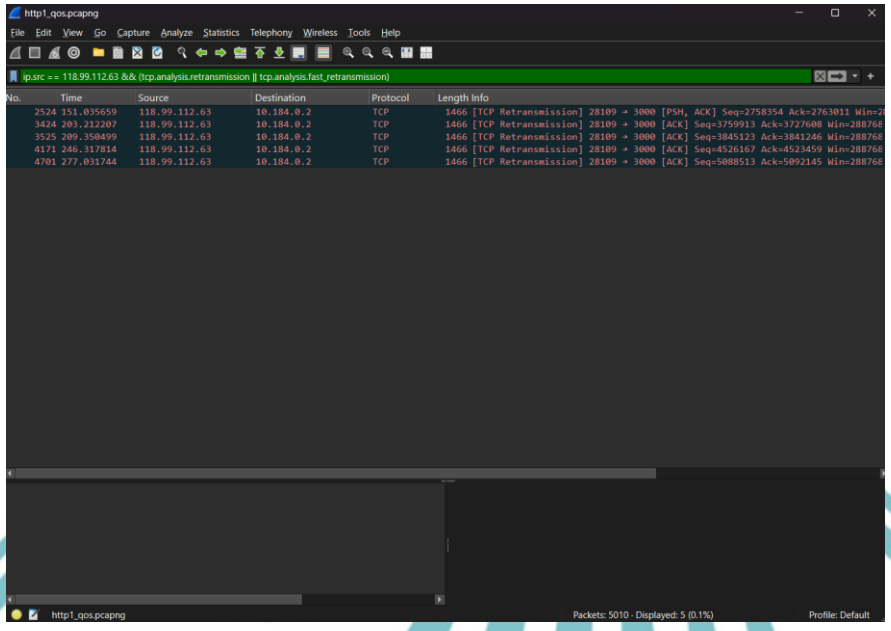
Protokol	Total Durasi Populasi	Rata-rata *Throughput*
MQTT	15 Menit	153.92 Kbps
Websocket	15 Menit	152.95 Kbps
HTTP	15 Menit	150.88 Kbps

Data pada Tabel 4.21 memperlihatkan urutan tingkat pemakaian laju jaringan dari ketiga protokol yang diuji. MQTT mencatatkan penggunaan tertinggi sebesar 153.92 Kbps. WebSocket menempati urutan kedua dengan 152.95 Kbps, sedangkan HTTP mencatatkan tingkat pemakaian terendah sebesar 150.88 Kbps.

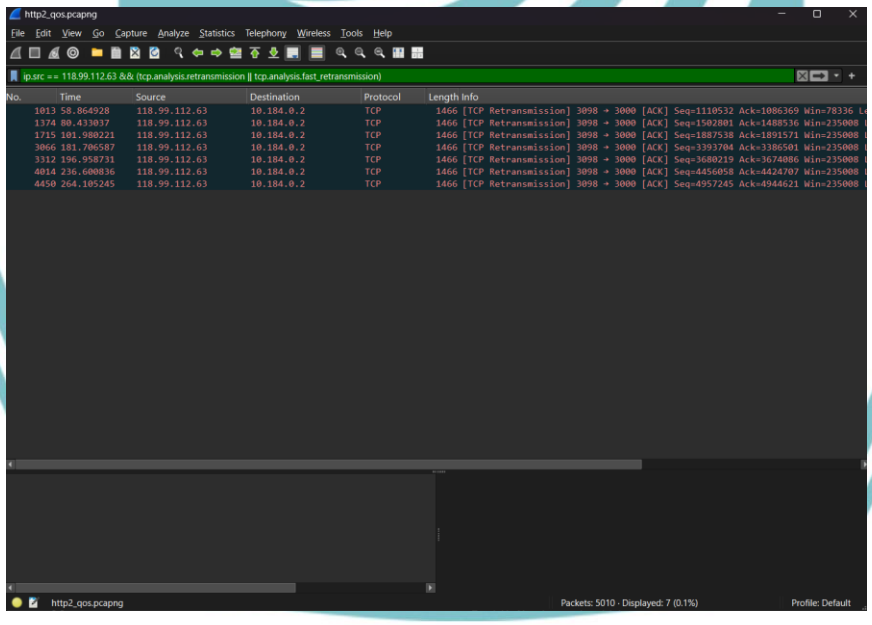
3) *Packet Loss*

Metrik Packet Loss mengukur persentase kegagalan pengiriman paket pada lapisan *transport* yang memicu mekanisme pengiriman ulang oleh TCP.

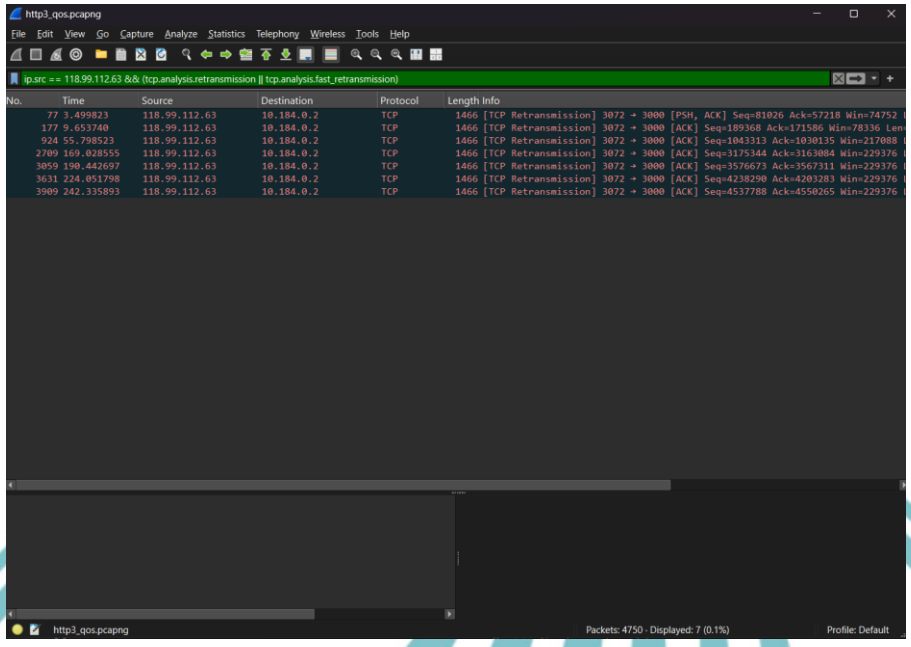
Hasil penyaringan data untuk melihat retransmisi TCP pada lalu lintas jaringan protokol HTTP divisualisasikan melalui tangkapan layar Wireshark pada Gambar 4.39, Gambar 4.40, dan Gambar 4.41.



Gambar 4.39 Hasil Pemindaian Retransmisi TCP Protokol HTTP (Sesi 1)



Gambar 4.40 Hasil Pemindaian Retransmisi TCP Protokol HTTP (Sesi 2)



Gambar 4.41 Hasil Pemindaian Retransmisi TCP Protokol HTTP (Sesi 3)

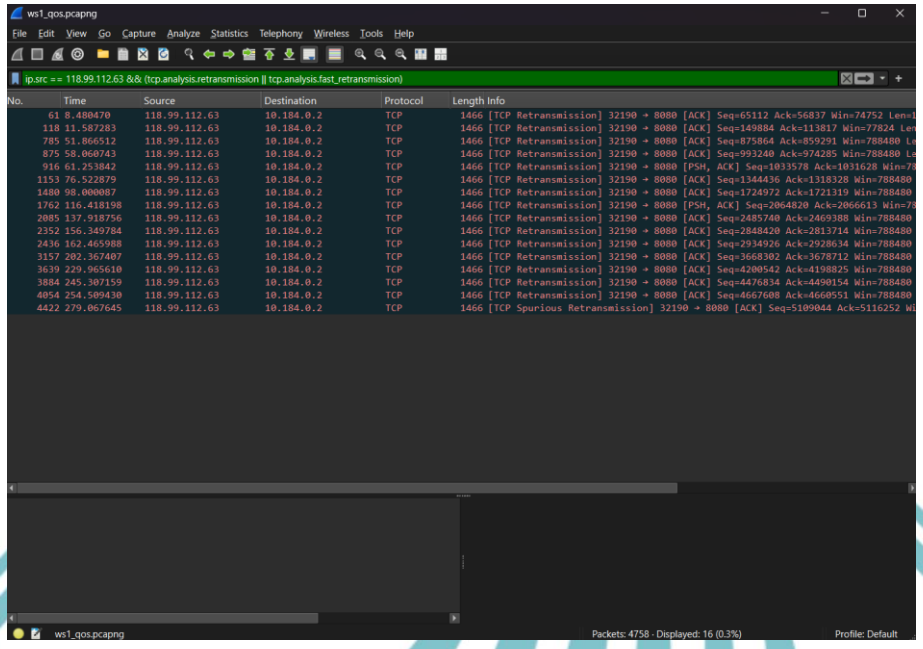
Hasil kalkulasi rasio *Packet Loss* untuk keseluruhan sesi transmisi HTTP didokumentasikan pada Tabel 4.22.

Tabel 4.22 Hasil Kalkulasi *Packet Loss* Protokol HTTP

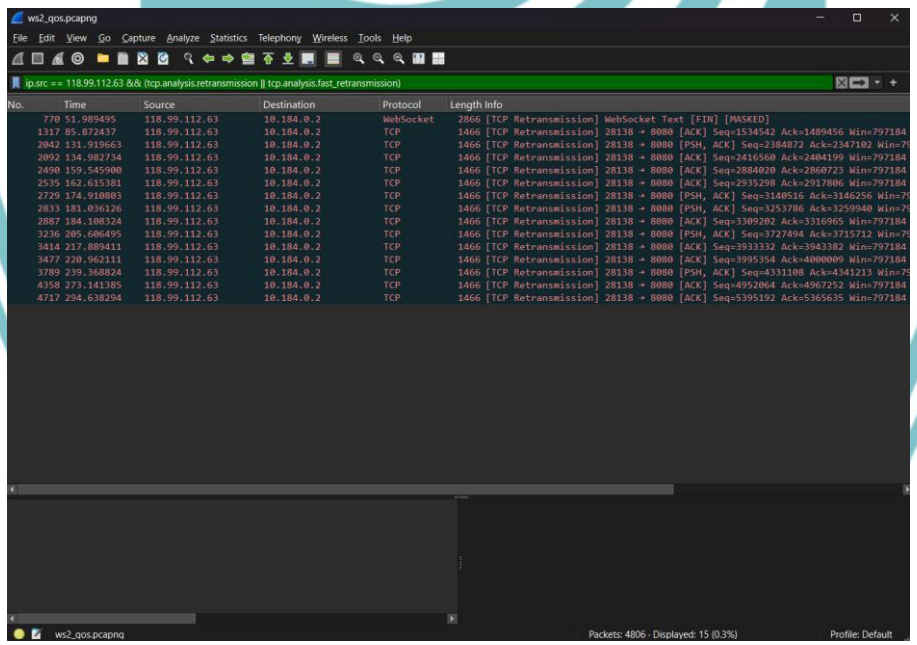
Sesi Pengujian	Total Paket Transmisi	Paket Hilang / Retransmisi	Persentase *Packet Loss*
Sesi 1	2.438 Paket	5	0.20%
Sesi 2	2.507 Paket	7	0.27%
Sesi 3	2.436 Paket	7	0.28%
Keseluruhan	7.381 Paket	19	0.26%

Data pada Tabel 4.22 menunjukkan transmisi HTTP mencatatkan rata-rata Packet Loss sebesar 0.26% selama periode observasi berlangsung.

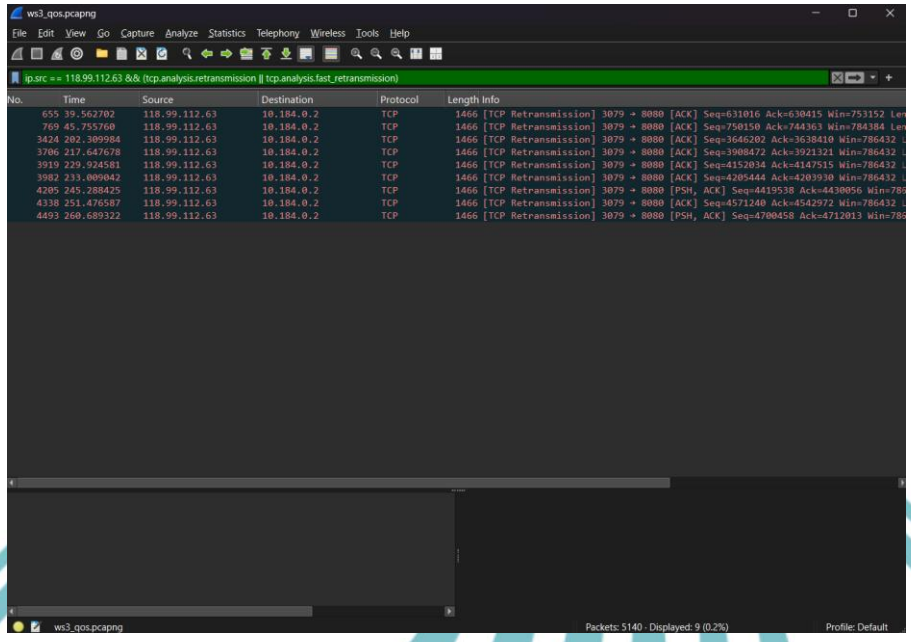
Selanjutnya, hasil pemindaian retransmisi TCP untuk pengujian protokol WebSocket disajikan pada Gambar 4.42, 4.43, dan 4.44.



Gambar 4.42 Pemindaian Retransmisi TCP Protokol WebSocket (Sesi 1)



Gambar 4.43 Pemindaian Retransmisi TCP Protokol WebSocket (Sesi 2)



Gambar 4.44 Pemindaian Retransmisi TCP Protokol WebSocket (Sesi 3)

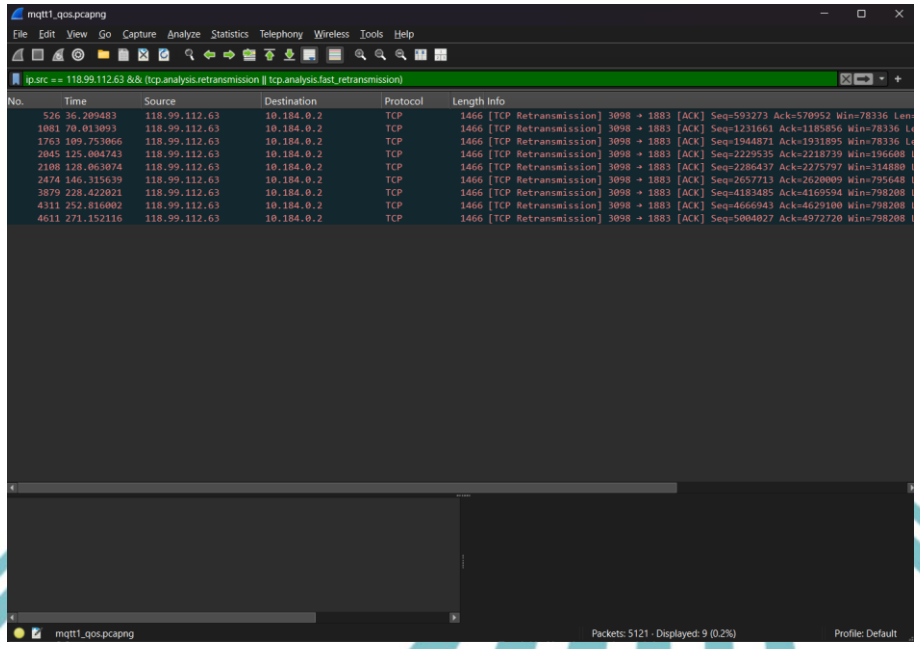
Rekapitulasi rasio keberhasilan pengiriman untuk protokol WebSocket disajikan pada Tabel 4.23.

Tabel 4.23 Hasil Kalkulasi *Packet Loss* Protokol WebSocket

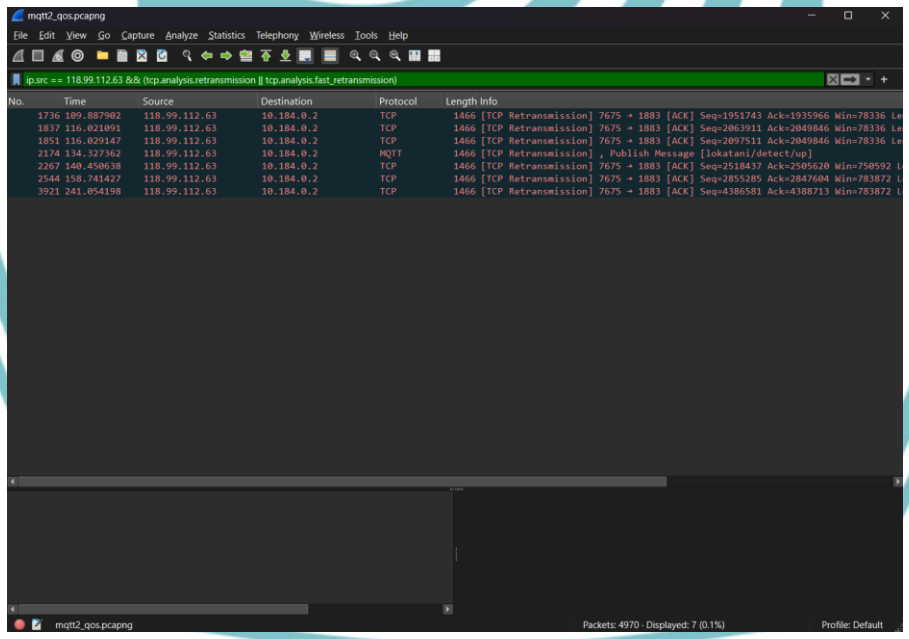
Sesi Pengujian	Total Paket Transmisi	Paket Hilang / Retransmisi	Persentase *Packet Loss*
Sesi 1	2.321 Paket	16	0.69%
Sesi 2	2.307 Paket	15	0.65%
Sesi 3	2.437 Paket	9	0.37%
Keseluruhan	7.065 Paket	40	0.57%

Tabel 4.23 memperlihatkan bahwa protokol WebSocket mencatatkan rata-rata *Packet Loss* keseluruhan sebesar 0.57%.

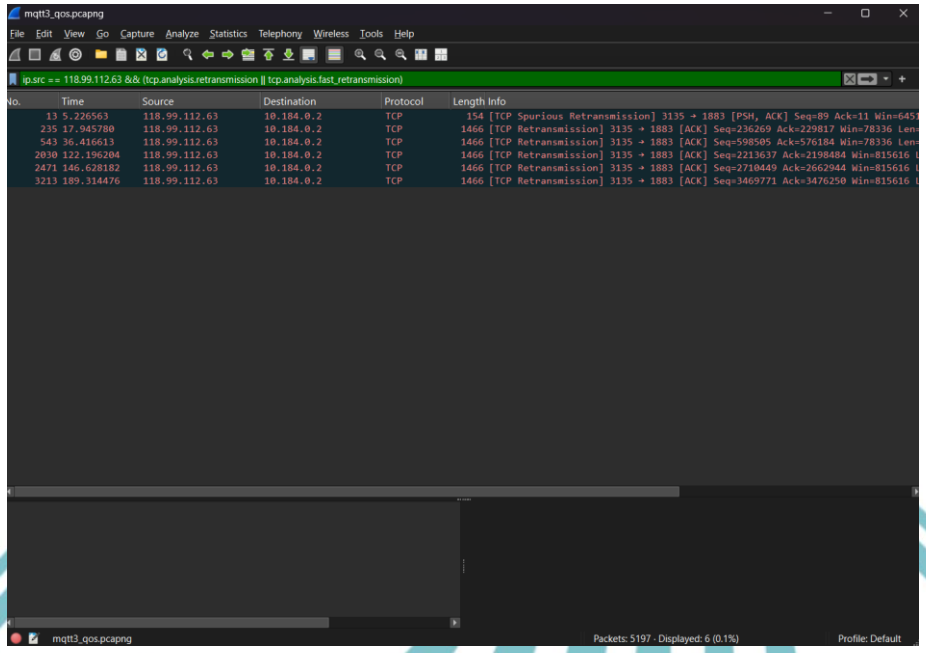
Terakhir, pemindaian lapisan transport untuk protokol standar telemetri IoT, yakni MQTT, ditunjukkan pada Gambar 4.45, 4.46, dan 4.47.



Gambar 4.45 Hasil Retransmisi TCP Protokol MQTT (Sesi 1)



Gambar 4.46 Hasil Retransmisi TCP Protokol MQTT (Sesi 2)



Gambar 4.47 Hasil Retransmisi TCP Protokol MQTT (Sesi 3)

Hasil kalkulasi *Packet Loss* untuk lalu lintas asinkron MQTT direkapitulasi pada Tabel 4.24.

Tabel 4.24 Hasil Kalkulasi *Packet Loss* Protokol MQTT

Sesi Pengujian	Total Paket Transmisi	Paket Hilang / Retransmisi	Persentase *Packet Loss*
Sesi 1	2.435 Paket	9	0.37%
Sesi 2	2.396 Paket	7	0.29%
Sesi 3	2.461 Paket	6	0.24%
Keseluruhan	7.292 Paket	22	0.30%

Data pada Tabel 4.24 menunjukkan protokol MQTT memiliki rata-rata *Packet Loss* sebesar 0.30%.

Untuk membandingkan tingkat keberhasilan infrastruktur transmisi dari ketiga protokol, seluruh angka diagregasi ke dalam satu tabel komparatif akhir.

Tabel 4.25. Agregasi komparasi akhir *Packet Loss*

Protokol	Total Paket Transmisi	Paket Hilang / Retransmisi	Persentase *Packet Loss*
HTTP	7.065 Paket	40 Paket	0.57%
WebSocket	7.292 Paket	22 paket	0.30%
MQTT	7.381 Paket	19 Paket	0.26%

Data komparatif pada Tabel 4.25 merepresentasikan tingkat kegagalan transmisi dari ketiga protokol. WebSocket mencatatkan persentase Packet Loss tertinggi yakni sebesar 0.57%. MQTT berada di urutan kedua dengan angka kegagalan 0.30%, sedangkan HTTP mencatatkan Packet Loss paling rendah yaitu sebesar 0.26% saat diuji menggunakan infrastruktur Wi-Fi 5G.

4) Rekapitulasi Metrik QoS

Keseluruhan data dari perhitungan latensi dan pengukuran Wireshark digabungkan menjadi satu tabel rekapitulasi akhir untuk melihat perbandingan kinerja dari masing-masing protokol. Hasil rekapitulasi ini disajikan pada Tabel 4.26.

