

PROJECT MATA KULIAH

SISTEM KONTROL TERDISTRIBUSI

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



Disusun oleh:

M Shofiyur Rochman (2042231031)

Maulidan Arridlo (2042231059)

**Departemen Teknik Instrumentasi
Fakultas Vokasi
Institut Teknologi Sepuluh Nopember
2025**

DAFTAR ISI

I	Pendahuluan	1
1.1	Rumusan Masalah	1
1.2	Batasan Masalah	2
1.3	Tujuan	2
1.4	Manfaat	2
II	Tinjauan Pustaka	3
2.1	ESP32-S3	3
2.2	RS485 dan MAX485	3
2.3	Sensor SHT20 (Modbus RTU)	3
2.4	Relay dan Exhaust Fan	3
2.5	Servo dan PWM	3
2.6	MQTT dan Time-Series	4
2.7	Rust di ESP32	4
2.8	InfluxDB	4
2.9	ThingsBoard	5
2.10	DWSIM	6
2.11	Python	6
III	Metodologi	8
3.1	Arsitektur Sistem	8
3.2	Diagram Blok Close-Loop	8
3.3	Metode: Penjelasan Kode Python	9
3.4	Metode: Penjelasan Kode ESP	9
3.5	Wiring Diagram	10
IV	Implementasi dan Hasil	11
4.1	Hasil Wiring Alat