Tabel 4.26 Rekapitulasi Hasil Keseluruhan Metrik QoS

Parameter QoS	HTTP	WebSocket	MQTT
Throughput Rata-rata (Kbps)	150.88	152.95	153.92
Packet Loss (%)	0.26%	0.57%	0.30%
Latency (ms)	27.71	25.11	26.10
Jitter (ms)	16.00	78.22	39.70

Berdasarkan Tabel 4.26, terlihat perbandingan hasil pengukuran dari keempat parameter QoS. Pada parameter Throughput, MQTT

mencatatkan nilai tertinggi sebesar 153.92 Kbps, disusul oleh WebSocket dan HTTP. Pada parameter *Packet Loss*, seluruh protokol berada di bawah angka 1%, dengan HTTP mencatatkan persentase kegagalan terkecil yakni 0.26%.

Untuk kecepatan waktu tempuh atau *Latency*, WebSocket menjadi yang paling cepat dengan rata-rata 25.11 ms, diikuti secara ketat oleh MQTT, dan HTTP menjadi yang paling lambat dengan 27.71 ms. Terakhir, pada parameter Jitter yang mengukur stabilitas jeda pengiriman, HTTP mencatatkan angka paling rendah dan stabil di 16.00 ms, sedangkan WebSocket memiliki tingkat fluktuasi jeda paling tinggi yang mencapai 78.22 ms.

C. Hasil Pengujian Keandalan Operasional Sistem

Pengujian keandalan operasional (*Reliability Testing*) mendokumentasikan performa arsitektur sistem perangkat lunak ketika dihadapkan pada skenario tekanan operasional aktual. Pengujian ini mengekstraksi data empiris terkait stabilitas penggunaan sumber daya komputasi di bawah beban kerja durasi panjang, kemampuan sistem dalam merespons anomali pemutusan jaringan fisik, serta tingkat resiliensi *server* utama dalam menghadapi kelumpuhan layanan dependensi.

1) Hasil Pengujian Keandalan Operasional Sistem

Uji ketahanan operasi sistem dieksekusi selama durasi pengamatan 6 jam tanpa henti menggunakan protokol MQTT. Pengujian ini memvalidasi stabilitas dua komponen kritis secara bersamaan, yakni alokasi memori pada *server Google Cloud Platform* (GCP) dan stabilitas suhu pada unit pemroses (CPU) Raspberry Pi di sisi *Edge*.

Stabilitas manajemen sumber daya pada *server* diekstraksi secara otomatis setiap 5 detik menggunakan skrip otomasi. Data mentah tersebut diagregasi menjadi rata-rata pemakaian per jam untuk memetakan tren penggunaan CPU dan RAM dari host mesin virtual maupun kontainer spesifik. Hasil

agregasi performa sumber daya selama 6 jam pengujian disajikan pada Tabel 4.27.

Tabel 4.27 Agregasi Performa Sumber Daya *Server* per Jam

Durasi Pengujian	CPU Host (%)	RAM Host (MB)	CPU *Backend* (%)	RAM *Backend* (%)	CPU *Database* (%)	RAM *Database* (%)
Jam ke-1	1.96	895.5	0.409	2.79	0.907	1.23
Jam ke-2	2.1	902.2	0.373	2.82	0.845	1.15
Jam ke-3	2	910.4	0.232	2.85	0.614	1.06
Jam ke-4	2.26	920.5	0.392	2.73	0.849	1.21
Jam ke-5	2.82	882.3	0.376	2	0.935	1.23
Jam ke-6	3.01	867.75	0.228	2.11	0.576	1.07

Berdasarkan Tabel 4.27, infrastruktur *cloud* berhasil mempertahankan penggunaan pemrosesan pada titik yang sangat efisien, dengan beban CPU Host maksimal hanya menyentuh 3.01%. Persentase penggunaan RAM pada *backend* Node.js berfluktuasi secara stabil pada rentang sempit 2.00% hingga 2.85%. Penurunan persentase memori pada jam ke-5 dan jam ke-6 secara empiris membuktikan bahwa mekanisme pembersihan memori otomatis (*Garbage Collection*) beroperasi sempurna untuk menghapus antrean data gambar yang telah diproses, sehingga tidak membentuk tren kenaikan eksponensial yang berujung pada kebocoran memori (*Memory Leak*). Penggunaan *Database* PostgreSQL juga berada pada rentang statis (~1%), mengonfirmasi kestabilan manajemen soket *persistent* di dalam sistem.

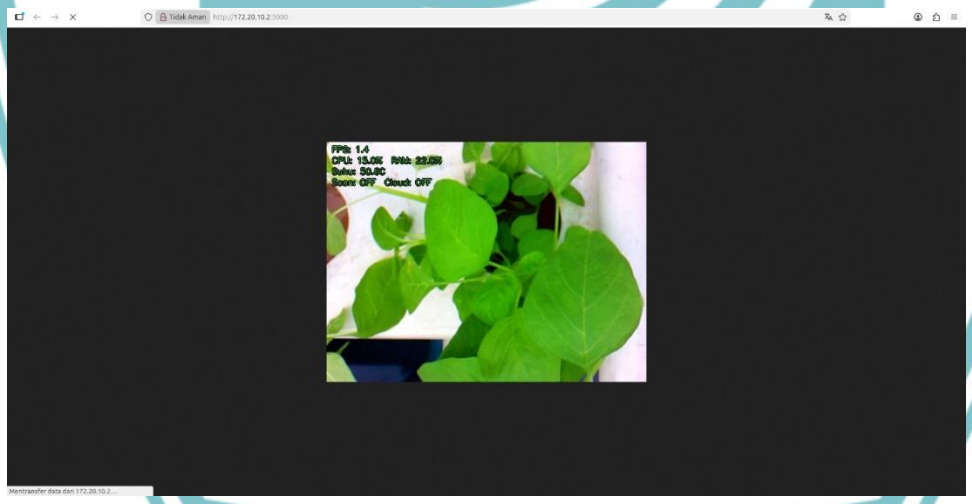
Fakta ketahanan arsitektural ini divalidasi lebih lanjut melalui status waktu aktif (*Uptime*) dari setiap kontainer pada akhir fase pengujian jam ke-6, sebagaimana ditunjukkan pada Gambar 4.48.

```
muhammadkhairumufid@hydrotect-server:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS
                    NAMES
4de57235d056   hydrotect-platform-frontend    "/docker-entrypoint.…"   4 days ago    Up 4 days
                    lokatani-frontend
0b6c873adc65   hydrotect-platform-backend     "docker-entrypoint.s…"   4 days ago    Up 4 days (healthy)
::]:8080->8080/tcp   lokatani-backend
f9b58420b2e9   postgres:16-alpine             "docker-entrypoint.s…"   4 weeks ago   Up 42 hours (healthy)
                    lokatani-db
9384a08c5288   eclipse-mosquitto:2            "/docker-entrypoint.…"   4 weeks ago   Up 42 hours
::]:9001->9001/tcp   lokatani-mqtt
```

Gambar 4.48 Hasil Perekaman *Uptime* Kontainer Pasca-Pengujian

Status "*Up*" pada Gambar 4.48 tanpa adanya intervensi muat ulang membuktikan seluruh layanan kebal terhadap terminasi paksa oleh *Out-Of-Memory* (OOM) *Killer* sistem operasi Linux.

Di sisi lain, beban pemrosesan transmisi ini juga berdampak langsung pada pelepasan panas perangkat keras *Edge*. Tangkapan layar yang memvisualisasikan parameter suhu Raspberry Pi melalui antarmuka pemantauan direpresentasikan pada Gambar 4.49.



Gambar 4.49 Pemantauan Visual Suhu Operasional Perangkat *Edge*

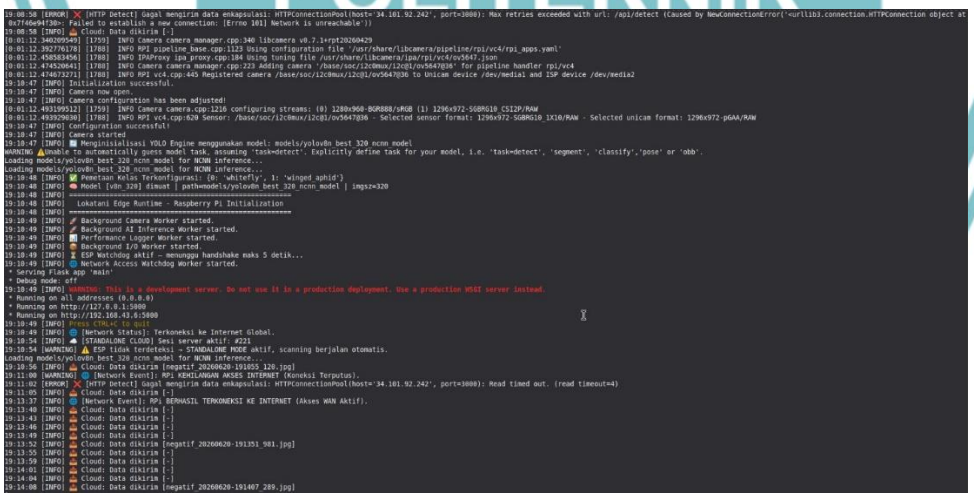
Perekaman performa suhu pada gambar 4.49 menunjukkan bahwa CPU Raspberry Pi mengalami peningkatan suhu pada fase inisiasi inferensi model YOLOv8, namun berhasil mencapai dan tertahan pada Kesetimbangan suhu di rentang statis 50.6°C hingga 55.0°C. Fakta bahwa suhu maksimal tertahan jauh di bawah ambang batas kritis perlambatan perangkat keras (*Thermal Throttling* pada 80°C) memberikan penilaian

bahwa arsitektur *Edge* beroperasi secara stabil dan aman untuk skenario penyisiran *greenhouse* berdurasi panjang.

2) Hasil Uji *Recovery* Jaringan Perangkat *Edge* (*Auto-Reconnect*)

Pengujian ini mengevaluasi kapabilitas *Exception Handling* pada skrip *Client* IoT di perangkat keras *Edge* saat menghadapi interupsi pemutusan jaringan fisik secara paksa. Fokus evaluasi terletak pada kemampuan utas program (*thread*) untuk menolak terminasi paksa akibat ketiadaan respons soket, serta kecepatan sistem dalam menyambung kembali (*Auto-Reconnect*) setelah akses internet (DNS) dipulihkan. Waktu *Recovery* dikalkulasi secara presisi dari selisih stempel waktu saat log "Akses WAN Aktif" muncul hingga transmisi data pertama berhasil dikonfirmasi.

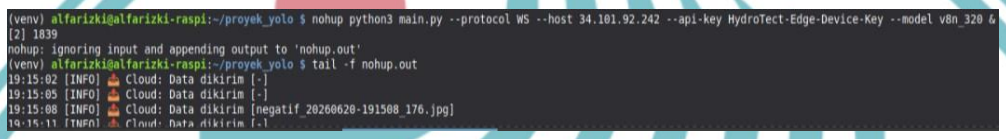
Pada pengujian menggunakan protokol HTTP, pemutusan jaringan memicu eksepsi yang berhasil ditangkap (*catch*) oleh sistem tanpa membekukan antrean kamera. Tangkapan layar terminal operasional pada Gambar 4.50 menunjukkan rentetan log kegagalan pengiriman yang disusul dengan keberhasilan transmisi paket sesaat setelah jaringan pulih.



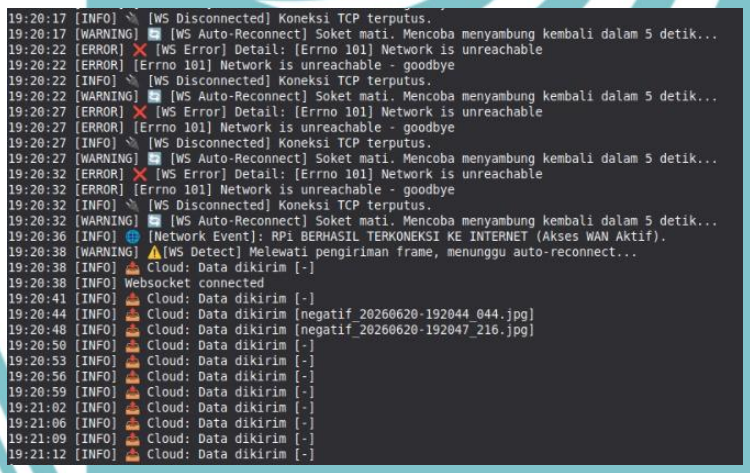
Gambar 4.50 Log Pemutusan dan Restorasi Koneksi Protokol HTTP pada Perangkat *Edge*

Berdasarkan log waktu komputasi, perangkat *Edge* mendeteksi *Recovery* akses internet pada pukul 19:13:37 dan berhasil mengeksekusi pengiriman payload HTTP pertama pada pukul 19:13:40, menghasilkan waktu *Recovery* aplikasi sebesar 3 detik.

Selanjutnya, pengujian pada protokol WebSocket didokumentasikan pada Gambar 4.51 dan Gambar 4.52. Skrip *Client* berhasil mendeteksi putusnya soket TCP dua arah secara otonom dan menginisiasi siklus penyambungan ulang secara proaktif.



Gambar 4.51 Log Transmisi Normal Protokol WebSocket

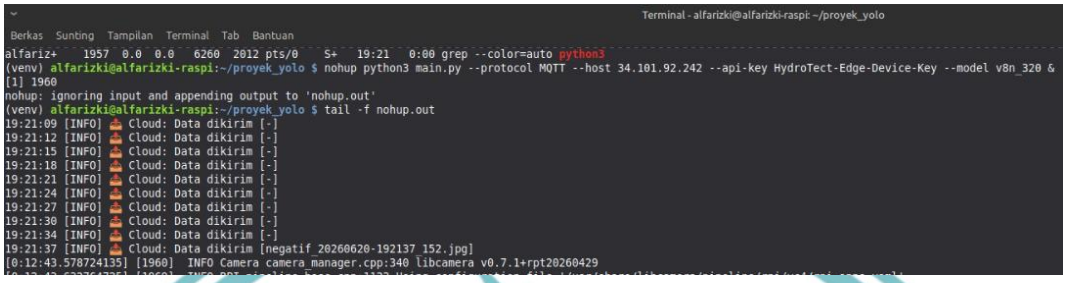


Gambar 4.52 Log Pemutusan dan Restorasi Soket *Persistent* WebSocket

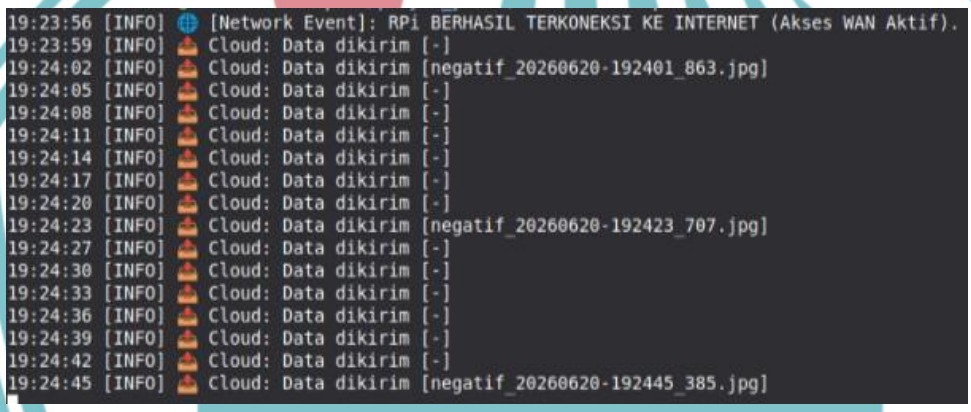
Data operasional mencatatkan bahwa sistem mendeteksi ketersediaan internet pada pukul 19:20:36 dan berhasil memulihkan *handshake* soket WebSocket untuk mengirim data pada pukul 19:20:38. Kecepatan *Recovery* ini merupakan yang tercepat dengan durasi hanya 2 detik.

Pengujian terakhir dieksekusi pada arsitektur asinkron MQTT. Bukti visual penanganan kejadian (*event handling*) oleh *background thread* Paho-MQTT

saat soket ditutup secara paksa dan disambung kembali disajikan pada Gambar 4.53 dan Gambar 4.54.



Gambar 4.53 Log Transmisi Normal MQTT



Gambar 4.54 Log Restorasi Status Terhubung dan Transmisi Lanjutan MQTT

Thread jaringan MQTT mendeteksi *Recovery* ping internet pada pukul 19:23:56 dan sukses memicu *network event* "RPi berhasil terkoneksi ke internet" sekaligus meneruskan pengiriman data yang tertunda pada pukul 19:23:59. Durasi *Recovery* ini terkalkulasi sebesar 3 detik.

Untuk memberikan komparasi daya tahan secara menyeluruh, metrik kelangsungan program dan waktu *Recovery* jaringan dari ketiga protokol direkapitulasi pada Tabel 4.28.

Tabel 4.28 Agregasi Kinerja *Recovery* Jaringan Otomatis

Protokol Transmisi	Status Terminasi	Titik Waktu Jaringan Pulih (WIB)	Titik Waktu Transmisi Sukses (WIB)	Waktu *Recovery* (*Recovery Time*)
HTTP	Bertahan (Tidak *Crash*)	19:13:37	19:13:40	3 Detik
WebSocket	Bertahan (Tidak *Crash*)	19:20:36	19:20:38	2 Detik
MQTT	Bertahan (Tidak *Crash*)	19:23:56	19:23:59	3 Detik

3) Hasil Uji Ketahanan *Microservices*

Pengujian ketahanan arsitektur *microservices* bertujuan untuk mengevaluasi kekebalan sistem terhadap kegagalan berantai. Fokus pengujian adalah memastikan bahwa berhentinya satu kontainer layanan dependensi tidak menyebabkan *server backend* ikut mengalami *crash*, dan sistem mampu memulihkan aliran data secara otonom sesaat setelah kontainer dependensi dihidupkan kembali.

a. Skenario Kegagalan Layanan *Broker* (MQTT)

Pada skenario ini, komunikasi aktif antara perangkat *edge* dan *server* melalui protokol MQTT diinterupsi. Interupsi disimulasikan dengan mengeksekusi perintah penghentian kontainer *broker* disertai jeda waktu pada antarmuka terminal, seperti yang ditunjukkan pada Gambar 4.55.

```
muhammadkhairumufid@hydrotect-server:~/hydrotect-platform$ docker compose stop mosquitto && sleep 10 && docker
compose start mosquitto
[+] stop 1/1
 ✔Container lokatani-mqtt Stopped
    0.5s
[+] start 1/1
 ✔Container lokatani-mqtt Started
    0.3s
```

Gambar 4.55 Eksekusi *Command* Penghentian Sementara Layanan MQTT

Validasi matinya layanan dibuktikan melalui pengecekan status kontainer pada Gambar 4.56. Saat kontainer MQTT dalam status berhenti, kontainer layanan inti lainnya terlihat tetap beroperasi secara normal.

```
muhammadkhairumufid@hydrotect-server:~$ docker container ls -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS
                                                     NAMES
f16a34e2fb3d   hydrotect-platform-frontend    "/docker-entrypoint.…"   2 hours ago   Up 2 hours
                                                     lokatani-frontend
5640155e965a   hydrotect-platform-backend     "docker-entrypoint.s…"   2 hours ago   Up 2 hours (healthy)
/tcp, 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp    lokatani-backend
f9b58420b2e9   postgres:16-alpine             "docker-entrypoint.s…"   6 weeks ago   Up 12 minutes (healthy)
/tcp                                                 lokatani-db
9384a08c5288   eclipse-mosquitto:2            "/docker-entrypoint.…"   6 weeks ago   Exited (0) 5 seconds ago
                                                     lokatani-mqtt
```

Gambar 4.56 Status Kontainer Saat Layanan MQTT Dimatikan

Setelah jeda waktu yang ditentukan berakhir, layanan kontainer otomatis dihidupkan kembali seperti yang diperlihatkan pada Gambar 4.57.

```
muhammadkhairumufid@hydrotect-server:~$ docker container ls -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS
                                                     NAMES
f16a34e2fb3d   hydrotect-platform-frontend    "/docker-entrypoint.…"   2 hours ago   Up 2 hours
                                                     lokatani-frontend
5640155e965a   hydrotect-platform-backend     "docker-entrypoint.s…"   2 hours ago   Up 2 hours (healthy)
tcp, 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp     lokatani-backend
f9b58420b2e9   postgres:16-alpine             "docker-entrypoint.s…"   6 weeks ago   Up 12 minutes (healthy)
tcp                                                  lokatani-db
9384a08c5288   eclipse-mosquitto:2            "/docker-entrypoint.…"   6 weeks ago   Up 2 seconds
tcp, 0.0.0.0:9001->9001/tcp, [::]:9001->9001/tcp     lokatani-mqtt
```

Gambar 4.57 Status Kontainer Saat Layanan MQTT Beroperasi Kembali

Log pada *server backend* mencatat seluruh rangkaian kejadian tersebut, mulai dari terputusnya koneksi, proses penyambungan ulang, hingga perhitungan durasi pemulihan. Berdasarkan catatan log sistem pada Gambar 4.58, total waktu henti yang terekam adalah 15.019 detik. Setelah dikurangi dengan jeda pemadaman buatan selama 10 detik, waktu pemulihan murni untuk layanan ini tercatat sebesar 5.019 detik.

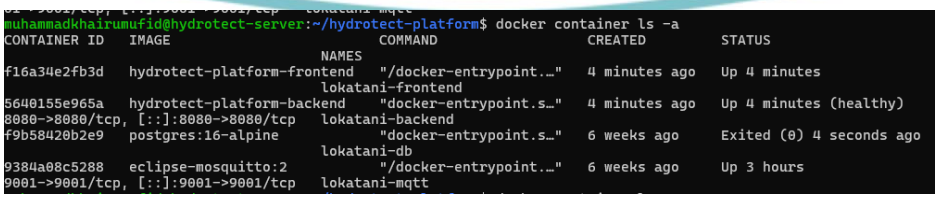
Gambar 4.58 Log *Backend* Menangkap Pemutusan dan Pemulihan Koneksi MQTT

b. Skenario Kegagalan Layanan *Database* (PostgreSQL)

Pada skenario kedua, pengiriman data dialihkan menggunakan protokol HTTP. Pengujian disimulasikan dengan mematikan kontainer database melalui eksekusi perintah terminal, lalu dihidupkan kembali setelah beberapa saat seperti yang didemonstrasikan pada Gambar 4.59.

Gambar 4.59 Eksekusi Perintah Penghentian dan Pemulihan Layanan *Database*

Pengecekan status kontainer saat *database* mengalami kelumpuhan ditunjukkan secara spesifik pada Gambar 4.60.



Gambar 4.60 Status Kontainer Saat Layanan *Database* Dimatikan

Status kontainer saat layanan *database* kembali beroperasi secara normal didokumentasikan pada Gambar 4.61.

```
muhammadkhairumufid@hydrotect-server:~/hydrotect-platform$ docker container ls -a
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS
                                 NAMES
f16a34e2fb3d   hydrotect-platform-frontend   "/docker-entrypoint.…"   6 minutes ago    Up 5 minutes
                                 lokatani-frontend
5640155e965a   hydrotect-platform-backend    "docker-entrypoint.s…"   6 minutes ago    Up 5 minutes (healthy)
.0:8080->8080/tcp, [::]:8080->8080/tcp   lokatani-backend
f9b58420b2e9   postgres:16-alpine            "docker-entrypoint.s…"   6 weeks ago      Up About a minute (healthy)
                                 lokatani-db
9384a08c5288   eclipse-mosquitto:2           "/docker-entrypoint.…"   6 weeks ago      Up 3 hours
.0:9001->9001/tcp, [::]:9001->9001/tcp   lokatani-mqtt
```

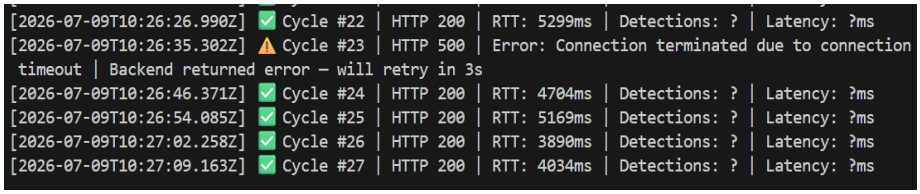
Gambar 4.61 Status Kontainer Saat Layanan *Database* Dipulihkan

Tangkapan log *backend* pada Gambar 4.62 menunjukkan kemampuan sistem dalam menangani galat koneksi *database* tanpa menghentikan proses utama aplikasi. Log tersebut mencatat total waktu henti sebesar 18.044 detik, yang berarti waktu pemulihan murninya adalah 8.044 detik setelah dikurangi jeda penghentian manual .

```
lokatani-backend  | [INFO ] 2026-07-09T10:26:22.093Z [ALERT] Created high alert #448 for log #12670
lokatani-backend  | [INFO ] 2026-07-09T10:26:30.174Z [ALERT] Created high alert #449 for log #12671
lokatani-backend  | [WARN ] 2026-07-09T10:26:30.925Z [RECOVERY] ⚠️ DOWNTIME_START | Service: POSTGRESQL | Time
stamp: 2026-07-09T10:26:30.925Z | Status: OFFLINE
lokatani-backend  | [ERROR] 2026-07-09T10:26:30.928Z [DB] Unexpected pool error: terminating connection due to
administrator command
lokatani-backend  | [ERROR] 2026-07-09T10:26:30.929Z [DB] Unexpected pool error: terminating connection due to
administrator command
lokatani-backend  | [ERROR] 2026-07-09T10:26:42.980Z [CONTROLLER] Detection error (HTTP): Connection terminated
 due to connection timeout
lokatani-backend  | [INFO ] 2026-07-09T10:26:48.969Z [RECOVERY] ✅ RECOVERY | Service: POSTGRESQL | Down_At: 20
26-07-09T10:26:30.925Z | Up_At: 2026-07-09T10:26:48.969Z | Recovery_Time: 18044ms (18.044s) | Status: ONLINE
lokatani-backend  | [INFO ] 2026-07-09T10:26:49.039Z [ALERT] Created low alert #450 for log #12672
lokatani-backend  | [INFO ] 2026-07-09T10:26:57.206Z [ALERT] Created medium alert #451 for log #12673
lokatani-backend  | [INFO ] 2026-07-09T10:27:04.878Z [ALERT] Created high alert #452 for log #12674
lokatani-backend  | [INFO ] 2026-07-09T10:27:11.925Z [ALERT] Created medium alert #453 for log #12675
```

Gambar 4.62 Log *Backend* Menangani Galat Kueri dan Pemulihan Layanan

Di sisi pengirim data, Gambar 4.63 memperlihatkan bahwa skrip berhasil menerima respons *error* dari *server* dan mengabaikannya untuk mencegah terhentinya siklus pengiriman interval 3 detik. Skrip terus mencoba mengirim data hingga akhirnya transmisi kembali sukses saat *database* sudah hidup kembali.



Gambar 4.63 Log Skrip *Mock* Mengabaikan Galat dan Melanjutkan Transmisi Data

c. Rekapitulasi Hasil Ketahanan *Microservices*

Untuk merangkum hasil dari kedua pengujian tersebut, parameter keandalan operasional disajikan pada Tabel 4.25. Untuk merangkum hasil dari kedua pengujian interupsi tersebut, parameter keandalan operasional disajikan pada Tabel 4.29. Waktu pemulihan murni dihitung dengan mengurangi total waktu henti yang terekam di sistem dengan variabel jeda pemadaman manual selama 10 detik.

Tabel 4.29 Rekapitulasi Uji Ketahanan *Microservices*

Skenario Kegagalan Dependensi	Status Layanan Node.js (*Backend*)	Status Utas Perangkat *Edge* (*Client*)	*Recovery Time*
Matikan Paksa MQTT Broker	Bertahan (*Up*), *Error Handled*	Bertahan, Masuk Fase *Reconnect*	5.019 Detik
Matikan Paksa PostgreSQL	Bertahan (*Up*), *Error Handled*	Bertahan, Mengabaikan Error	8.044 Detik

Data pada Tabel 4.29 membuktikan bahwa arsitektur sistem terhindar dari risiko kegagalan berantai. *Server backend* tetap mempertahankan operasionalnya saat layanan dependensi dimatikan secara paksa. Waktu pemulihan murni yang dibutuhkan untuk mengembalikan transmisi pada layanan *broker* tercatat sebesar 5.019 detik, sedangkan layanan *database* membutuhkan waktu sebesar 8.044 detik.

4.4.4. Analisis Data

Subbab ini menguraikan analisis dari data pengujian yang telah dipaparkan sebelumnya. Evaluasi dibagi menjadi tiga bagian utama, penerimaan fungsionalitas antarmuka perangkat lunak, perbandingan efisiensi protokol jaringan, dan ketahanan infrastruktur microservices secara keseluruhan.

A. Analisis Fungsionalitas dan Penerimaan Pengguna

Evaluasi fungsionalitas memperlihatkan hubungan yang selaras antara kemampuan teknis sistem dengan tingkat penerimaan pengguna. Hasil pengujian *Blackbox* (Tabel 4.7) mencatatkan tingkat keberhasilan 100%, yang memvalidasi ketiadaan cacat logika pada rute antar-kontainer dan manajemen otentikasi JWT. Keberhasilan teknis ini sejalan dengan hasil *User Acceptance Test* (Tabel 4.9), di mana pihak pengelola kebun memberikan skor maksimal (5) untuk indikator Stabilitas Sistem dan Kinerja Jaringan. Hal ini membuktikan bahwa proses pemuatan gambar Base64 secara asinkron berhasil dieksekusi tanpa memicu kebocoran memori pada *browser* pengguna

Di sisi lain, parameter Kemudahan Navigasi dan Kegunaan Data dari sistem pendukung keputusan memperoleh skor netral (3). Berdasarkan kerangka kerja *Technology Acceptance Model*, penurunan nilai kebermanfaatan ini sangat wajar dan sudah terprediksi, mengingat modul tersebut saat ini masih menggunakan data mitigasi umum. Ketiadaan data agronomi aktual yang divalidasi langsung oleh pakar secara otomatis menurunkan nilai guna dari fitur tersebut di lapangan. Meskipun demikian, tidak adanya laporan *bug* pada antarmuka aplikasi serta saran responden yang hanya berfokus pada penyesuaian fisik alat (seperti rotasi dan tinggi kamera) menegaskan bahwa perangkat lunak ini sudah matang dan siap untuk dioperasikan.

B. Analisis Perbandingan Kinerja Protokol QoS

Evaluasi kinerja QoS tidak ditujukan untuk mengukur kualitas fisik jaringan, melainkan untuk mengungkap ketidakefisienan arsitektur dari setiap protokol. Analisis ini didasarkan pada perhitungan matematis dan

diklasifikasikan menggunakan standar indeks TIPHON (ETSI TR 101 329 V2.1.1). Karena penggunaan infrastruktur Wi-Fi 5G telah menstabilkan kualitas saluran fisik, perbedaan performa yang muncul murni disebabkan oleh *overhead* protokol dan mekanisme pengiriman dari lapisan aplikasi HTTP, WebSocket, dan MQTT.

1) Integritas Saluran dan Analisis Efisiensi *Throughput*

Integritas jaringan fisik dievaluasi melalui metrik *Packet Loss* yang dihitung menggunakan rumus:

$$Packet\ Loss\ (\%) = \frac{(Paket\ Dikirim - Paket\ Diterima)}{Paket\ Dikirim} \times 100\%$$

Data pengujian empiris pada Tabel 4.23 memperlihatkan persentase kegagalan untuk protokol HTTP sebesar 0.26%, MQTT sebesar 0.30%, dan WebSocket sebesar 0.57%. Berdasarkan standardisasi TIPHON, seluruh hasil ini masuk ke dalam kategori "Sangat Bagus" (Indeks 4). Tingkat integritas transmisi yang sangat tinggi ini membuktikan tidak adanya kemacetan jaringan fisik sebagai faktor penghambat.

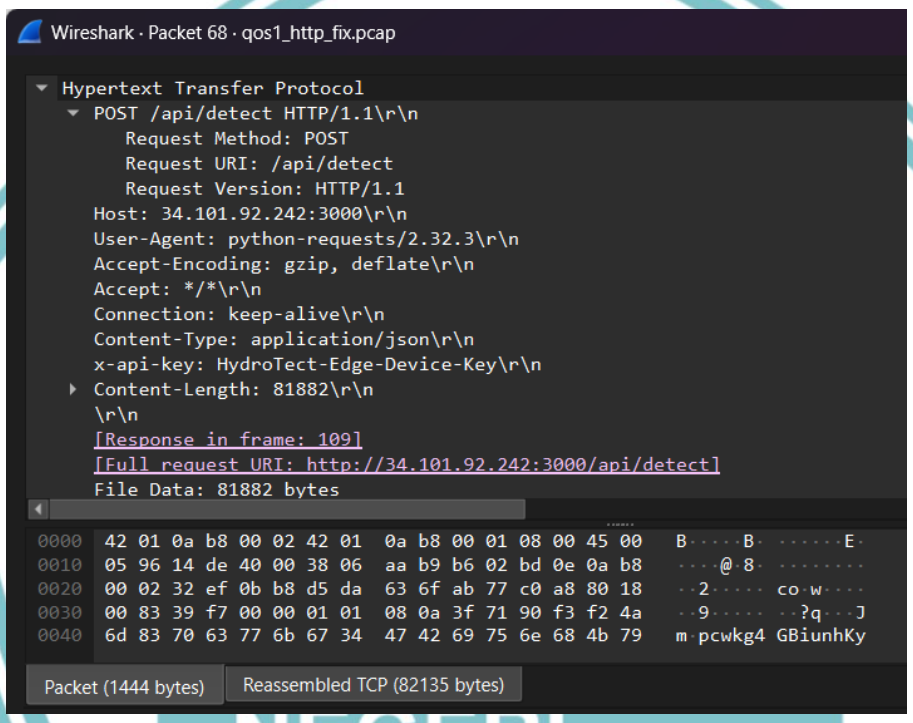
Analisis selanjutnya difokuskan pada efisiensi pemanfaatan *Throughput*, yang dikalkulasikan sebagai:

$$Throughput = \frac{Jumlah\ data\ yang\ dikirim}{Waktu\ pengiriman\ data}$$

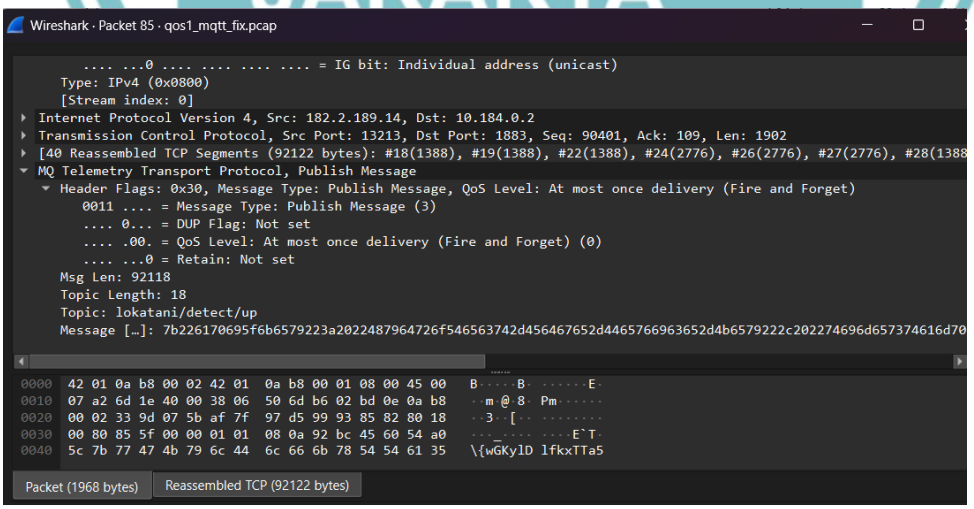
Hasil rekapitulasi membuktikan MQTT mencatatkan *Throughput* tertinggi (153.92 Kbps), melebihi WebSocket (152.95 Kbps) dan HTTP (150.88 Kbps). Tingginya pemakaian jaringan pada MQTT ini merupakan bukti efisiensi dari arsitektur asinkron. Tanpa adanya sistem penahanan yang menunggu balasan, MQTT mampu memasukkan tumpukan data ke dalam jaringan secara lancar. Sebaliknya, HTTP menjadi yang paling lambat dalam memanfaatkan saluran fisik akibat mekanisme transmisi sinkron

yang memaksa proses pengiriman tertahan untuk menunggu penerimaan kode status dari *server*.

Ketidakefisienan pada HTTP juga dipicu oleh pembengkakan teks pada lapisan aplikasi, yang dibuktikan melalui pembedahan paket pada Gambar 4.64 dan Gambar 4.65.



Gambar 4.64 *Overhead* Lapisan Aplikasi pada Protokol HTTP



Gambar 4.65 Efisiensi Framing Biner pada Protokol MQTT

Berdasarkan Gambar 4.64, setiap *payload* matriks yang dikirimkan melalui HTTP wajib merangkum teks header secara utuh dan berulang (seperti *Host*, *User-Agent*, *Connection*: *keep-alive*, *Content-Type*). Beban karakter teks mentah ini memicu pemborosan kapasitas saluran jaringan. Di sisi lain, Gambar 4.65 membuktikan bahwa MQTT beroperasi sebagai protokol ringan yang hanya membutuhkan *Fixed Header* biner untuk mengeksekusi transmisi berukuran besar (Amirkhanov dkk., 2025).

2) Analisis Waktu Tempuh Aplikasi (*End-to-End Latency*)

End-to-End Latency mewakili waktu tempuh pengiriman data dari ujung ke ujung yang dirumuskan sebagai:

$$\text{Latency} = \text{Waktu Terima}_{(\text{Server})} - \text{Waktu Kirim}_{(\text{Edge Device})}$$

Hasil perhitungan menempatkan rata-rata HTTP pada titik paling lambat (27.71 ms), sementara MQTT dan WebSocket lebih cepat (26.10 ms dan 25.11 ms). Berdasarkan klasifikasi TIPHON, rentang waktu ini sangat singkat dan masuk pada kategori "Sangat Bagus" (Indeks 4)

Secara teoritis, keterlambatan HTTP kerap dihubungkan dengan rekonstruksi iteratif dari TCP *Three-Way Handshake*. Namun, pembuktian pada konfigurasi *server backend* di Gambar 4.66 menolak asumsi tersebut.

```
// ── TCP Keep-Alive for High-Frequency Polling ──────
// Prevents TCP handshake overhead during 15 FPS HTTP
polling.
server.keepAliveTimeout = 65000;    // 65s — keep socket open
between polls
server.headersTimeout = 66000;      // Must exceed
keepAliveTimeout
```

Gambar 4.66 Implementasi Eksplisit Soket TCP *Keep-Alive* pada Lapisan Server

Kode pada Gambar 4.66 membuktikan bahwa *server* HTTP secara aktif menahan soket TCP agar tetap terbuka selama 65 detik di antara interval pengiriman, sehingga meniadakan proses pembukaan koneksi berulang di

lapisan jaringan. Keterlambatan waktu tempuh HTTP murni diakibatkan oleh hambatan pemrosesan teks pada prosesor. Mesin *backend* harus membongkar struktur teks *multipart/form-data* berukuran besar secara konstan. Sebaliknya, aliran data biner pada soket persisten WebSocket dan MQTT dieksekusi secara instan oleh sistem pembaca (*parser*) tanpa beban konversi tambahan.

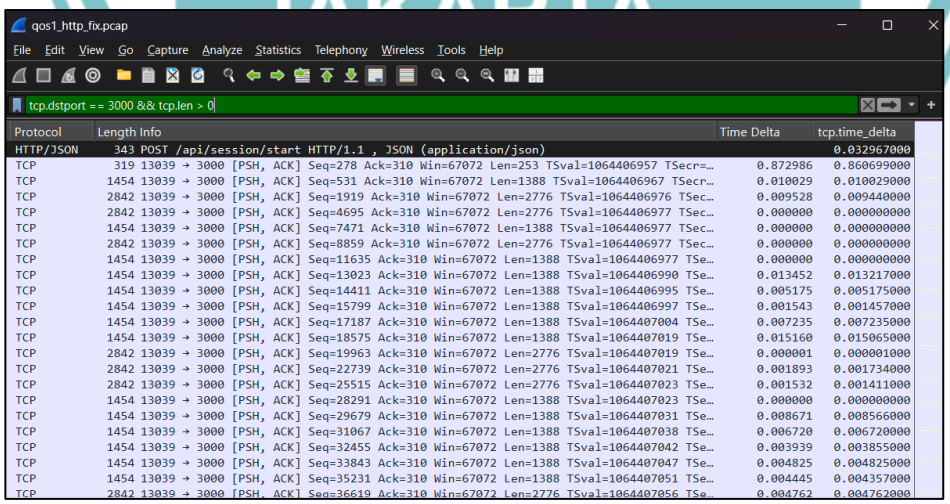
3) Analisis Stabilitas *Jitter* dan Bottleneck Perangkat Keras

Jitter menghitung simpangan waktu kedatangan antar-paket yang dirumuskan melalui Standar Deviasi Sampel:

$$Jitter = \frac{Total\ variasi\ delay}{Total\ paket\ yang\ diterima}$$

Hasil agregasi memperlihatkan HTTP menghasilkan *Jitter* terendah (16.00 ms) dibanding MQTT (39.70 ms) dan WebSocket (78.22 ms).

Nilai *Jitter* HTTP yang sangat rendah ini tidak menandakan efisiensi yang baik, melainkan sebuah ilusi metrik yang dipicu oleh mekanisme pengiriman data secara serentak (*burst transmission*). Hal ini dikonfirmasi melalui rekaman selisih waktu pada lapisan jaringan yang terlihat pada Gambar 4.67.



Gambar 4.67 Rapatnya *Interval Time Delta* Akibat Pola Burst Transmission HTTP

Pada Gambar 4.67, terlihat jeda waktu antar-paket mencapai angka yang sangat kecil. HTTP mengumpulkan potongan gambar berukuran besar dan mengirimkannya secara rapat pada waktu yang bersamaan karena sifat jalurnya yang sinkron.

Di sisi lain, lonjakan *Jitter* pada WebSocket (78.22 ms) membuktikan kelemahan protokol ini pada lingkungan perangkat keras berspesifikasi rendah. Sesuai dengan aturan Internet Engineering Task Force RFC 6455, standar komunikasi WebSocket mewajibkan proses enkripsi kriptografi (*XOR Masking*) untuk seluruh data sebelum dimasukkan ke saluran jaringan (Internet Engineering Task Force, 2011). Kewajiban pengodean algoritma pada gambar puluhan kilobita ini membebani prosesor Raspberry Pi, menghasilkan hambatan pemrosesan fisik yang memicu penundaan sesaat. Penundaan dorongan paket dari perangkat pengirim inilah yang pada akhirnya terekam sebagai fluktuasi *Jitter* yang tinggi di *server* penerima.

C. Analisis Keandalan Operasional Sistem (*Reliability*)

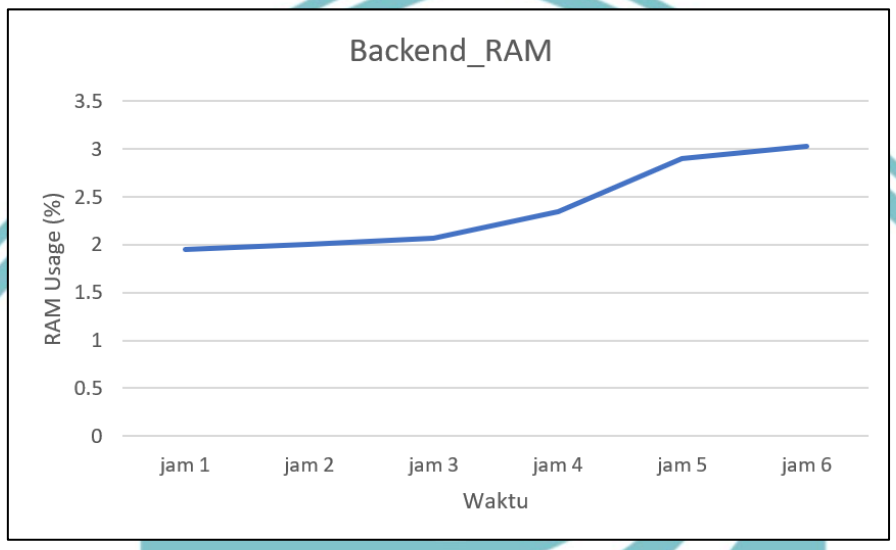
Analisis keandalan operasional membuktikan daya tahan arsitektur sistem dari tiga ancaman komputasi terdistribusi, degradasi perangkat keras, eror jaringan fisik, dan kegagalan dependensi kontainer. Pembuktian empiris pada domain ini membuktikan bahwa arsitektur perangkat lunak yang dirancang telah memenuhi standar *fault-tolerance* untuk diimplementasikan pada operasional jangka panjang di lapangan.

1) Evaluasi Sumber Daya dan *memory leak*

Penurunan performa pada sistem IoT berdurasi panjang umumnya dipicu oleh dua factor, perlambatan prosesor akibat panas berlebih dan *Memory Leak*. Perekaman suhu perangkat *Edge* membuktikan bahwa CPU Raspberry Pi tertahan secara statis pada rentang 50.6°C hingga 55.0°C selama 6 jam transmisi tanpa henti. Berhentinya kenaikan suhu secara eksponensial ini merupakan bukti empiris terciptanya Kesetimbangan suhu. Beban kerja inferensi YOLOv8 dan komputasi jaringan telah

seimbang dengan tingkat disipasi panas fisik perangkat, menjauhkan sistem dari batas kritis throttling (80°C).

Di sisi *server cloud*, ketahanan dianalisis melalui tren penggunaan Random Access Memory (RAM) pada layanan *backend* Node.js, sebagaimana divisualisasikan pada Gambar 4.68.

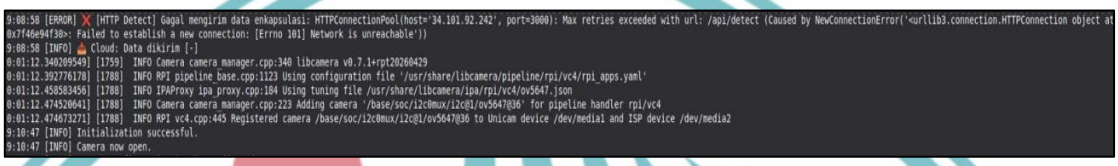


Gambar 4.68 Kurva Stabil Penggunan RAM *Server Backend* Selama 6 Jam

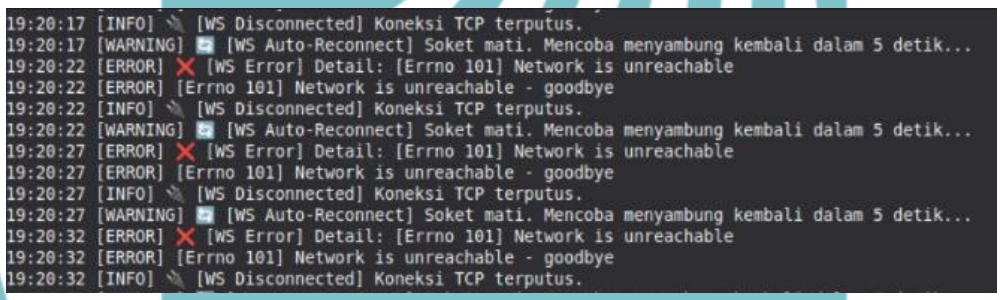
Grafik pada Gambar 4.68 memperlihatkan pola mendatar pada jam pertama hingga ketiga, disusul dengan sedikit eskalasi dari 2% menjadi 3% pada paruh kedua pengujian. Kenaikan minor sebesar 1% ini adalah hal wajar dari akumulasi ruang *cache* pada mesin V8 JavaScript, bukan indikator *Memory Leak*. Ketiadaan lonjakan grafik secara diagonal menuju angka 100% merupakan bukti absolut bahwa mekanisme *Garbage Collection* di dalam Node.js beroperasi dengan sempurna. Mesin *backend* secara berkala sukses membuang ribuan alokasi memori dari data gambar Base64 yang telah selesai didistribusikan ke *database*, sehingga sistem terbebas dari masalah penyimpanan.

2) Resiliensi Perangkat *Edge* dan Penanganan Eksepsi

Kestabilan operasional perangkat keras *Edge* tidak memiliki nilai jika aplikasi di dalamnya rentan terhadap *Crash*. Pemutusan koneksi internet secara fisik membuktikan efektivitas implementasi blok penanganan eror (*Exception Handling*) pada skrip aktuator Python. Hal ini terekam jelas pada log eksekusi terminal (Gambar 4.69 dan 4.70).



Gambar 4.69 Log Tangkapan Eksepsi [Errno 101] pada Protokol HTTP



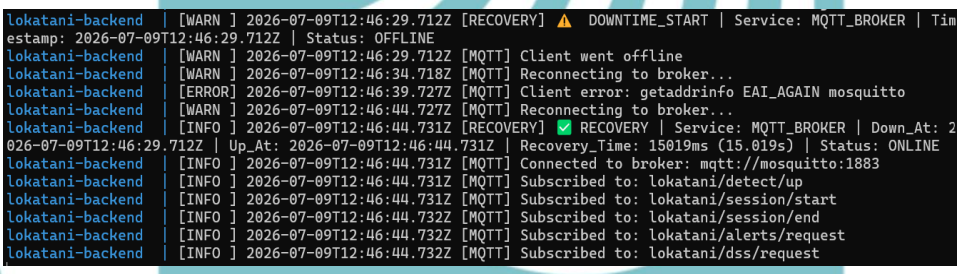
Gambar 4.70 Loop *Recovery* Otomatis Akibat Galat [Errno 101] pada WebSocket

Pesan eror seperti [Errno 101] *Network is unreachable* dan pengecualian *library Max retries exceeded* berhasil di-*handle* secara aman oleh arsitektur perangkat lunak *Client*, mencegah *thread* mati. Alih-alih mengalami *hang*, skrip membuang antrean gambar secara terkendali untuk mencegah RAM Raspberry Pi penuh, dan secara otonom masuk ke dalam siklus *polling reconnect* setiap 5 detik. Mekanisme pengamanan inilah yang memungkinkan sistem mencatatkan *Recovery Time* yang sangat singkat, yakni berkisar 2-3 detik sesaat setelah jaringan seluler dipulihkan.

3) Kekebalan Arsitektur *Microservices* terhadap Kegagalan Berantai

Evaluasi terakhir membedah kemampuan sistem dalam mencegah *cascading failure* saat layanan dependensi mengalami kelumpuhan paksa. Kondisi *uptime* pada *server* utama *backend* saat *broker* MQTT dan *database* dimatikan membuktikan sempurnanya isolasi kegagalan pada jaringan *docker bridge*.

Dinamika arsitektural yang paling menonjol pada pengujian ini adalah perbedaan *recovery time* antara MQTT yang mencatatkan waktu 15.019 detik dan *database* yang membutuhkan waktu hingga 18.044 detik. Waktu pemulihan pada MQTT dapat dianalisis secara akurat melalui rekaman log Node.js yang ditunjukkan pada Gambar 4.71.



Gambar 4.71 Upaya Restorasi *Persistent* Node.js Saat Layanan Mosquitto dimatikan

Pada Gambar 4.71 memperlihatkan bahwa selama kontainer *broker* mati, sistem *backend* secara aktif mendeteksi interupsi tersebut dan terus-menerus mencoba melakukan koneksi ulang. Durasi waktu sebesar 15.019 detik ini terjadi karena akumulasi dari waktu henti kontainer selama 10 detik ditambah dengan jeda waktu dari algoritma koneksi ulang sebesar 5 detik. Begitu kontainer selesai berjalan kembali, sistem langsung berhasil melakukan *TCP handshake* serta menyelesaikan proses *re-subscribe* ke berbagai topik perpesanan secara instan, sehingga waktu pemulihan murninya tercatat sangat efisien di angka 5.019 detik.

Sebaliknya, waktu pemulihan murni pada layanan *database* memakan waktu lebih lama, yaitu 8.044 detik. Hal ini disebabkan oleh mekanisme

penyambungan ulang PostgreSQL yang bersifat pasif. *Server backend* tidak melakukan pengecekan secara aktif, melainkan baru mencoba terhubung kembali ketika ada permintaan pengiriman data baru yang masuk dari perangkat *edge* setiap 3 detik. Waktu pemulihan ini juga semakin membengkak karena kontainer PostgreSQL membutuhkan waktu beban tambahan untuk menginisialisasi memori dan membaca berkas log internal sebelum jalurnya benar-benar siap menerima kumpulan koneksi yang baru. Meskipun terdapat perbedaan waktu, kombinasi resiliensi *microservices* dan arsitektur pemulihan otomatis ini tetap menjamin platform bebas dari kelumpuhan sistem secara menyeluruh.

D. Kesimpulan Evaluasi Komparatif Arsitektur

Protokol HTTP dinilai kurang efisien akibat pembengkakan teks *header* dan mekanisme pengiriman sinkron yang menahan waktu komputasi, meskipun sistem sudah menerapkan *TCP Keep-Alive*. Sementara itu, WebSocket tidak disarankan karena kewajiban proses enkripsi *XOR Masking* membebani prosesor pada perangkat pengirim, yang berdampak langsung pada tingginya fluktuasi *Jitter*. MQTT berhasil mengungguli keduanya melalui struktur data biner yang ringan dan mekanisme pengiriman asinkron, sehingga mampu menghasilkan *Throughput* maksimal tanpa membebani perangkat keras keras.

Keunggulan jalur pengiriman data ini ditopang dengan sempurna oleh ketahanan infrastruktur *microservices* pada *server*. Hasil pengujian operasional membuktikan tercapainya kesetimbangan suhu fisik perangkat, manajemen memori yang efisien sehingga terhindar dari *memory leak*, serta isolasi kontainer yang terbukti kebal terhadap *cascading failure*. Secara keseluruhan, penggabungan MQTT dan *microservices* berhasil menghasilkan sistem pertanian cerdas dengan tingkat toleransi kesalahan yang tinggi dan sangat andal untuk digunakan dalam jangka panjang di lapangan.

BAB V
PENUTUP

5.1.Kesimpulan

Berdasarkan hasil perancangan arsitektur, implementasi sistem, dan ekstraksi data komparatif empiris yang telah diuraikan pada bab sebelumnya, penelitian ini menghasilkan kesimpulan sebagai berikut:

1. Sistem *monitoring* deteksi hama berbasis *microservices* berhasil dirancang bangun dan divalidasi dengan tingkat keberhasilan 100% pada pengujian *Blackbox*. Pada *User Acceptance Test* (UAT), pengguna mengonfirmasi kepuasan terhadap operasional sistem di lapangan.
2. Ketiga protokol lapisan aplikasi (HTTP, WebSocket, dan MQTT) berhasil diimplementasikan secara terisolasi di dalam lingkungan *Docker Bridge Network*. Arsitektur ini mampu memfasilitasi transmisi dinamis data gambar Base64 dari perangkat *Edge* ke *server* Node.js dan Mosquitto secara stabil tanpa memicu konflik port.
3. Arsitektur sistem terbukti andal dengan mencapai Kesetimbangan suhu (50.6°C–55.0°C) dan stabilitas RAM (2-3%) selama 6 jam operasi tanpa kebocoran memori. Saat jaringan terputus, perangkat *Edge* mampu melakukan *Auto-Reconnect* dalam 2-3 detik. Infrastruktur *microservices* juga terbukti kebal terhadap *cascading failure*, dengan kemampuan memulihkan layanan secara otomatis dalam waktu 15.019 detik untuk layanan broker dan 18.044 detik untuk layanan *database*.
4. Evaluasi kinerja QoS mengonfirmasi MQTT memiliki rata-rata *Throughput* tertinggi (153.92 Kbps), diikuti WebSocket (152.95 Kbps) dan HTTP (150.88 Kbps). Tingkat *Packet Loss* seluruh protokol sangat ideal di bawah 1%, dengan HTTP mencatatkan angka kegagalan terkecil (0.26%). Untuk kecepatan waktu tempuh, WebSocket mencatatkan rata-rata tercepat (25.11 ms), disusul ketat oleh MQTT (26.10 ms), dan HTTP paling lambat (27.71 ms). Pada parameter *Jitter*, HTTP mencatatkan fluktuasi terendah (16.00

ms), sedangkan WebSocket memiliki fluktuasi jeda kedatangan paling tinggi (78.22 ms).

5. Penurunan kualitas pada metrik QoS terbukti berkaitan secara empiris dengan batasan arsitektur dari setiap protokol. Keterlambatan waktu tempuh pada HTTP dipicu oleh inefisiensi *application-layer bloat* dan mekanisme pengiriman yang sinkron. Lonjakan *Jitter* pada WebSocket merupakan bentuk hambatan fisik akibat kewajiban komputasi kriptografi *XOR Masking* di tingkat mikrokontroler. MQTT terbukti menjadi standar arsitektur yang paling optimal karena berhasil menghindari kedua kelemahan tersebut melalui penggunaan format data biner yang ringan dan mekanisme transmisi asinkron.

5.2. Saran

Berdasarkan batasan masalah dan temuan selama pengujian, rekomendasi untuk pengembangan penelitian selanjutnya adalah sebagai berikut:

1. Penelitian selanjutnya disarankan untuk mengintegrasikan data agronomi aktual yang telah divalidasi langsung oleh pakar pertanian. Hal ini bertujuan untuk menaikkan nilai guna praktis dari fitur sistem pendukung keputusan bagi para pengelola kebun di lapangan.
2. Mengingat evaluasi kinerja pada penelitian ini dieksekusi menggunakan infrastruktur Wi-Fi 5G berkapasitas tinggi dengan persentase kehilangan paket yang sangat rendah, penelitian masa depan perlu melakukan uji beban (*stress test*) menggunakan simulasi pembatasan kecepatan jaringan atau pada lingkungan seluler yang tidak stabil. Selain itu, perlu dilakukan pengukuran batas kapasitas maksimal *broker* MQTT dengan menghubungkan puluhan hingga ratusan perangkat *edge* secara bersamaan.
3. Untuk mengurangi beban komputasi *payload* pada perangkat keras, penelitian selanjutnya direkomendasikan untuk mengganti format teks Base64 dengan mekanisme serialisasi biner murni, seperti *Protocol Buffers* atau *MessagePack*. Ruang lingkup analisis juga dapat diperluas dengan mengevaluasi protokol alternatif yang dirancang khusus untuk komunikasi *microservices* berkinerja tinggi, seperti gRPC atau CoAP.

DAFTAR PUSTAKA

Amirkhanov, B., Amirkhanova, G., Kunelbayev, M., Adilzhanova, S., & Tokhtassyn, M. (2025). Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on Latency, stability, and integration. *International Journal of Innovative Research and Scientific Studies*, 8(1), 679-694.

An, S., Eck, T., & Yim, H. (2023). Understanding Consumers' Acceptance Intention to Use Mobile Food Delivery Applications through an Extended Technology Acceptance Model. *Sustainability*, 15(1).

Dinata, D. C. (2023). *Rancang Bangun Sistem Penyiraman Pestisida dan Deteksi Hama Menggunakan Metode YOLO*. Laporan Penelitian Politeknik Negeri Jakarta. Depok: Politeknik Negeri Jakarta.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, 195-216.

ETSI. (1999). *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS)*. ETSI TR 101 329 V2.1.1.

Tsaqief, M. F., & Sutopo, J. (2025). Comparative performance analysis between the MQTT and WebSocket protocols. *Bit-Tech*, 8(2), 2227-2237.

Gao, Y., et al. (2026). Mitigating GIL Bottlenecks in Edge AI Systems. *arXiv preprint arXiv:2601.10582*.

Grigorik, I. (2013). *High Performance Browser Networking: What every web developer should know about networking and web performance*. O'Reilly Media, Inc.

Hussain, S., et al. (2022). *Database* Connection Pooling in Modern Web Applications: Performance and Reliability Analysis. *Journal of Cloud Computing*.

Internet Engineering Task Force (IETF). (2011). *RFC 6455: The WebSocket Protocol*. [Online] Diakses dari https://datatracker.ietf.org/doc/html/rfc6455

Josefsson, S. (2006). *The Base16, Base32, and Base64 Data Encodings*. RFC 4648, Internet Engineering Task Force (IETF).

Kojansow, C. W., Manembu, P. D. K., & Rumagit, A. M. (2024). Internet Of Things (IoT) Platform Development using WebSocket communication. *Jurnal Teknik Informatika*, 19(3), 259-269.

Kuo, Y. H., et al. (2001). *Software Reliability Engineering: Testing and Quality Management*. John Wiley & Sons.

Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.

Lemire, D. (2019). *What is the space overhead of Base64 encoding?*. [Online] Diakses dari web log penulis.

Light, R. (2017). Mosquitto: server and client implementation of the MQTT protocol. *The Journal of Open Source Software*, 2(13), 265.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. *National Institute of Standards and Technology, Special Publication 800-145*.

Merenda, M., et al. (2020). Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors*, 20(9), 2533.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2.

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing* (3rd ed.). John Wiley & Sons.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc.

Ornbo, G. (2012). *Sams Teach Yourself Node.js in 24 Hours*. Sams Publishing.

Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.

Rofiansyah, W., Zalianty, F. R., Ito, F. A. L., Wijayanto, I., Ryanu, H. H., & Irawati, I. D. (2025). IoT-based control and monitoring system for hydroponic plant growth using image processing and mobile applications. *PeerJ Computer Science*. https://doi.org/10.7717/peerj-cs.2763

Satyanarayanan, M. (2017). The Emergence of Edge Computing. *Computer*, 50(1), 30-39.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill.

Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.

Sugiyono. (2022). *Metode Penelitian Kuantitatif, Kualitatif, dan R&D*. Bandung: Alfabeta.

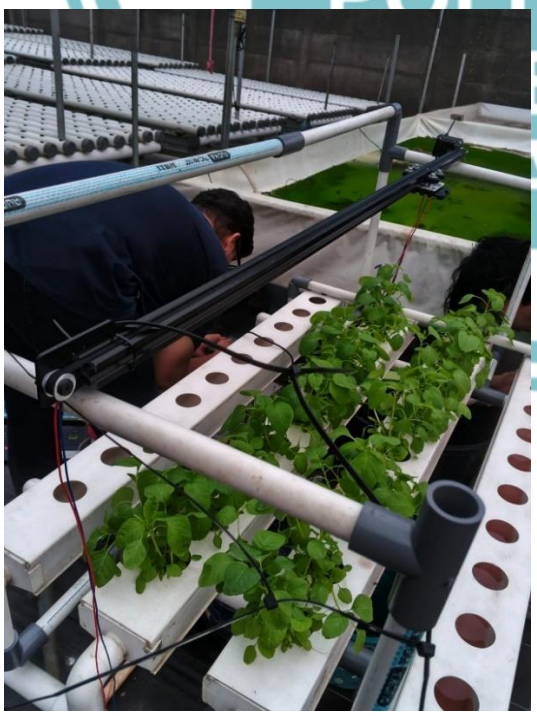
Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, 14(6), 80-83.

DAFTAR RIWAYAT HIDUP

Muhammad Khairu Mufid lahir di Depok tahun 2004. Penulis menyelesaikan pendidikan dasar di SDIT Mutiara Islam pada tahun 2016, dilanjutkan ke tingkat menengah di SMP Negeri 5 Depok dan lulus pada tahun 2019, lalu melanjutkan ke sekolah Tingkat atas di MA Negeri 7 Jakarta dengan peminatan Matematika dan Ilmu Pengetahuan Alam (MIPA), lulus pada tahun 2022. Saat ini, penulis sedang menyelesaikan pendidikan sarjana terapan di Politeknik Negeri Jakarta, Jurusan Teknik Informatika dan Komputer, Program Studi Teknik Multimedia dan Jaringan. Selama menempuh masa perkuliahan, penulis aktif berorganisasi dalam Kelompok Studi Mahasiswa (KSM) serta berpartisipasi dan mengembangkan kompetensi melalui perlombaan, baik di bidang akademik maupun non-akademik.

LAMPIRAN

Lampiran 1. Dokumentasi Observasi dan Implementasi



Lampiran 2. Instrumen dan Hasil *Raw Data User Acceptance Test* (UAT)

Siapa yang telah menjawab?

Email

najib290501@gmail.com

BAGIAN 1: IDENTITAS DAN DATA DEMOGRAFI

Nama Lengkap Responden

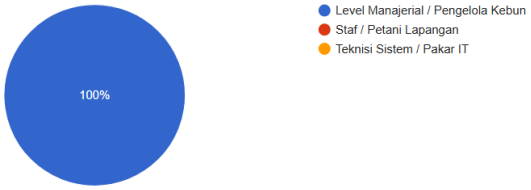
1 jawaban

Ainun Najib

Posisi / Peran Operasional

Salin diagram

1 jawaban



Tanggal dan Waktu Pengujian

1 jawaban

Jun 2026 | 23

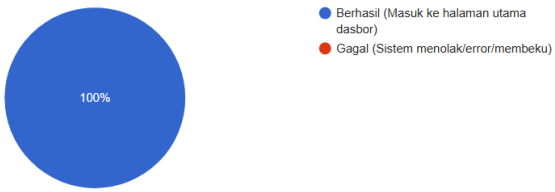
Bagian 2: Pengujian Eksplorasi Fungsional (Black-Box Testing)

Tugas 1: Autentikasi Pengguna (Login Sistem)

Salin diagram

Deskripsi Tugas: Masukkan *username* dan *password* yang diberikan pada halaman utama dasbor.

1 jawaban

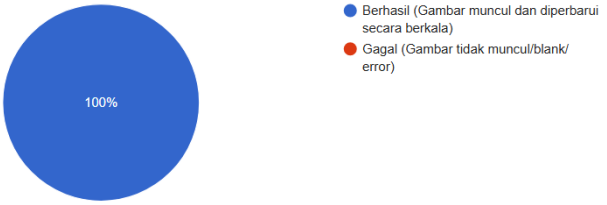


Tugas 2: Akses Pemantauan Langsung (Live Monitor)

Salin diagram

Deskripsi Tugas: Akses menu "Live Monitor", perhatikan aliran matriks citra (*video streaming*) dari kamera Raspberry Pi di kebun.

1 jawaban



Tugas 3: Validasi Trigger DSS & Rekomendasi Mitigasi

Salin diagram

Deskripsi Tugas: Hadapkan objek hama pada kamera. Amati apakah panel Sistem Pendukung Keputusan (DSS) otomatis muncul di samping layar saat kotak penanda (*bounding box*) YOLO mendeteksi hama.

1 jawaban

Tugas 4: Penelusuran Riwayat Laporan Pengujian

Salin diagram

Deskripsi Tugas: Akses menu "Laporan", cari log deteksi yang masuk hari ini, lalu klik tombol "Detail" untuk memuat ulang data foto serta koordinat bounding box.

1 jawaban

Tugas 5: Modifikasi Parameter Batas Sensitivitas (Confidence Threshold)

Salin diagram

Deskripsi Tugas: Akses menu "Pengaturan", ubah nilai batas deteksi (*Confidence Threshold*) menjadi angka 0.70, lalu klik tombol simpan.

1 jawaban

Tugas 6: Responsivitas Perangkat Bergerak (*Mobile Access*)

Salin diagram

Deskripsi Tugas: Gunakan peramban (*browser*) pada ponsel pintar Anda untuk membuka tautan dasbor Lokatani. Pastikan antarmuka menyesuaikan ukuran layar tanpa ada tombol yang terpotong.

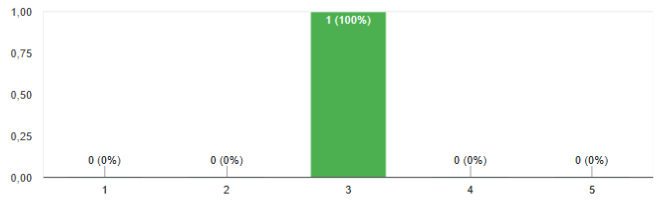
1 jawaban

Bagian 3: Penilaian Kualitas Kinerja dan Kepuasan Pengguna

[KEMUDAHAN NAVIGASI] Tampilan antarmuka dasbor mudah dipahami dan struktur menunya intuitif, bahkan bagi pengguna tanpa latar belakang IT ahli.

Salin diagram

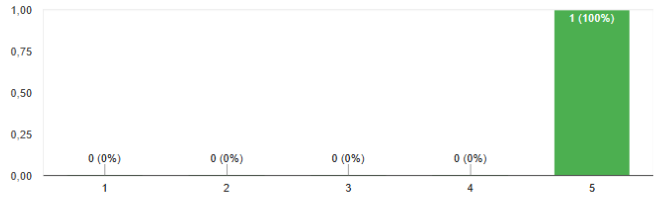
1 jawaban



[KINERJA JARINGAN] Kecepatan pembaruan data dan pengiriman gambar dari perangkat lapangan ke dasbor berjalan lancar tanpa tundaan (*delay*) yang mengganggu operasional.

Salin diagram

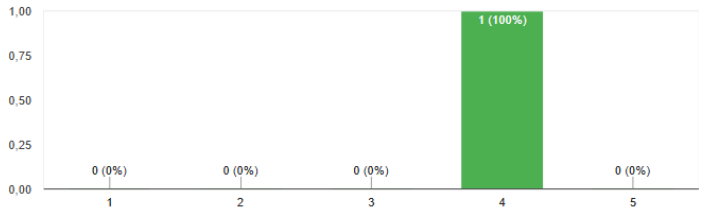
1 jawaban



[EFEKTIVITAS VISUAL] Fitur kotak penanda (*Bounding Box*) dan label persentase akurasi pada Live Monitor sangat membantu mengenali posisi hama secara instan.

Salin diagram

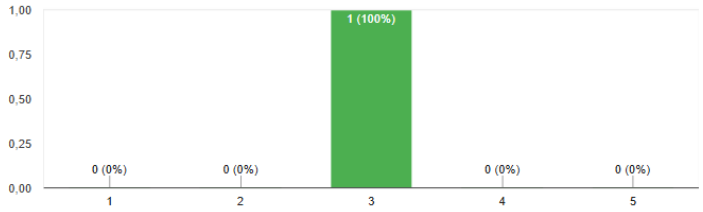
1 jawaban



[UTILITAS DATA DSS] Keberadaan modul panel informasi yang otomatis menampilkan instruksi penanganan mitigasi saat terjadi deteksi sangat membantu mempercepat proses pengambilan keputusan tindakan.

Salin diagram

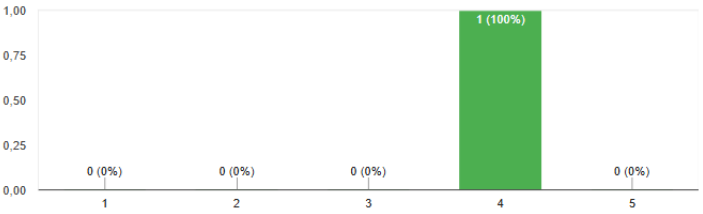
1 jawaban



[IMPLIKASI OPERASIONAL] Secara keseluruhan, implementasi platform pemantauan ini efektif meningkatkan efisiensi waktu manajemen kebun dibandingkan metode inspeksi manual.

Salin diagram

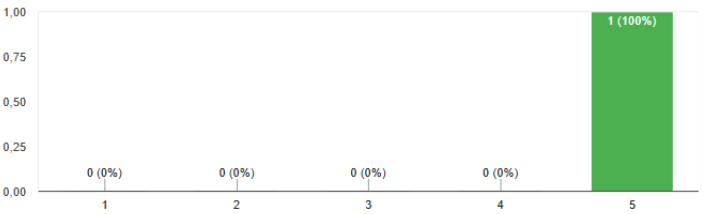
1 jawaban

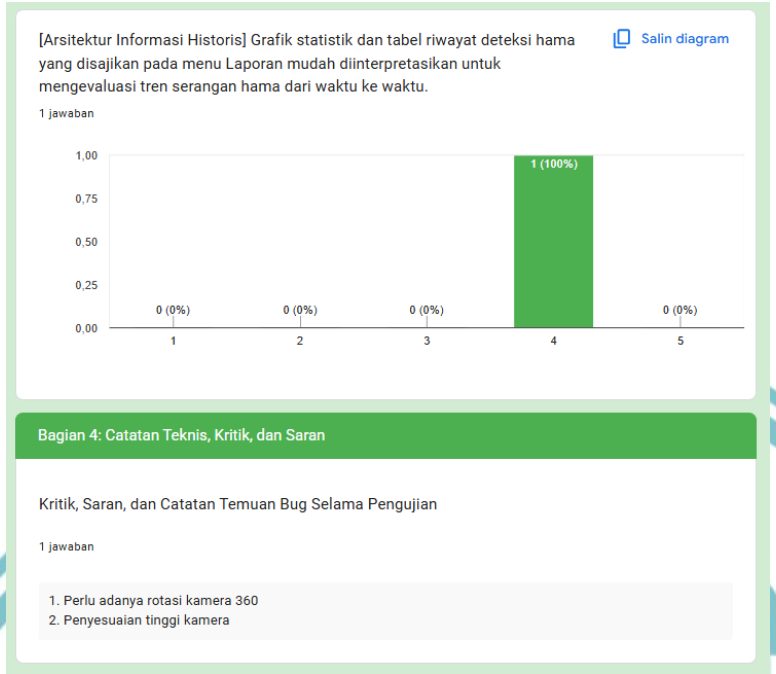


[STABILITAS & PENANGANAN GALAT]: Selama durasi pengujian, sistem dasbor berjalan dengan stabil tanpa mengalami pembekuan layar (*freeze*), keluar paksa (*crash*), atau proses memuat (*loading*) yang tidak berujung.

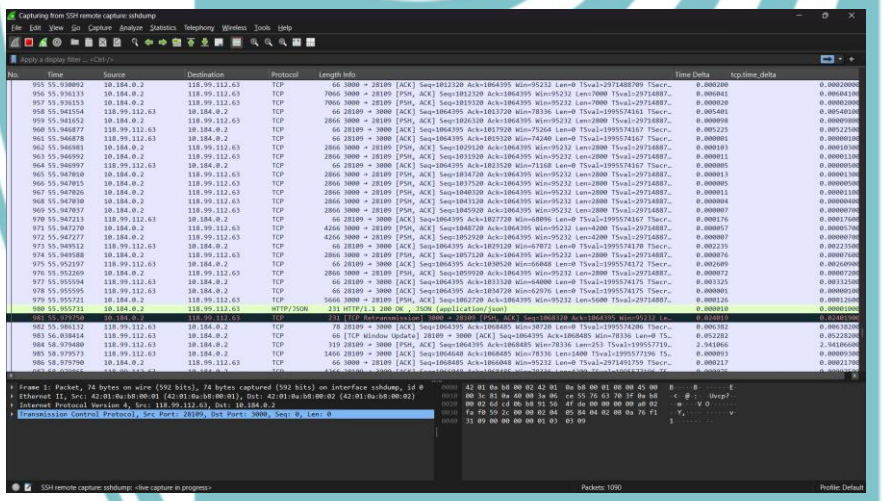
Salin diagram

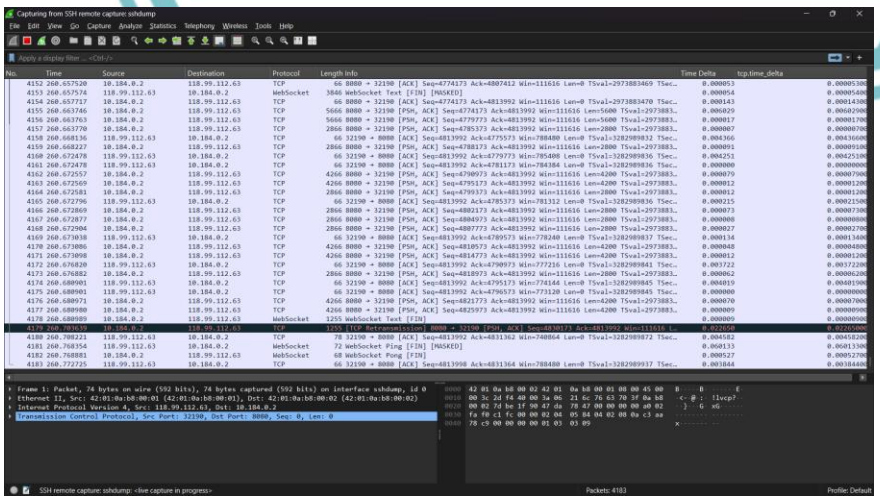
1 jawaban





Lampiran 3. Proses Packet Sniffing SSH Remote Wireshark





Lampiran 4. Data Mentah Pengujian *Quality of Service* (QoS)

protokol	waktu_kirim	waktu_terima	latency_ms
HTTP	2026-07-11 13:41:53.058+00	2026-07-11 13:41:53.225+00	167
HTTP	2026-07-11 13:41:56.139+00	2026-07-11 13:41:56.163+00	24
HTTP	2026-07-11 13:41:59.217+00	2026-07-11 13:41:59.242+00	25
HTTP	2026-07-11 13:42:02.264+00	2026-07-11 13:42:02.291+00	27
HTTP	2026-07-11 13:42:05.341+00	2026-07-11 13:42:05.367+00	26
HTTP	2026-07-11 13:42:08.41+00	2026-07-11 13:42:08.437+00	27
HTTP	2026-07-11 13:42:11.469+00	2026-07-11 13:42:11.494+00	25
HTTP	2026-07-11 13:42:14.534+00	2026-07-11 13:42:14.562+00	28
HTTP	2026-07-11 13:42:17.591+00	2026-07-11 13:42:17.618+00	27
HTTP	2026-07-11 13:42:20.671+00	2026-07-11 13:42:20.697+00	26
HTTP	2026-07-11 13:42:23.751+00	2026-07-11 13:42:23.775+00	24
HTTP	2026-07-11 13:42:26.821+00	2026-07-11 13:42:26.846+00	25
HTTP	2026-07-11 13:42:29.9+00	2026-07-11 13:42:29.927+00	27
HTTP	2026-07-11 13:42:32.981+00	2026-07-11 13:42:33.007+00	26
HTTP	2026-07-11 13:42:36.043+00	2026-07-11 13:42:36.071+00	28
HTTP	2026-07-11 13:42:39.102+00	2026-07-11 13:42:39.132+00	30
HTTP	2026-07-11 13:42:42.19+00	2026-07-11 13:42:42.219+00	29
HTTP	2026-07-11 13:42:45.247+00	2026-07-11 13:42:45.272+00	25
HTTP	2026-07-11 13:42:48.317+00	2026-07-11 13:42:48.343+00	26
HTTP	2026-07-11 13:42:51.383+00	2026-07-11 13:42:51.408+00	25
HTTP	2026-07-11 13:42:54.469+00	2026-07-11 13:42:54.494+00	25
HTTP	2026-07-11 13:42:57.541+00	2026-07-11 13:42:57.565+00	24
HTTP	2026-07-11 13:43:00.619+00	2026-07-11 13:43:00.647+00	28
HTTP	2026-07-11 13:43:03.692+00	2026-07-11 13:43:03.716+00	24
HTTP	2026-07-11 13:43:06.749+00	2026-07-11 13:43:06.775+00	26
HTTP	2026-07-11 13:43:09.808+00	2026-07-11 13:43:09.829+00	21
HTTP	2026-07-11 13:43:12.888+00	2026-07-11 13:43:12.914+00	26
HTTP	2026-07-11 13:43:15.946+00	2026-07-11 13:43:15.971+00	25
HTTP	2026-07-11 13:43:22.075+00	2026-07-11 13:43:22.1+00	25
HTTP	2026-07-11 13:43:25.157+00	2026-07-11 13:43:25.182+00	25
HTTP	2026-07-11 13:43:28.225+00	2026-07-11 13:43:28.25+00	25
HTTP	2026-07-11 13:43:31.287+00	2026-07-11 13:43:31.311+00	24
HTTP	2026-07-11 13:43:34.372+00	2026-07-11 13:43:34.397+00	25
HTTP	2026-07-11 13:43:37.437+00	2026-07-11 13:43:37.458+00	21
HTTP	2026-07-11 13:43:40.522+00	2026-07-11 13:43:40.548+00	26
HTTP	2026-07-11 13:43:43.59+00	2026-07-11 13:43:43.616+00	26
HTTP	2026-07-11 13:43:46.639+00	2026-07-11 13:43:46.665+00	26
HTTP	2026-07-11 13:43:49.701+00	2026-07-11 13:43:49.726+00	25
HTTP	2026-07-11 13:43:52.781+00	2026-07-11 13:43:52.808+00	27
HTTP	2026-07-11 13:43:55.849+00	2026-07-11 13:43:55.876+00	27

HTTP	2026-07-11 13:43:58.919+00	2026-07-11 13:43:58.945+00	26
HTTP	2026-07-11 13:44:01.98+00	2026-07-11 13:44:02.007+00	27
HTTP	2026-07-11 13:44:05.034+00	2026-07-11 13:44:05.057+00	23
HTTP	2026-07-11 13:44:08.111+00	2026-07-11 13:44:08.136+00	25
HTTP	2026-07-11 13:44:11.181+00	2026-07-11 13:44:11.209+00	28
HTTP	2026-07-11 13:44:14.245+00	2026-07-11 13:44:14.272+00	27
HTTP	2026-07-11 13:44:17.308+00	2026-07-11 13:44:17.333+00	25
HTTP	2026-07-11 13:44:20.364+00	2026-07-11 13:44:20.395+00	31
HTTP	2026-07-11 13:44:23.439+00	2026-07-11 13:44:23.461+00	22
HTTP	2026-07-11 13:44:26.517+00	2026-07-11 13:44:26.541+00	24
HTTP	2026-07-11 13:44:29.589+00	2026-07-11 13:44:29.616+00	27
HTTP	2026-07-11 13:44:32.651+00	2026-07-11 13:44:32.675+00	24
HTTP	2026-07-11 13:44:35.721+00	2026-07-11 13:44:35.751+00	30
HTTP	2026-07-11 13:44:38.794+00	2026-07-11 13:44:38.82+00	26
HTTP	2026-07-11 13:44:41.868+00	2026-07-11 13:44:41.896+00	28
HTTP	2026-07-11 13:44:44.929+00	2026-07-11 13:44:44.973+00	44
HTTP	2026-07-11 13:44:47.988+00	2026-07-11 13:44:48.018+00	30
HTTP	2026-07-11 13:44:51.041+00	2026-07-11 13:44:51.07+00	29
HTTP	2026-07-11 13:44:54.116+00	2026-07-11 13:44:54.141+00	25
HTTP	2026-07-11 13:44:57.202+00	2026-07-11 13:44:57.228+00	26
HTTP	2026-07-11 13:45:00.25+00	2026-07-11 13:45:00.275+00	25
HTTP	2026-07-11 13:45:03.319+00	2026-07-11 13:45:03.343+00	24
HTTP	2026-07-11 13:45:06.378+00	2026-07-11 13:45:06.406+00	28
HTTP	2026-07-11 13:45:09.453+00	2026-07-11 13:45:09.476+00	23
HTTP	2026-07-11 13:45:12.534+00	2026-07-11 13:45:12.56+00	26
HTTP	2026-07-11 13:45:15.602+00	2026-07-11 13:45:15.625+00	23
HTTP	2026-07-11 13:45:18.674+00	2026-07-11 13:45:18.699+00	25
HTTP	2026-07-11 13:45:21.75+00	2026-07-11 13:45:21.77+00	20
HTTP	2026-07-11 13:45:24.819+00	2026-07-11 13:45:24.845+00	26
HTTP	2026-07-11 13:45:27.896+00	2026-07-11 13:45:27.92+00	24
HTTP	2026-07-11 13:45:30.985+00	2026-07-11 13:45:31.012+00	27
HTTP	2026-07-11 13:45:34.058+00	2026-07-11 13:45:34.083+00	25
HTTP	2026-07-11 13:45:37.116+00	2026-07-11 13:45:37.145+00	29
HTTP	2026-07-11 13:45:40.206+00	2026-07-11 13:45:40.23+00	24
HTTP	2026-07-11 13:45:43.304+00	2026-07-11 13:45:43.324+00	20
HTTP	2026-07-11 13:45:46.426+00	2026-07-11 13:45:46.45+00	24
HTTP	2026-07-11 13:45:49.488+00	2026-07-11 13:45:49.515+00	27
HTTP	2026-07-11 13:45:52.561+00	2026-07-11 13:45:52.707+00	146
HTTP	2026-07-11 13:45:55.639+00	2026-07-11 13:45:55.665+00	26
HTTP	2026-07-11 13:45:58.715+00	2026-07-11 13:45:58.739+00	24
HTTP	2026-07-11 13:46:01.783+00	2026-07-11 13:46:01.811+00	28
HTTP	2026-07-11 13:46:04.848+00	2026-07-11 13:46:04.872+00	24
HTTP	2026-07-11 13:46:07.927+00	2026-07-11 13:46:07.953+00	26
HTTP	2026-07-11 13:46:11.006+00	2026-07-11 13:46:11.035+00	29
HTTP	2026-07-11 13:46:14.095+00	2026-07-11 13:46:14.122+00	27

HTTP	2026-07-11 13:46:17.17+00	2026-07-11 13:46:17.199+00	29
HTTP	2026-07-11 13:46:20.218+00	2026-07-11 13:46:20.242+00	24
HTTP	2026-07-11 13:46:23.294+00	2026-07-11 13:46:23.318+00	24
HTTP	2026-07-11 13:46:26.366+00	2026-07-11 13:46:26.39+00	24
HTTP	2026-07-11 13:46:29.423+00	2026-07-11 13:46:29.452+00	29
HTTP	2026-07-11 13:46:32.494+00	2026-07-11 13:46:32.522+00	28
HTTP	2026-07-11 13:46:35.581+00	2026-07-11 13:46:35.605+00	24
HTTP	2026-07-11 13:46:38.642+00	2026-07-11 13:46:38.667+00	25
HTTP	2026-07-11 13:46:41.708+00	2026-07-11 13:46:41.732+00	24
HTTP	2026-07-11 13:46:44.801+00	2026-07-11 13:46:44.824+00	23
MQTT	2026-07-11 14:12:25.433+00	2026-07-11 14:12:25.475+00	42
MQTT	2026-07-11 14:12:28.546+00	2026-07-11 14:12:28.57+00	24
MQTT	2026-07-11 14:12:31.63+00	2026-07-11 14:12:31.653+00	23
MQTT	2026-07-11 14:12:34.707+00	2026-07-11 14:12:34.73+00	23
MQTT	2026-07-11 14:12:37.782+00	2026-07-11 14:12:37.803+00	21
MQTT	2026-07-11 14:12:40.849+00	2026-07-11 14:12:40.879+00	30
MQTT	2026-07-11 14:12:43.924+00	2026-07-11 14:12:43.949+00	25
MQTT	2026-07-11 14:12:47.017+00	2026-07-11 14:12:47.04+00	23
MQTT	2026-07-11 14:12:50.093+00	2026-07-11 14:12:50.119+00	26
MQTT	2026-07-11 14:12:53.167+00	2026-07-11 14:12:53.191+00	24
MQTT	2026-07-11 14:12:56.248+00	2026-07-11 14:12:56.27+00	22
MQTT	2026-07-11 14:12:59.326+00	2026-07-11 14:12:59.351+00	25
MQTT	2026-07-11 14:13:02.399+00	2026-07-11 14:13:02.422+00	23
MQTT	2026-07-11 14:13:05.484+00	2026-07-11 14:13:05.575+00	91
MQTT	2026-07-11 14:13:08.574+00	2026-07-11 14:13:08.601+00	27
MQTT	2026-07-11 14:13:11.663+00	2026-07-11 14:13:11.689+00	26
MQTT	2026-07-11 14:13:14.758+00	2026-07-11 14:13:14.785+00	27
MQTT	2026-07-11 14:13:17.831+00	2026-07-11 14:13:17.855+00	24
MQTT	2026-07-11 14:13:20.911+00	2026-07-11 14:13:20.939+00	28
MQTT	2026-07-11 14:13:23.997+00	2026-07-11 14:13:24.023+00	26
MQTT	2026-07-11 14:13:27.085+00	2026-07-11 14:13:27.109+00	24
MQTT	2026-07-11 14:13:30.193+00	2026-07-11 14:13:30.219+00	26
MQTT	2026-07-11 14:13:33.224+00	2026-07-11 14:13:33.247+00	23
MQTT	2026-07-11 14:13:36.279+00	2026-07-11 14:13:36.303+00	24
MQTT	2026-07-11 14:13:39.31+00	2026-07-11 14:13:39.332+00	22
MQTT	2026-07-11 14:13:42.363+00	2026-07-11 14:13:42.388+00	25
MQTT	2026-07-11 14:13:45.41+00	2026-07-11 14:13:45.436+00	26
MQTT	2026-07-11 14:13:48.454+00	2026-07-11 14:13:48.478+00	24
MQTT	2026-07-11 14:13:51.524+00	2026-07-11 14:13:51.557+00	33
MQTT	2026-07-11 14:13:54.584+00	2026-07-11 14:13:54.61+00	26
MQTT	2026-07-11 14:13:57.665+00	2026-07-11 14:13:57.687+00	22
MQTT	2026-07-11 14:14:00.71+00	2026-07-11 14:14:00.735+00	25
MQTT	2026-07-11 14:14:03.764+00	2026-07-11 14:14:03.789+00	25
MQTT	2026-07-11 14:14:06.802+00	2026-07-11 14:14:06.822+00	20

MQTT	2026-07-11 14:14:09.85+00	2026-07-11 14:14:09.875+00	25
MQTT	2026-07-11 14:14:12.901+00	2026-07-11 14:14:12.925+00	24
MQTT	2026-07-11 14:14:15.932+00	2026-07-11 14:14:15.957+00	25
MQTT	2026-07-11 14:14:18.984+00	2026-07-11 14:14:19.009+00	25
MQTT	2026-07-11 14:14:22.027+00	2026-07-11 14:14:22.051+00	24
MQTT	2026-07-11 14:14:25.075+00	2026-07-11 14:14:25.098+00	23
MQTT	2026-07-11 14:14:28.144+00	2026-07-11 14:14:28.168+00	24
MQTT	2026-07-11 14:14:31.203+00	2026-07-11 14:14:31.23+00	27
MQTT	2026-07-11 14:14:34.231+00	2026-07-11 14:14:34.255+00	24
MQTT	2026-07-11 14:14:37.286+00	2026-07-11 14:14:37.31+00	24
MQTT	2026-07-11 14:14:40.343+00	2026-07-11 14:14:40.378+00	35
MQTT	2026-07-11 14:14:43.373+00	2026-07-11 14:14:43.398+00	25
MQTT	2026-07-11 14:14:46.428+00	2026-07-11 14:14:46.474+00	46
MQTT	2026-07-11 14:14:49.464+00	2026-07-11 14:14:49.49+00	26
MQTT	2026-07-11 14:14:52.519+00	2026-07-11 14:14:52.544+00	25
MQTT	2026-07-11 14:14:55.559+00	2026-07-11 14:14:55.584+00	25
MQTT	2026-07-11 14:14:58.623+00	2026-07-11 14:14:58.646+00	23
MQTT	2026-07-11 14:15:01.664+00	2026-07-11 14:15:01.688+00	24
MQTT	2026-07-11 14:15:04.721+00	2026-07-11 14:15:04.748+00	27
MQTT	2026-07-11 14:15:07.774+00	2026-07-11 14:15:07.799+00	25
MQTT	2026-07-11 14:15:10.807+00	2026-07-11 14:15:10.836+00	29
MQTT	2026-07-11 14:15:13.857+00	2026-07-11 14:15:13.882+00	25
MQTT	2026-07-11 14:15:16.929+00	2026-07-11 14:15:16.969+00	40
MQTT	2026-07-11 14:15:19.992+00	2026-07-11 14:15:20.023+00	31
MQTT	2026-07-11 14:15:23.054+00	2026-07-11 14:15:23.078+00	24
MQTT	2026-07-11 14:15:26.105+00	2026-07-11 14:15:26.125+00	20
MQTT	2026-07-11 14:15:29.152+00	2026-07-11 14:15:29.175+00	23
MQTT	2026-07-11 14:15:32.212+00	2026-07-11 14:15:32.236+00	24
MQTT	2026-07-11 14:15:35.27+00	2026-07-11 14:15:35.294+00	24
MQTT	2026-07-11 14:15:38.324+00	2026-07-11 14:15:38.349+00	25
MQTT	2026-07-11 14:15:41.368+00	2026-07-11 14:15:41.394+00	26
MQTT	2026-07-11 14:15:44.413+00	2026-07-11 14:15:44.435+00	22
MQTT	2026-07-11 14:15:47.453+00	2026-07-11 14:15:47.476+00	23
MQTT	2026-07-11 14:15:50.503+00	2026-07-11 14:15:50.528+00	25
MQTT	2026-07-11 14:15:53.563+00	2026-07-11 14:15:53.588+00	25
MQTT	2026-07-11 14:15:56.632+00	2026-07-11 14:15:56.657+00	25
MQTT	2026-07-11 14:15:59.682+00	2026-07-11 14:15:59.709+00	27
MQTT	2026-07-11 14:16:02.727+00	2026-07-11 14:16:02.75+00	23
MQTT	2026-07-11 14:16:05.795+00	2026-07-11 14:16:05.821+00	26
MQTT	2026-07-11 14:16:08.85+00	2026-07-11 14:16:08.874+00	24
MQTT	2026-07-11 14:16:11.886+00	2026-07-11 14:16:11.912+00	26
MQTT	2026-07-11 14:16:14.94+00	2026-07-11 14:16:14.965+00	25
MQTT	2026-07-11 14:16:17.999+00	2026-07-11 14:16:18.024+00	25
MQTT	2026-07-11 14:16:21.031+00	2026-07-11 14:16:21.053+00	22
MQTT	2026-07-11 14:16:24.07+00	2026-07-11 14:16:24.093+00	23

MQTT	2026-07-11 14:16:27.131+00	2026-07-11 14:16:27.155+00	24
MQTT	2026-07-11 14:16:30.19+00	2026-07-11 14:16:30.216+00	26
MQTT	2026-07-11 14:16:33.228+00	2026-07-11 14:16:33.253+00	25
MQTT	2026-07-11 14:16:36.277+00	2026-07-11 14:16:36.301+00	24
MQTT	2026-07-11 14:16:39.344+00	2026-07-11 14:16:39.367+00	23
MQTT	2026-07-11 14:16:42.386+00	2026-07-11 14:16:42.41+00	24
MQTT	2026-07-11 14:16:45.436+00	2026-07-11 14:16:45.464+00	28
MQTT	2026-07-11 14:16:48.484+00	2026-07-11 14:16:48.594+00	110
MQTT	2026-07-11 14:16:51.55+00	2026-07-11 14:16:51.575+00	25
MQTT	2026-07-11 14:16:54.581+00	2026-07-11 14:16:54.604+00	23
MQTT	2026-07-11 14:16:57.656+00	2026-07-11 14:16:57.681+00	25
MQTT	2026-07-11 14:17:00.701+00	2026-07-11 14:17:00.725+00	24
MQTT	2026-07-11 14:17:03.748+00	2026-07-11 14:17:03.776+00	28
MQTT	2026-07-11 14:17:06.782+00	2026-07-11 14:17:06.806+00	24
MQTT	2026-07-11 14:17:09.843+00	2026-07-11 14:17:09.899+00	56
WS	2026-07-11 14:18:27.903+00	2026-07-11 14:18:27.94+00	37
WS	2026-07-11 14:18:30.918+00	2026-07-11 14:18:30.939+00	21
WS	2026-07-11 14:18:34.02+00	2026-07-11 14:18:34.045+00	25
WS	2026-07-11 14:18:37.134+00	2026-07-11 14:18:37.156+00	22
WS	2026-07-11 14:18:40.243+00	2026-07-11 14:18:40.266+00	23
WS	2026-07-11 14:18:43.34+00	2026-07-11 14:18:43.365+00	25
WS	2026-07-11 14:18:46.44+00	2026-07-11 14:18:46.465+00	25
WS	2026-07-11 14:18:49.538+00	2026-07-11 14:18:49.56+00	22
WS	2026-07-11 14:18:52.643+00	2026-07-11 14:18:52.66+00	17
WS	2026-07-11 14:18:55.756+00	2026-07-11 14:18:55.778+00	22
WS	2026-07-11 14:18:58.757+00	2026-07-11 14:18:58.779+00	22
WS	2026-07-11 14:19:01.866+00	2026-07-11 14:19:01.887+00	21
WS	2026-07-11 14:19:04.982+00	2026-07-11 14:19:05.004+00	22
WS	2026-07-11 14:19:08.094+00	2026-07-11 14:19:08.114+00	20
WS	2026-07-11 14:19:11.2+00	2026-07-11 14:19:11.22+00	20
WS	2026-07-11 14:19:14.299+00	2026-07-11 14:19:14.32+00	21
WS	2026-07-11 14:19:17.395+00	2026-07-11 14:19:17.416+00	21
WS	2026-07-11 14:19:20.49+00	2026-07-11 14:19:20.522+00	32
WS	2026-07-11 14:19:23.574+00	2026-07-11 14:19:23.712+00	138
WS	2026-07-11 14:19:26.679+00	2026-07-11 14:19:26.703+00	24
WS	2026-07-11 14:19:29.807+00	2026-07-11 14:19:29.827+00	20
WS	2026-07-11 14:19:32.871+00	2026-07-11 14:19:32.89+00	19
WS	2026-07-11 14:19:35.894+00	2026-07-11 14:19:35.918+00	24
WS	2026-07-11 14:19:38.956+00	2026-07-11 14:19:38.981+00	25
WS	2026-07-11 14:19:42.007+00	2026-07-11 14:19:42.029+00	22
WS	2026-07-11 14:19:45.081+00	2026-07-11 14:19:45.102+00	21
WS	2026-07-11 14:19:48.132+00	2026-07-11 14:19:48.153+00	21
WS	2026-07-11 14:19:51.209+00	2026-07-11 14:19:51.23+00	21
WS	2026-07-11 14:19:54.271+00	2026-07-11 14:19:54.292+00	21

WS	2026-07-11 14:19:57.349+00	2026-07-11 14:19:57.369+00	20
WS	2026-07-11 14:20:00.438+00	2026-07-11 14:20:00.46+00	22
WS	2026-07-11 14:20:03.508+00	2026-07-11 14:20:03.528+00	20
WS	2026-07-11 14:20:06.564+00	2026-07-11 14:20:06.587+00	23
WS	2026-07-11 14:20:09.648+00	2026-07-11 14:20:09.666+00	18
WS	2026-07-11 14:20:12.719+00	2026-07-11 14:20:12.742+00	23
WS	2026-07-11 14:20:15.794+00	2026-07-11 14:20:15.816+00	22
WS	2026-07-11 14:20:18.856+00	2026-07-11 14:20:18.878+00	22
WS	2026-07-11 14:20:21.92+00	2026-07-11 14:20:21.944+00	24
WS	2026-07-11 14:20:24.987+00	2026-07-11 14:20:25.008+00	21
WS	2026-07-11 14:20:28.068+00	2026-07-11 14:20:28.086+00	18
WS	2026-07-11 14:20:31.135+00	2026-07-11 14:20:31.157+00	22
WS	2026-07-11 14:20:34.207+00	2026-07-11 14:20:34.232+00	25
WS	2026-07-11 14:20:37.272+00	2026-07-11 14:20:37.293+00	21
WS	2026-07-11 14:20:40.342+00	2026-07-11 14:20:40.374+00	32
WS	2026-07-11 14:20:43.398+00	2026-07-11 14:20:43.419+00	21
WS	2026-07-11 14:20:46.477+00	2026-07-11 14:20:46.503+00	26
WS	2026-07-11 14:20:49.542+00	2026-07-11 14:20:49.564+00	22
WS	2026-07-11 14:20:52.62+00	2026-07-11 14:20:52.64+00	20
WS	2026-07-11 14:20:55.698+00	2026-07-11 14:20:55.718+00	20
WS	2026-07-11 14:20:58.774+00	2026-07-11 14:20:58.796+00	22
WS	2026-07-11 14:21:01.833+00	2026-07-11 14:21:01.854+00	21
WS	2026-07-11 14:21:04.899+00	2026-07-11 14:21:04.92+00	21
WS	2026-07-11 14:21:07.968+00	2026-07-11 14:21:07.99+00	22
WS	2026-07-11 14:21:11.043+00	2026-07-11 14:21:11.065+00	22
WS	2026-07-11 14:21:14.115+00	2026-07-11 14:21:14.136+00	21
WS	2026-07-11 14:21:17.185+00	2026-07-11 14:21:17.206+00	21
WS	2026-07-11 14:21:20.264+00	2026-07-11 14:21:20.414+00	150
WS	2026-07-11 14:21:23.331+00	2026-07-11 14:21:23.352+00	21
WS	2026-07-11 14:21:26.401+00	2026-07-11 14:21:26.421+00	20
WS	2026-07-11 14:21:29.473+00	2026-07-11 14:21:29.492+00	19
WS	2026-07-11 14:21:32.53+00	2026-07-11 14:21:32.555+00	25
WS	2026-07-11 14:21:35.601+00	2026-07-11 14:21:35.621+00	20
WS	2026-07-11 14:21:38.682+00	2026-07-11 14:21:38.704+00	22
WS	2026-07-11 14:21:41.754+00	2026-07-11 14:21:41.775+00	21
WS	2026-07-11 14:21:44.806+00	2026-07-11 14:21:44.829+00	23
WS	2026-07-11 14:21:47.891+00	2026-07-11 14:21:47.912+00	21
WS	2026-07-11 14:21:50.962+00	2026-07-11 14:21:50.984+00	22
WS	2026-07-11 14:21:54.019+00	2026-07-11 14:21:54.04+00	21
WS	2026-07-11 14:21:57.086+00	2026-07-11 14:21:57.107+00	21
WS	2026-07-11 14:22:00.166+00	2026-07-11 14:22:00.184+00	18
WS	2026-07-11 14:22:03.232+00	2026-07-11 14:22:03.272+00	40
WS	2026-07-11 14:22:06.308+00	2026-07-11 14:22:06.326+00	18
WS	2026-07-11 14:22:09.379+00	2026-07-11 14:22:09.402+00	23
WS	2026-07-11 14:22:12.4+00	2026-07-11 14:22:12.418+00	18

WS	2026-07-11 14:22:15.464+00	2026-07-11 14:22:15.484+00	20
WS	2026-07-11 14:22:18.518+00	2026-07-11 14:22:18.539+00	21
WS	2026-07-11 14:22:21.609+00	2026-07-11 14:22:21.629+00	20
WS	2026-07-11 14:22:24.682+00	2026-07-11 14:22:24.706+00	24
WS	2026-07-11 14:22:27.744+00	2026-07-11 14:22:27.767+00	23
WS	2026-07-11 14:22:30.807+00	2026-07-11 14:22:30.827+00	20
WS	2026-07-11 14:22:33.865+00	2026-07-11 14:22:33.887+00	22
WS	2026-07-11 14:22:36.942+00	2026-07-11 14:22:36.965+00	23
WS	2026-07-11 14:22:40.01+00	2026-07-11 14:22:40.032+00	22
WS	2026-07-11 14:22:43.083+00	2026-07-11 14:22:43.104+00	21
WS	2026-07-11 14:22:46.145+00	2026-07-11 14:22:46.168+00	23
WS	2026-07-11 14:22:49.211+00	2026-07-11 14:22:49.233+00	22
WS	2026-07-11 14:22:52.283+00	2026-07-11 14:22:52.426+00	143
WS	2026-07-11 14:22:55.358+00	2026-07-11 14:22:55.383+00	25
WS	2026-07-11 14:22:58.431+00	2026-07-11 14:22:58.447+00	16
WS	2026-07-11 14:23:01.497+00	2026-07-11 14:23:01.519+00	22
WS	2026-07-11 14:23:04.559+00	2026-07-11 14:23:04.581+00	22
WS	2026-07-11 14:23:07.649+00	2026-07-11 14:23:07.672+00	23
WS	2026-07-11 14:23:10.714+00	2026-07-11 14:23:10.736+00	22
WS	2026-07-11 14:23:13.785+00	2026-07-11 14:23:13.803+00	18
WS	2026-07-11 14:23:16.847+00	2026-07-11 14:23:16.868+00	21

Lampiran 5. Data Hasil Pengujian Soak Testing

Timestamp	Host CPU Usage	Host RAM (MB)	Backend CPU	Backend RAM	DB CPU	DBR RAM
6/23/2026 12:26	1	896	0.01	2.77	0.03	1.02
6/23/2026 12:26	1	896	0.01	2.78	15.56	1.43
6/23/2026 12:26	5	912	0	2.77	0.03	1.44
6/23/2026 12:26	2	910	0	2.79	0.02	1.44
6/23/2026 12:26	2	913	0	2.78	0.28	1.44
6/23/2026 12:26	0	896	0	2.77	0.18	1.02
6/23/2026 12:26	55	996	0.41	2.78	0.06	1.02
6/23/2026 12:27	0	910	0.34	2.78	0.03	1.02
6/23/2026 12:27	1	920	0.02	2.78	0.14	1.34
6/23/2026 12:27	1	911	0	2.77	4.97	1.35
6/23/2026 12:27	0	912	0.29	2.78	0.14	1.34
6/23/2026 12:27	1	911	0.33	2.78	0.02	1.02
6/23/2026 12:27	1	903	0.03	2.78	0.17	1.03
6/23/2026 12:28	7	902	0.41	2.78	0.02	1.02
6/23/2026 12:28	1	899	0	2.78	0	1.02

6/23/2026 12:28	1	911	0.4	2.78	0.05	1.43
6/23/2026 12:28	1	911	0	2.78	0.02	1.44
6/23/2026 12:28	6	914	0	2.78	0.18	1.44
6/23/2026 12:28	2	900	0	2.78	0.29	1.02
6/23/2026 12:28	1	901	0	2.78	0.03	1.01
6/23/2026 12:29	1	899	0	2.78	4.73	1.02
6/23/2026 12:29	1	899	7.32	2.78	0	1.43
6/23/2026 12:29	2	912	0	2.78	0.03	1.43
6/23/2026 12:29	0	913	0.36	2.79	0.02	1.43
6/23/2026 12:29	1	916	0.28	2.78	0.03	1.43
6/23/2026 12:29	0	896	4.15	2.78	0.03	1.02
6/23/2026 12:30	2	898	0.01	2.77	0.2	1.03
6/23/2026 12:30	7	900	0.35	2.78	0.02	1.02
6/23/2026 12:30	1	908	0	2.78	0.03	1.44
6/23/2026 12:30	1	910	0.42	2.79	0.03	1.43
6/23/2026 12:30	1	911	0.01	2.79	4.9	1.44
6/23/2026 12:30	3	904	0	2.79	0.18	1.02
6/23/2026 12:30	1	903	0.34	2.78	0.01	1.02
6/23/2026 12:31	0	903	0	2.78	0.03	1.02
6/23/2026 12:31	1	897	0	2.78	14.25	1.43
6/23/2026 12:31	1	912	3.72	2.78	0	1.43
6/23/2026 12:31	0	907	0	2.78	0.04	1.43
6/23/2026 12:31	6	916	0.15	2.79	0.03	1.43
6/23/2026 12:31	1	902	0.42	2.78	0.03	1.01
6/23/2026 12:31	0	898	0.41	2.78	0	1.02
6/23/2026 12:32	1	898	0.01	2.79	5.36	1.02
6/23/2026 12:32	1	910	0.35	2.79	0.01	1.43
6/23/2026 12:32	0	916	0	2.78	0.03	1.44
6/23/2026 12:32	1	916	0.38	2.79	0.02	1.44
6/23/2026 12:32	1	916	0.01	2.78	0.16	1.02
6/23/2026 12:32	6	893	0.03	2.78	0.2	1.02
6/23/2026 12:33	9	897	0.45	2.78	0	1.01
6/23/2026 12:33	6	901	0	2.78	0.14	1.02
6/23/2026 12:33	1	906	0	2.78	0.02	1.43
6/23/2026 12:33	1	903	0.01	2.78	0	1.44
6/23/2026 12:33	1	907	0.45	2.78	5	1.44
6/23/2026 12:33	1	904	0.44	2.78	0	1.03
6/23/2026 12:33	1	903	0.26	2.78	0.02	1.02
6/23/2026 12:34	1	905	0	2.78	0	1.03
6/23/2026 12:34	2	903	0	2.79	0.2	1.44
6/23/2026 12:34	2	915	0	2.78	0.14	1.44
6/23/2026 12:34	0	900	0	2.78	0	1.44
6/23/2026 12:34	5	903	0	2.78	0.03	1.44
6/23/2026 12:34	1	893	3.59	2.78	0	1.02
6/23/2026 12:35	2	891	0	2.78	0.02	1.03

6/23/2026 12:35	2	885	0	2.79	0.23	1.02
6/23/2026 12:35	2	897	0	2.78	4.86	1.4
6/23/2026 12:35	0	897	0.34	2.8	0	1.4
6/23/2026 12:35	0	902	0	2.79	0.07	1.4
6/23/2026 12:35	0	899	0.01	2.78	0.18	1.02
6/23/2026 12:35	2	900	0	2.79	0	1.02
6/23/2026 12:36	2	904	0	2.79	0.19	1.03
6/23/2026 12:36	0	901	0.34	2.79	0.82	1.43
6/23/2026 12:36	10	912	0.03	2.78	0.32	1.44
6/23/2026 12:36	2	914	0	2.78	0	1.44
6/23/2026 12:36	0	915	0	2.79	0.02	1.44
6/23/2026 12:36	8	907	0.01	2.78	0	1.02
6/23/2026 12:36	1	915	0.01	2.79	5.03	1.02
6/23/2026 12:37	0	913	0	2.79	0.14	1.02
6/23/2026 12:37	0	921	0.34	2.79	0.18	1.44
6/23/2026 12:37	1	919	0	2.79	0	1.44
6/23/2026 12:37	0	923	0	2.79	0.04	1.44
6/23/2026 12:37	1	926	0.5	2.8	0.19	1.02
6/23/2026 12:37	5	910	0.01	2.79	1.71	1.03
6/23/2026 12:38	6	910	0	2.78	0.02	1.03
6/23/2026 12:38	0	907	0	2.78	0.16	1.02
6/23/2026 12:38	1	916	0	2.78	0.03	1.44
6/23/2026 12:38	1	913	0.34	2.79	4.85	1.45
6/23/2026 12:38	1	913	0.36	2.79	0.17	1.44
6/23/2026 12:38	1	910	0.36	2.8	0	1.02
6/23/2026 12:38	1	904	0.01	2.79	0.19	1.03
6/23/2026 12:39	1	906	0.32	2.79	0.03	1.03
6/23/2026 12:39	15	910	0.02	2.78	0	1.35
6/23/2026 12:39	1	912	0.01	2.79	0.03	1.34
6/23/2026 12:39	5	903	0	2.78	0.78	1.35
6/23/2026 12:39	1	907	0	2.78	3.14	1.02
6/23/2026 12:39	1	893	0	2.78	0	1.03
6/23/2026 12:40	1	895	0	2.77	4.75	1.03
6/23/2026 12:40	1	891	0	2.78	0	1.03
6/23/2026 12:40	1	904	0	2.78	0.05	1.44
6/23/2026 12:40	2	911	0	2.78	0.03	1.45
6/23/2026 12:40	1	913	0.42	2.78	0	1.43
6/23/2026 12:40	1	909	0.3	2.78	0.03	1.02
6/23/2026 12:40	0	908	0.32	2.78	0	1.03
6/23/2026 12:41	6	912	0.03	2.78	0.18	1.03
6/23/2026 12:41	1	906	0.42	2.78	0.77	1.44
6/23/2026 12:41	1	920	3.83	2.78	0.15	1.44
6/23/2026 12:41	1	915	0.3	2.79	5.06	1.44
6/23/2026 12:41	1	918	0.01	2.79	0.22	1.44
6/23/2026 12:41	1	909	0	2.78	0.17	1.03

6/23/2026 12:41	1	907	0.43	2.79	0	1.03
6/23/2026 12:42	1	910	0	2.78	0.1	1.02
6/23/2026 12:42	1	907	0.77	2.79	0	1.39
6/23/2026 12:42	0	918	0	2.78	0.12	1.39
6/23/2026 12:42	6	909	0	2.78	0.01	1.39
6/23/2026 12:42	0	914	0.39	2.79	0.03	1.4
6/23/2026 12:42	2	904	3.2	2.79	1.76	1.02
6/23/2026 12:43	1	904	0	2.78	5.02	1.03
6/23/2026 12:43	1	908	0	2.78	0.18	1.03
6/23/2026 12:43	1	914	0.42	2.79	0	1.44
6/23/2026 12:43	1	903	0	2.78	0.03	1.44
6/23/2026 12:43	1	904	0.33	2.78	0	1.43
6/23/2026 12:43	1	904	0.01	2.78	0.17	1.03
6/23/2026 12:43	1	902	0.31	2.78	0.24	1.03
6/23/2026 12:44	3	905	0.31	2.78	0.03	1.03
6/23/2026 12:44	3	905	0	2.78	0.07	1.03
6/23/2026 12:44	5	904	0	2.78	0.04	1.02
6/23/2026 12:44	1	902	0	2.79	0.76	1.03
6/23/2026 12:44	1	900	0	2.78	5.09	1.03
6/23/2026 12:44	1	902	0.4	2.78	0	1.02
6/23/2026 12:45	3	899	0	2.78	0.06	1.02
6/23/2026 12:45	2	897	0	2.78	0.02	1.02
6/23/2026 12:45	2	892	0	2.78	0.01	1.03
6/23/2026 12:45	1	893	0	2.78	0.03	1.03
6/23/2026 12:45	1	895	0	2.79	0	1.03
6/23/2026 12:45	5	891	0.32	2.79	0.03	1.02
6/23/2026 12:45	1	892	0.01	2.78	0.13	1.03
6/23/2026 12:46	11	898	0.4	2.79	0.17	1.44
6/23/2026 12:46	0	898	0.41	2.78	5.49	1.47
6/23/2026 12:46	0	902	0.02	2.78	0.2	1.46
6/23/2026 12:46	1	905	0	2.78	0.44	1.43
6/23/2026 12:46	2	897	0	2.8	0.17	1.43
6/23/2026 12:46	2	889	0	2.78	0.02	1.02
6/23/2026 12:46	1	911	0.35	2.79	0	1.03
6/23/2026 12:47	1	898	0.26	2.79	0.03	1.03
6/23/2026 12:47	6	902	0	2.79	0	1.44
6/23/2026 12:47	1	903	0	2.79	0.05	1.44
6/23/2026 12:47	0	897	0.41	2.79	0.03	1.44
6/23/2026 12:47	0	897	0.61	2.79	4.86	1.03
6/23/2026 12:47	1	888	3.79	2.79	0.02	1.03
6/23/2026 12:48	1	891	0	2.78	0	1.02
6/23/2026 12:48	0	891	0.33	2.79	0.03	1.03
6/23/2026 12:48	0	896	0	2.79	0.03	1.43
6/23/2026 12:48	1	903	0.01	2.8	0.12	1.44
6/23/2026 12:48	1	902	0	2.79	0	1.44

6/23/2026 12:48	6	881	0	2.79	0.18	1.03
6/23/2026 12:48	1	882	1.26	2.79	0.02	1.02
6/23/2026 12:49	1	881	0.01	2.8	0.03	1.03
6/23/2026 12:49	6	884	0.43	2.79	4.87	1.44
6/23/2026 12:49	1	894	4.01	2.79	0.16	1.44
6/23/2026 12:49	2	897	0	2.79	0.9	1.44
6/23/2026 12:49	1	898	0	2.8	3.68	1.03
6/23/2026 12:49	1	890	0	2.79	0.02	1.03
6/23/2026 12:50	1	891	0	2.79	0.01	1.03
6/23/2026 12:50	2	894	0	2.8	0.03	1.02
6/23/2026 12:50	6	891	3.79	2.8	0	1.3
6/23/2026 12:50	0	897	0	2.79	0.03	1.31
6/23/2026 12:50	1	900	0.01	2.8	0.02	1.31
6/23/2026 12:50	1	893	0	2.79	5.04	1.31
6/23/2026 12:50	9	887	0.41	2.79	0.17	1.03
6/23/2026 12:51	1	888	0.58	2.8	0	1.02
6/23/2026 12:51	0	898	0.01	2.8	0.79	1.44
6/23/2026 12:51	2	900	0.18	2.79	0.13	1.44
6/23/2026 12:51	1	904	0	2.8	0.18	1.44
6/23/2026 12:51	1	904	0.27	2.8	0	1.44
6/23/2026 12:51	0	888	0.01	2.8	0	1.03
6/23/2026 12:51	5	881	0	2.8	0.02	1.02
6/23/2026 12:52	3	877	0	2.79	0.01	1.03
6/23/2026 12:52	2	885	0	2.8	0.03	1.3
6/23/2026 12:52	1	887	0.34	2.8	5.19	1.31
6/23/2026 12:52	1	883	0.01	2.79	0.03	1.31
6/23/2026 12:52	1	889	0.01	2.79	0.03	1.03
6/23/2026 12:52	2	877	0	2.79	0.16	1.03
6/23/2026 12:53	1	876	0.39	2.8	0.03	1.02
6/23/2026 12:53	0	874	0.29	2.8	0	1.02
6/23/2026 12:53	1	891	0.01	2.79	0.1	1.44
6/23/2026 12:53	5	890	0.16	2.8	0.18	1.44
6/23/2026 12:53	1	886	0.01	2.8	0.18	1.45
6/23/2026 12:53	1	879	0.31	2.79	0.03	1.02
6/23/2026 12:53	2	878	0.01	2.79	4.75	1.03
6/23/2026 12:54	1	875	0	2.79	0.16	1.03
6/23/2026 12:54	1	875	7.69	2.8	0.01	1.44
6/23/2026 12:54	5	890	3.74	2.8	0.02	1.45
6/23/2026 12:54	1	890	0	2.8	0	1.45
6/23/2026 12:54	1	890	0.33	2.8	4.14	1.03
6/23/2026 12:54	1	876	0.01	2.79	0.21	1.03
6/23/2026 12:55	6	878	0.32	2.8	0.01	1.03
6/23/2026 12:55	3	877	0.3	2.8	0.03	1.02
6/23/2026 12:55	2	890	0	2.79	0.07	1.44
6/23/2026 12:55	0	890	0	2.79	5.28	1.44

6/23/2026 12:55	1	901	0	2.8	0.15	1.44
6/23/2026 12:55	4	893	0	2.79	0.02	1.03
6/23/2026 12:55	1	893	4.25	2.79	0	1.03
6/23/2026 12:56	1	884	0	2.79	0.02	1.03
6/23/2026 12:56	0	886	0	2.79	11.26	1.35
6/23/2026 12:56	1	900	0.01	2.8	0.01	1.35
6/23/2026 12:56	6	901	0.37	2.79	0.02	1.35
6/23/2026 12:56	1	899	0	2.8	0.03	1.35
6/23/2026 12:56	1	893	0.46	2.79	0.02	1.02
6/23/2026 12:56	2	909	0.02	2.79	5.68	1.02
6/23/2026 12:57	1	903	0.01	2.79	0.14	1.03
6/23/2026 12:57	1	912	0.34	2.8	0.03	1.44
6/23/2026 12:57	5	901	0	2.8	0.09	1.44
6/23/2026 12:57	0	892	0	2.79	0.02	1.45
6/23/2026 12:57	1	898	0	2.79	0.19	1.03
6/23/2026 12:57	1	886	0.44	2.79	0.03	1.02
6/23/2026 12:58	5	894	0.43	2.8	0	1.03
6/23/2026 12:58	1	886	0	2.79	0.02	1.03
6/23/2026 12:58	2	897	0	2.8	0.02	1.44
6/23/2026 12:58	1	895	0.02	2.8	5.1	1.44
6/23/2026 12:58	1	899	0	2.79	0	1.45
6/23/2026 12:58	1	892	0	2.8	0.05	1.03
6/23/2026 12:58	1	891	0	2.79	0.02	1.03
6/23/2026 12:59	1	893	0	2.79	0.14	1.03
6/23/2026 12:59	0	893	7.76	2.8	0.02	1.44
6/23/2026 12:59	0	902	0	2.8	0	1.45
6/23/2026 12:59	3	905	0.01	2.79	0.09	1.44
6/23/2026 12:59	4	903	0	2.8	0.16	1.44
6/23/2026 12:59	1	895	0	2.79	0.18	1.03
6/23/2026 13:00	1	893	0.35	2.79	0	1.03
6/23/2026 13:00	2	890	0.19	2.8	5.09	1.04
6/23/2026 13:00	1	903	0	2.8	0.03	1.45
6/23/2026 13:00	3	906	0	2.8	0.17	1.45
6/23/2026 13:00	1	899	0	2.8	0.04	1.44
6/23/2026 13:00	3	895	0	2.8	0	1.03
6/23/2026 13:00	0	891	0	2.8	0.02	1.03
6/23/2026 13:01	1	890	0	2.8	0.01	1.03
6/23/2026 13:01	5	889	0	2.8	11.98	1.44
6/23/2026 13:01	1	899	0.31	2.8	0.16	1.44
6/23/2026 13:01	0	904	0	2.8	0	1.44
6/23/2026 13:01	1	899	0	2.8	5.03	1.44
6/23/2026 13:01	0	881	0	2.8	0.17	1.02
6/23/2026 13:01	1	881	0.37	2.8	0.18	1.02
6/23/2026 13:02	2	879	0.4	2.79	0	1.02
6/23/2026 13:02	1	891	0.01	2.79	0.03	1.44

6/23/2026 13:02	2	897	3.94	2.8	0.01	1.45
6/23/2026 13:02	1	897	0	2.8	0.17	1.44
6/23/2026 13:02	6	896	0.27	2.8	0.02	1.03
6/23/2026 13:02	1	876	0.01	2.79	0	1.03
6/23/2026 13:03	1	885	0	2.8	0.03	1.03
6/23/2026 13:03	1	884	0	2.79	5.03	1.03
6/23/2026 13:03	1	893	0	2.79	0.03	1.39
6/23/2026 13:03	1	889	0.1	2.8	0	1.4
6/23/2026 13:03	1	898	0.01	2.8	0.03	1.4
6/23/2026 13:03	1	884	0.01	2.79	0.01	1.03
6/23/2026 13:03	5	880	0.01	2.8	0.18	1.03
6/23/2026 13:04	1	882	0.3	2.8	0.01	1.03
6/23/2026 13:04	5	881	7.96	2.8	0.01	1.44
6/23/2026 13:04	0	893	0	2.8	0.02	1.45
6/23/2026 13:04	1	902	0.87	2.8	0.15	1.44
6/23/2026 13:04	0	897	0	2.79	5.06	1.45
6/23/2026 13:04	1	890	0.32	2.8	0.01	1.03
6/23/2026 13:05	1	888	0.03	2.79	0.18	1.03
6/23/2026 13:05	2	889	0	2.79	0	1.03
6/23/2026 13:05	1	899	0	2.79	0.03	1.44
6/23/2026 13:05	6	902	0	2.79	0.01	1.44
6/23/2026 13:05	1	899	0	2.79	0.01	1.45
6/23/2026 13:05	8	888	0.01	2.8	0.16	1.03
6/23/2026 13:05	1	889	0.01	2.8	0	1.03
6/23/2026 13:06	1	890	0	2.79	0.04	1.03
6/23/2026 13:06	1	892	0.33	2.8	21.3	1.44
6/23/2026 13:06	1	894	0.17	2.8	0.03	1.45
6/23/2026 13:06	1	900	0	2.79	0.21	1.44
6/23/2026 13:06	1	899	0.16	2.8	0.22	1.44
6/23/2026 13:06	1	891	0.22	2.79	0	1.02
6/23/2026 13:06	1	877	0.37	2.8	0.02	1.04
6/23/2026 13:07	2	876	0.01	2.79	0.16	1.04
6/23/2026 13:07	5	884	0	2.79	0	1.44
6/23/2026 13:07	2	895	0	2.79	0.17	1.45
6/23/2026 13:07	1	900	0.32	2.8	0	1.44
6/23/2026 13:07	0	900	0	2.79	4.77	1.03
6/23/2026 13:07	1	888	0.05	2.79	0	1.03
6/23/2026 13:08	1	892	0	2.79	0.04	1.03
6/23/2026 13:08	1	890	0	2.79	0	1.03
6/23/2026 13:08	1	899	0.44	2.8	0.02	1.44
6/23/2026 13:08	1	900	0	2.8	0	1.44
6/23/2026 13:08	1	889	0	2.8	0.26	1.44
6/23/2026 13:08	1	878	0	2.8	0.16	1.04
6/23/2026 13:08	5	885	4.31	2.8	0.03	1.03
6/23/2026 13:09	0	882	0.3	2.8	0	1.03

6/23/2026 13:09	1	882	7.3	2.8	0.21	1.45
6/23/2026 13:09	2	885	0	2.79	5.23	1.45
6/23/2026 13:09	1	882	0	2.79	0.2	1.45
6/23/2026 13:09	0	882	0	2.79	0.16	1.45
6/23/2026 13:09	0	880	0	2.8	0.02	1.03
6/23/2026 13:10	2	880	0.03	2.79	0.03	1.03
6/23/2026 13:10	1	883	0	2.79	0	1.04
6/23/2026 13:10	1	889	0	2.79	0.03	1.45
6/23/2026 13:10	11	891	0.01	2.8	0.23	1.45
6/23/2026 13:10	1	884	0	2.79	0	1.44
6/23/2026 13:10	2	885	0.86	2.79	0.04	1.03
6/23/2026 13:10	1	878	0	2.8	6.39	1.04
6/23/2026 13:11	1	870	0	2.8	0.04	1.03
6/23/2026 13:11	1	874	0	2.79	14.83	1.45
6/23/2026 13:11	2	885	0	2.79	0.22	1.45
6/23/2026 13:11	0	890	0.41	2.8	0	1.45
6/23/2026 13:11	1	890	0.01	2.8	0.03	1.45
6/23/2026 13:11	1	893	0.01	2.8	0.19	1.03
6/23/2026 13:11	9	890	0	2.79	0	1.03
6/23/2026 13:12	1	892	0.01	2.8	0.02	1.03
6/23/2026 13:12	1	897	0	2.8	0	1.45
6/23/2026 13:12	3	893	0	2.81	5.18	1.44
6/23/2026 13:12	1	893	0	2.81	0.94	1.45
6/23/2026 13:12	0	894	0	2.8	0.15	1.04
6/23/2026 13:12	1	878	0.38	2.8	0	1.03
6/23/2026 13:13	7	882	0	2.81	0.04	1.04
6/23/2026 13:13	0	879	0	2.81	0.22	1.03
6/23/2026 13:13	2	892	0	2.81	0.03	1.44
6/23/2026 13:13	5	884	0.26	2.81	0.18	1.45
6/23/2026 13:13	1	880	0.34	2.81	0.04	1.45
6/23/2026 13:13	0	874	0.01	2.81	0.21	1.04
6/23/2026 13:13	1	874	0.46	2.8	5.3	1.04
6/23/2026 13:14	1	872	0	2.81	0.03	1.03
6/23/2026 13:14	3	876	7.73	2.81	0.77	1.45
6/23/2026 13:14	1	887	0	2.8	0.02	1.45
6/23/2026 13:14	1	896	0	2.8	0.16	1.45
6/23/2026 13:14	1	889	0	2.8	0.05	1.45
6/23/2026 13:14	1	873	0	2.8	0.03	1.04
6/23/2026 13:15	6	874	0	2.8	0	1.03
6/23/2026 13:15	0	884	0	2.81	0.02	1.04
6/23/2026 13:15	0	887	0	2.8	0	1.45
6/23/2026 13:15	1	895	3.81	2.81	4.73	1.39
6/23/2026 13:15	1	895	0	2.81	0.96	1.43
6/23/2026 13:15	3	881	0	2.81	0.04	1.03
6/23/2026 13:15	2	885	0.01	2.8	0.81	1.03

6/23/2026 13:16	2	886	0	2.81	0.06	1.03
6/23/2026 13:16	1	886	0.37	2.8	14.68	1.45
6/23/2026 13:16	1	888	0	2.81	0.04	1.47
6/23/2026 13:16	6	892	0.02	2.81	0.02	1.47
6/23/2026 13:16	2	887	0	2.8	0.2	1.48
6/23/2026 13:16	1	892	0	2.8	0.02	1.39
6/23/2026 13:16	1	913	4.23	2.8	5.43	1.41
6/23/2026 13:17	4	901	0.01	2.81	0.03	1.46
6/23/2026 13:17	1	901	0.9	2.81	0	1.46
6/23/2026 13:17	1	903	0	2.81	0.17	1.46
6/23/2026 13:17	1	900	0	2.8	0.83	1.46
6/23/2026 13:17	1	900	0	2.8	0	1.46
6/23/2026 13:17	1	895	0.01	2.81	0.02	1.03
6/23/2026 13:18	13	894	0.01	2.8	0.16	1.04
6/23/2026 13:18	1	887	0	2.8	0.03	1.03
6/23/2026 13:18	1	898	0.14	2.8	0.04	1.44
6/23/2026 13:18	5	896	0	2.81	4.09	1.45
6/23/2026 13:18	0	893	0.01	2.8	3.36	1.45
6/23/2026 13:18	2	886	0	2.8	0.16	1.03
6/23/2026 13:18	2	882	0	2.8	0.03	1.03
6/23/2026 13:19	1	884	0	2.8	0	1.03
6/23/2026 13:19	2	886	6.15	2.81	0.82	1.44
6/23/2026 13:19	1	899	0	2.8	0	1.45
6/23/2026 13:19	1	899	0.01	2.81	0.17	1.45
6/23/2026 13:19	5	890	0	2.8	0.01	1.45
6/23/2026 13:19	1	884	0	2.8	0.01	1.04
6/23/2026 13:20	0	884	0	2.8	0.02	1.03
6/23/2026 13:20	1	884	0.01	2.8	5.19	1.03
6/23/2026 13:20	0	888	0.29	2.81	0.02	1.45
6/23/2026 13:20	1	897	0.14	2.81	0.02	1.45
6/23/2026 13:20	0	897	0	2.8	0.4	1.44
6/23/2026 13:20	3	880	0.39	2.81	0.19	1.04
6/23/2026 13:20	1	878	0.39	2.81	0.83	1.04
6/23/2026 13:21	1	878	0	2.81	0.02	1.03
6/23/2026 13:21	6	878	0	2.8	14.99	1.45
6/23/2026 13:21	1	889	0.01	2.8	0.19	1.44
6/23/2026 13:21	1	893	0.01	2.81	0.18	1.45
6/23/2026 13:21	2	893	0.33	2.81	4.86	1.45
6/23/2026 13:21	0	888	0	2.8	0	1.04
6/23/2026 13:21	3	883	3.9	2.81	0.03	1.04
6/23/2026 13:22	1	884	0	2.8	0.15	1.04
6/23/2026 13:22	2	889	0.28	2.8	0.03	1.45
6/23/2026 13:22	1	895	0.4	2.8	0.02	1.45
6/23/2026 13:22	1	896	0.16	2.81	0.77	1.45
6/23/2026 13:22	5	893	0	2.8	0.04	1.04

6/23/2026 13:22	2	876	0.01	2.81	0.15	1.03
6/23/2026 13:23	9	882	0	2.81	0.03	1.04
6/23/2026 13:23	1	886	0	2.81	4.76	1.04
6/23/2026 13:23	2	891	0	2.8	0.2	1.45
6/23/2026 13:23	1	898	3.88	2.82	0.19	1.45
6/23/2026 13:23	1	898	0	2.81	0.16	1.45
6/23/2026 13:23	1	884	0.33	2.81	0.03	1.03
6/23/2026 13:23	1	884	0	2.81	0	1.03
6/23/2026 13:24	1	882	0.02	2.8	0.21	1.04
6/23/2026 13:24	5	883	7.09	2.81	0.95	1.45
6/23/2026 13:24	1	893	0.01	2.8	0.16	1.45
6/23/2026 13:24	1	896	0.29	2.81	0.18	1.45
6/23/2026 13:24	1	893	0	2.8	4.79	1.45
6/23/2026 13:24	1	878	0	2.8	0.17	1.04
6/23/2026 13:24	5	878	0.39	2.8	0	1.03
6/23/2026 13:25	0	876	0	2.81	0.02	1.04
6/23/2026 13:25	1	890	0	2.8	0	1.45
6/23/2026 13:25	1	892	0	2.8	0.02	1.45
6/23/2026 13:25	1	893	0.01	2.81	0.17	1.45
6/23/2026 13:25	6	894	0.48	2.81	0.03	1.03
6/23/2026 13:25	1	878	0.32	2.8	0.74	1.03

Lampiran 6. *Source Code* Berkas docker-compose.yml

```yaml
services:

  # ── PostgreSQL Database ──────────────────────────
  postgres:
    image: postgres:16-alpine
    container_name: lokatani-db
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${DB_NAME:-lokatani}
      POSTGRES_USER: ${DB_USER:-postgres}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-No77203239}
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
      - ./backend-app/src/db/schema.sql:/docker-entrypoint-initdb.d/01-schema.sql:ro
      - ./backend-app/src/db/seed.sql:/docker-entrypoint-initdb.d/02-seed.sql:ro
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER:-postgres} -d ${DB_NAME:-lokatani}"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    logging:
      driver: "json-file"
```

```
    options:
      max-size: "10m"
      max-file: "3"
  networks:
    - lokatani-net

# ── Eclipse Mosquitto MQTT Broker ──────────────────
mosquitto:
  image: eclipse-mosquitto:2
  container_name: lokatani-mqtt
  restart: unless-stopped
  ports:
    - "1883:1883"
    - "9001:9001"
  volumes:
    - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
    - mosquitto-data:/mosquitto/data
    - mosquitto-log:/mosquitto/log
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"
  networks:
    - lokatani-net

# ── Node.js Backend (HTTP + WS + MQTT) ─────────────
backend:
  build:
    context: ./backend-app
    dockerfile: Dockerfile
  container_name: lokatani-backend
  restart: unless-stopped
  depends_on:
    postgres:
      condition: service_healthy
    mosquitto:
      condition: service_started
  ports:
    - "3000:3000"
    - "8080:8080"
  environment:
    DB_HOST: postgres
    DB_PORT: 5432
    DB_USER: ${DB_USER:-postgres}
    DB_PASSWORD: ${DB_PASSWORD:-No77203239}
    DB_NAME: ${DB_NAME:-lokatani}
    DB_POOL_MAX: 20
    PORT_HTTP: 3000
    PORT_WS: 8080
    MQTT_BROKER_URL: mqtt://mosquitto:1883
    MQTT_CLIENT_ID: lokatani-backend
    JWT_SECRET: ${JWT_SECRET:-super-secret-jwt-key-change-in-production}
    JWT_EXPIRES_IN_SEC: ${JWT_EXPIRES_IN_SEC:-86400}
    API_KEY: ${API_KEY:-lokatani-edge-device-key}
```

```
      UPLOAD_DIR: uploads/images
      LOG_LEVEL: INFO
      MIN_CONFIDENCE: 0.60
    volumes:
      - backend-uploads:/app/uploads/images
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
    networks:
      - lokatani-net

  # ── React Frontend (Nginx SPA) ──────────────────
  frontend:
    build:
      context: ./frontend-app
      dockerfile: Dockerfile
    container_name: lokatani-frontend
    restart: unless-stopped
    depends_on:
      - backend
    ports:
      - "80:80"
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
    networks:
      - lokatani-net

volumes:
  pgdata:
    name: lokatani-pgdata
  mosquitto-data:
    name: lokatani-mqtt-data
  mosquitto-log:
    name: lokatani-mqtt-log
  backend-uploads:
    name: lokatani-uploads

networks:
  lokatani-net:
    name: lokatani-network
    driver: bridge
```

Lampiran 7. *Source Code* Skrip *Edge Device*

```
import argparse
import base64
import cv2
import csv
import logging
```

```python
import os
import threading
import time
import queue
import signal
import psutil
import socket
from datetime import datetime, timezone
from flask import Flask, Response
from picamera2 import Picamera2
from clients import create_protocol_client
from inference.processor import InferenceEngine
from hardware.serial_worker import esp8266_usb_worker

# ── LOGGING CONFIGURATION ──────────────────────────────────────
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%H:%M:%S",
)
log = logging.getLogger("LokataniEdge")

app = Flask(__name__)

MODEL_REGISTRY = {
    "v8n_640": {"path": "models/best_v8n_ncnn_model",          "imgsz": 640},
    "v8n_416": {"path": "models/yolov8n_best_416_ncnn_model",  "imgsz": 416},
    "v8n_320": {"path": "models/yolov8n_best_320_ncnn_model",  "imgsz": 320},
    "v5s_320": {"path": "models/yolov5s_best_320.pt",          "imgsz": 320},
}

# ── ARGUMENT PARSER ────────────────────────────────────────────
parser = argparse.ArgumentParser(description="Lokatani Integrated Edge Stream & Cloud
Gateway.")
parser.add_argument("--protocol", type=str, choices=["HTTP", "WS", "MQTT"],
default="HTTP", help="Protokol komunikasi cloud")
parser.add_argument("--host", type=str, default="localhost", help="IP/Hostname Cloud
Backend")
parser.add_argument("--http-port", type=int, default=3000, help="Port HTTP API")
parser.add_argument("--ws-port", type=int, default=8080, help="Port WebSocket")
parser.add_argument("--broker", type=str, default=None, help="MQTT Broker URL")
parser.add_argument("--api-key", type=str, default="lokatani-edge-device-key", help="API
Key autentikasi device")
parser.add_argument("--serial-port", type=str, default="/dev/ttyUSB0", help="Port USB
Serial ESP8266")
parser.add_argument("--min-confidence", type=float, default=0.25, help="Threshold minimum
confidence YOLO")
parser.add_argument("--run-duration", type=int, default=0, help="Durasi jalan dalam detik
(0 = selamanya)")

parser.add_argument(
    "--model",
    choices=list(MODEL_REGISTRY.keys()),
    default="v8n_320",
    help="Pilih model yang dipakai untuk inferensi (default: v8n_320)"
```

```
)

parser.add_argument(
    "--imgsz",
    type=int,
    default=None,
    help="Override ukuran input manual (opsional, default ikut konfigurasi model)"
)

args, unknown = parser.parse_known_args()

model_cfg = MODEL_REGISTRY[args.model]
model_path = model_cfg["path"]
final_imgsz = args.imgsz if args.imgsz is not None else model_cfg["imgsz"]

# ── CONFIGURATION & STORAGE ──────────────────────────────────────
RUN_TAG = args.model
BASE_OUTPUT_DIR = os.path.join("hasil_pengujian", RUN_TAG)
LOG_FILE = os.path.join(BASE_OUTPUT_DIR, "hasil_pengujian_hama.csv")
IMAGE_SAVE_DIR = os.path.join(BASE_OUTPUT_DIR, "bukti_hama")
RAW_DATASET_DIR = os.path.join(BASE_OUTPUT_DIR, "raw_dataset")
PERF_LOG_FILE = os.path.join(BASE_OUTPUT_DIR, "performa_sistem.csv")
ESP_CONNECT_TIMEOUT = 5

os.makedirs(IMAGE_SAVE_DIR, exist_ok=True)
os.makedirs(RAW_DATASET_DIR, exist_ok=True)

if not os.path.exists(LOG_FILE):
    with open(LOG_FILE, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([
            "Timestamp",
            "Session_ID",
            "File_Foto",
            "AI_Winged_Aphid",
            "AI_Whitefly",
            "Real_Winged_Aphid",
            "Real_Whitefly",
            "Waktu_Inferensi(s)",
            "Status_TP_FP_FN"
        ])

# ── DATA SHARE REPOSITORY (THREAD-SAFE CHANNELS) ─────────────────
global_frame_raw = None
global_frame_annotated = None

# Lock infrastructure
state_lock = threading.Lock()
frame_lock = threading.Lock()
shutdown_event = threading.Event()
_shutdown_in_progress = threading.Event()

io_queue = queue.Queue(maxsize=50)

shared_state = {
```

```
    "esp_hardware_ready": False,
    "esp_handshake_done": False,
    "is_scanning": False,
    "session_id": f"LOCAL_BOOT_{datetime.now().strftime('%Y%m%d_%H%M%S')}",
    "internet_connected": False,
    "is_cloud_active": False,
    "last_inf_time": 0.0,
    "current_cpu_util": 0.0,
    "current_ram_util": 0.0,
    "current_cpu_temp": 25.0,
}

# ── HARDWARE & MODEL INITIALIZATION ──────────────────────────────
picam2 = Picamera2()
config = picam2.create_preview_configuration(main={"format": "BGR888", "size": (1280,
960)})
picam2.configure(config)
picam2.set_controls({
    # Biarkan AE jalan otomatis — hapus ExposureTime & AnalogueGain
    "AeConstraintMode": 1,  # Highlight constraint: cegah area putih (whitefly) jadi blob
    "AeExposureMode": 0,    # Normal AE mode
    "AwbMode": 0,           # Auto white balance

    # Visual enhancement untuk deteksi hama
    "Saturation": 1.3,      # Boost warna hijau daun vs putih whitefly
    "Contrast": 1.2,        # Turunin dari 1.4 — jangan clip highlights
    "Sharpness": 2.0,       # Tetap tinggi buat deteksi insect kecil
})
picam2.start()

engine = InferenceEngine(model_path, args.min_confidence, imgsz=final_imgsz)
log.info(f"🧠 Model [{args.model}] dimuat | path={model_path} | imgsz={final_imgsz}")

# ── CORE CAMERA BACKGROUND WORKER ────────────────────────────────
def camera_stream_worker():
    """Worker Thread: Mengambil data matriks sensor gambar secara kontinu (~30 FPS)"""
    global global_frame_raw
    log.info("🚀 Background Camera Worker started.")
    while not shutdown_event.is_set():
        try:
            frame_rgb = picam2.capture_array()
            with frame_lock:
                global_frame_raw = frame_rgb
            time.sleep(0.03)
        except Exception as e:
            log.error(f"Error pada thread kamera: {e}")
            time.sleep(1)

def check_internet_connection(host="1.1.1.1", port=53, timeout=3.0) -> bool:
    """
    Melakukan TCP handshake kilat ke DNS publik untuk memeriksa akses internet global.
    Menggunakan port 53 (DNS) karena hampir pasti dibuka oleh ISP/Hotspot.
    """
    try:
        # Mencoba membuat koneksi socket ke DNS publik
```

```
        socket.setdefaulttimeout(timeout)
        with socket.create_connection((host, port), timeout=timeout):
            return True
    except OSError:
        return False

def network_watchdog_worker():
    """
    Worker Thread: Memantau konektivitas internet global secara real-time.
    Mencetak log spesifik saat koneksi internet terhubung kembali (back-online).
    """
    log.info("🌐 Network Access Watchdog Worker started.")
    last_state = False  # False = Offline, True = Online

    # Inisialisasi pengecekan awal saat boot
    initial_check = check_internet_connection()
    with state_lock:
        shared_state["internet_connected"] = initial_check
    last_state = initial_check

    if initial_check:
        log.info("🌐 [Network Status]: Terkoneksi ke Internet Global.")
    else:
        log.warning("🌐 [Network Status]: Offline / Tidak ada akses internet.")

    while not shutdown_event.is_set():
        try:
            # Periksa koneksi internet ke Cloudflare/Google DNS tiap 3 detik
            current_state = check_internet_connection(host="1.1.1.1", port=53,
timeout=2.0)

            if current_state != last_state:
                with state_lock:
                    shared_state["internet_connected"] = current_state

                if current_state:
                    log.info("🌐 [Network Event]: RPi BERHASIL TERKONEKSI KE INTERNET
(Akses WAN Aktif).")
                else:
                    log.warning("🌐 [Network Event]: RPi KEHILANGAN AKSES INTERNET
(Koneksi Terputus).")

                last_state = current_state

            # Jeda deteksi agar tidak membebani lalu lintas jaringan
            time.sleep(3.0)
        except Exception as e:
            log.error(f"Error pada Network Watchdog Worker: {e}")
            time.sleep(5)

# ── STATS OVERLAY HELPERS ──────────────────────────────────────
_fps_counter = {"count": 0, "last_time": time.time(), "fps": 0.0}
_fps_lock = threading.Lock()

def update_fps():
```

```
    with _fps_lock:
        _fps_counter["count"] += 1
        now = time.time()
        elapsed = now - _fps_counter["last_time"]
        if elapsed >= 1.0:
            _fps_counter["fps"] = round(_fps_counter["count"] / elapsed, 2)
            _fps_counter["count"] = 0
            _fps_counter["last_time"] = now

def get_cpu_temp():
    try:
        with open("/sys/class/thermal/thermal_zone0/temp", "r") as f:
            return round(int(f.read().strip()) / 1000, 1)
    except Exception:
        return None

# ── PERFORMANCE LOGGER ──────────────────────────────────────────
def perf_logger_worker():
    """Worker Thread: Catat FPS, CPU, RAM, Suhu ke CSV tiap 1 detik."""
    log.info("📊 Performance Logger Worker started.")
    with open(PERF_LOG_FILE, mode='a', newline='') as f:
        writer = csv.writer(f)
        while not shutdown_event.is_set():
            try:
                with _fps_lock:
                    fps = _fps_counter["fps"]

                timestamp = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%S.%f')[:-
3] + 'Z'
                cpu = psutil.cpu_percent(interval=None)
                ram = psutil.virtual_memory().percent
                suhu = get_cpu_temp()

                with state_lock:
                    shared_state["current_cpu_util"] = cpu
                    shared_state["current_ram_util"] = ram
                    if suhu is not None:
                        shared_state["current_cpu_temp"] = suhu
                    inf_time = shared_state.get("last_inf_time", 0.0)

                writer.writerow([timestamp, fps, cpu, ram, suhu if suhu is not None else
"", f"{inf_time:.4f}"])
                f.flush()
                time.sleep(1.0)
            except Exception as e:
                log.error(f"Error pada Performance Logger: {e}")
                time.sleep(1)

# ── I/O & CLOUD WORKER (CONSUMER THREAD) ────────────────────────
def io_worker(client):
    """
    Worker Thread: Mengambil data dari antrean (queue) untuk disimpan ke
    SD Card dan dikirim ke Cloud. Memisahkan I/O dari komputasi AI.
    """
    log.info("📦 Background I/O Worker started.")
```

```
    with open(LOG_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)

        while True:
            try:
                data_paket = io_queue.get()

                if data_paket is None:
                    io_queue.task_done()
                    break

                writer.writerow(data_paket['csv_row'])
                file.flush()

                if data_paket['image_to_save'] is not None:
                    cv2.imwrite(data_paket['filepath'], data_paket['image_to_save'],
[cv2.IMWRITE_JPEG_QUALITY, 85])

                if data_paket.get('raw_image_to_save') is not None:
                    cv2.imwrite(data_paket['raw_filepath'],
data_paket['raw_image_to_save'], [cv2.IMWRITE_JPEG_QUALITY, 85])

                with state_lock:
                    is_network_up = shared_state["internet_connected"]

                if data_paket['send_to_cloud'] and client and is_network_up:
                    try:
                        img_to_encode = data_paket['image_to_save'] if
data_paket['image_to_save'] is not None else data_paket['fallback_raw_image']
                        ret, jpeg_buf = cv2.imencode('.jpg', img_to_encode,
[cv2.IMWRITE_JPEG_QUALITY, 85])

                        if ret:
                            b64_str = base64.b64encode(jpeg_buf.tobytes()).decode("utf-8")
                            payload_data = {
                                "scan_session_id": data_paket['session_id'],
                                "timestamp": data_paket['timestamp'],
                                "image_base64": f"{b64_str}",
                                "detections": data_paket['detections'],
                                "min_confidence": args.min_confidence
                            }
                            client.send_detection(data_paket['session_id'], payload_data)
                            log.info(f"📤 Cloud: Data dikirim
[{data_paket['filename']}]")
                    except Exception as e:
                        log.error(f"⚠️ Gagal mengirim data ke cloud: {e}")

                io_queue.task_done()

            except Exception as e:
                log.error(f"Error pada I/O Worker: {e}")

def esp_watchdog(client):
```

```
    log.info(f"⏳ ESP Watchdog aktif — menunggu handshake maks {ESP_CONNECT_TIMEOUT}
detik...")
    deadline = time.time() + ESP_CONNECT_TIMEOUT

    while time.time() < deadline:
        with state_lock:
            if shared_state["esp_handshake_done"]:
                log.info("☑ ESP terdeteksi — mode: ESP-Controlled.")
                return
        time.sleep(0.5)

    new_session = None
    if client:
        try:
            # Proteksi jika cloud unreached saat booting agar thread gak mati
            new_session = client.start_session()
        except Exception as e:
            log.error(f"⚠ Gagal inisialisasi sesi cloud saat boot (Server offline/RTO):
{e}")

    with state_lock:
        if not shared_state["esp_handshake_done"]:
            shared_state["is_scanning"] = True
            if new_session:
                shared_state["session_id"] = new_session
                shared_state["is_cloud_active"] = True
                log.info(f"☁  [STANDALONE CLOUD] Sesi server aktif: #{new_session}")
            else:
                shared_state["session_id"] =
f"STANDALONE_{datetime.now().strftime('%Y%m%d_%H%M%S')}"
                shared_state["is_cloud_active"] = False
                log.warning("☁  [STANDALONE LOCAL] Cloud tidak tersedia, simpan lokal.")
    log.warning("⚠  ESP tidak terdeteksi → STANDALONE MODE aktif, scanning berjalan
otomatis.")

# ── PIPELINE ENGINE: INFERENCE & DATA PACKAGING WORKER ──────────────
def ai_inference_worker():
    """Worker Thread: Memproses Digital Zoom, Inferensi YOLOv8, Log CSV, & Pipeline
Cloud."""
    global global_frame_annotated
    log.info("🚀 Background AI Inference Worker started.")
    last_save_time = 0
    last_sent_time = 0
    SAVE_INTERVAL = 5.0
    SEND_INTERVAL = 3.0

    while not shutdown_event.is_set():
        try:
            with state_lock:
                current_scanning = shared_state["is_scanning"]
                current_session_id = shared_state["session_id"]
                current_cloud_active = shared_state["is_cloud_active"]

            if not current_scanning:
                time.sleep(0.1)
```

```
                continue

            with frame_lock:
                if global_frame_raw is None:
                    time.sleep(0.01)
                    continue
                local_raw = global_frame_raw.copy()

            annotated_img, detections, counts, inf_time, cropped_raw =
engine.detect_and_process(local_raw)
            update_fps()

            with frame_lock:
                global_frame_annotated = annotated_img.copy()

            count_winged_aphids = counts.get("winged_aphid", 0)
            count_whitefly = counts.get("whitefly", 0)

            with state_lock:
                shared_state["last_inf_time"] = inf_time

            timestamp_sekarang = datetime.now(timezone.utc).strftime('%Y-%m-
%dT%H:%M:%S.%f')[:-3] + 'Z'
            current_time = time.time()
            hama_detected = (count_winged_aphids > 0 or count_whitefly > 0)

            should_send = (current_time - last_sent_time >= SEND_INTERVAL)

            filename = "-"
            if hama_detected or (current_time - last_save_time >= SAVE_INTERVAL):
                waktu_detail = datetime.now().strftime("%Y%m%d-%H%M%S_%f")[:-3]
                if hama_detected:
                    filename =
f"hama_{waktu_detail}_A{count_winged_aphids}_K{count_whitefly}.jpg"
                else:
                    filename = f"negatif_{waktu_detail}.jpg"

            if current_scanning:
                img_save_payload = annotated_img.copy() if filename != "-" else None
                raw_save_payload = cropped_raw.copy() if filename != "-" else None
                fallback_payload = annotated_img.copy() if (should_send and
current_cloud_active and filename == "-") else None

                paket = {
                    'csv_row': [
                        timestamp_sekarang,
                        current_session_id,
                        filename,
                        count_winged_aphids,
                        count_whitefly,
                        "",
                        "",
                        f"{inf_time:.4f}",
                        ""
                    ],
```

```
                    'image_to_save': img_save_payload,
                    'raw_image_to_save': raw_save_payload,
                    'fallback_raw_image': fallback_payload,
                    'filepath': os.path.join(IMAGE_SAVE_DIR, filename) if filename != "-"
else "",
                    'raw_filepath': os.path.join(RAW_DATASET_DIR, filename.replace('.jpg',
'_raw.jpg')) if filename != "-" else "",
                    'send_to_cloud': should_send and current_cloud_active,
                    'session_id': current_session_id,
                    'timestamp': timestamp_sekarang,
                    'detections': detections,
                    'filename': filename
                }

                try:
                    # Berikan toleransi waktu tunggu 0.1 detik sebelum benar-benar
melempar/drop frame
                    io_queue.put(paket, timeout=0.1)
                except queue.Full:
                    log.warning("⚠️ Antrean I/O Penuh! Melewati frame ini biar AI gak
ngelag.")

                if filename != "-":
                    last_save_time = current_time
                if should_send:
                    last_sent_time = current_time

            time.sleep(0.01)

        except Exception as e:
            log.error(f"Error pada thread AI Inference Worker: {e}")
            time.sleep(1)

# ── FLASK STREAMING GENERATOR ──────────────────────────────────────
def gen_frames():
    """Generator streaming mjpeg yang mengambil data ter-update dari AI Engine"""
    log.info("📺 Client baru terhubung ke Flask Live Stream.")
    while True:
        with frame_lock:
            if global_frame_annotated is None:
                time.sleep(0.04)
                continue
            frame_bgr = global_frame_annotated.copy()

        with state_lock:
            scanning = shared_state.get("is_scanning", False)
            cloud = shared_state.get("is_cloud_active", False)
            cpu = shared_state.get("current_cpu_util", 0.0)
            ram = shared_state.get("current_ram_util", 0.0)
            suhu = shared_state.get("current_cpu_temp", 25.0)

        with _fps_lock:
            fps = _fps_counter["fps"]

        overlay_lines = [
```

```
            f"FPS: {fps:.1f}",
            f"CPU: {cpu:.1f}%  RAM: {ram:.1f}%",
            f"Suhu: {suhu:.1f}C",
            f"Scan: {'ON' if scanning else 'OFF'}  Cloud: {'ON' if cloud else 'OFF'}",
        ]

        y = 20
        for line in overlay_lines:
            cv2.putText(frame_bgr, line, (10, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 0), 3)
            cv2.putText(frame_bgr, line, (10, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 0), 1)
            y += 22

        ret, buffer = cv2.imencode('.jpg', frame_bgr, [cv2.IMWRITE_JPEG_QUALITY, 70])
        if not ret:
            time.sleep(0.04)
            continue

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + buffer.tobytes() + b'\r\n')

        time.sleep(0.04)

shutdown_lock = threading.Lock()

def graceful_shutdown(signum=None, frame=None):
    # Gunakan kuncian atomik, gagalkan thread kedua yang mencoba masuk secara bersamaan
    if not shutdown_lock.acquire(blocking=False):
        return

    if shutdown_event.is_set():
        return

    log.warning(f"🔴 Sinyal shutdown ({signum}) diterima — memicu event stop ke semua
thread...")

    # 1. Hentikan loop utama produsen skrip (AI & Camera Workers)
    shutdown_event.set()

    # 2. Kirim poison pill dan kunci aliran data hingga buffer kosong
    try:
        io_queue.put(None, timeout=1.0)
    except queue.Full:
        log.warning("⚠️ Antrean penuh, poison pill terpaksa dilewati.")

    log.info("⏳ Menunggu io_worker mengosongkan sisa buffer ke SD Card...")

    timeout_deadline = time.time() + 5.0
    while not io_queue.empty() and time.time() < timeout_deadline:
        time.sleep(0.1)

    if not io_queue.empty():
        log.warning("⚠️ Batas waktu habis! Sisa antrean terpaksa di-drop demi stabilitas
sistem.")
```

```
    # 3. Lepaskan driver tingkat hardware eksternal secara defensif
    try:
        picam2.stop()
        log.info("📷 Perangkat keras kamera berhasil dilepaskan.")
    except Exception as e:
        log.error(f"Gagal menghentikan kamera: {e}")

    # 4. Lepaskan soket komunikasi jaringan cloud
    if cloud_client and hasattr(cloud_client, "close"):
        try:
            cloud_client.close()
            log.info("🔌 Soket komunikasi cloud berhasil ditutup.")
        except Exception as e:
            log.error(f"Gagal menutup client cloud secara bersih: {e}")

    log.info("☑️ Semua resource selesai dibersihkan. Keluar.")
    os._exit(0)

@app.route('/')
def index():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

# ── MAIN ENTRY POINT ──────────────────────────────────────────
if __name__ == '__main__':
    log.info("============================================================")
    log.info("  Lokatani Edge Runtime - Raspberry Pi Initialization  ")
    log.info("============================================================")

    cloud_client = create_protocol_client(
        protocol_type=args.protocol,
        host=args.host,
        http_port=args.http_port,
        ws_port=args.ws_port,
        broker=args.broker,
        api_key=args.api_key
    )

    signal.signal(signal.SIGTERM, graceful_shutdown)
    signal.signal(signal.SIGINT, graceful_shutdown)

    threads = [
        threading.Thread(target=camera_stream_worker, daemon=True),
        threading.Thread(target=ai_inference_worker, daemon=True),
        threading.Thread(target=perf_logger_worker, daemon=True),
        threading.Thread(target=io_worker, args=(cloud_client,), daemon=True),
        threading.Thread(target=esp8266_usb_worker, args=(args, state_lock, shared_state,
cloud_client, log), daemon=True),
        threading.Thread(target=esp_watchdog, args=(cloud_client,), daemon=True),
        threading.Thread(target=network_watchdog_worker, daemon=True)
    ]

    for t in threads:
        t.start()
```

```
    if args.run_duration > 0:
        def auto_shutdown():
            time.sleep(args.run_duration)
            log.info(f"⏰ Durasi {args.run_duration} detik selesai — sistem mati
otomatis.")
            os.kill(os.getpid(), signal.SIGTERM)
        threading.Thread(target=auto_shutdown, daemon=True).start()
        log.info(f"⏱️  Auto-shutdown aktif dalam {args.run_duration} detik.")

    app.run(host='0.0.0.0', port=5000, threaded=True, debug=False)
```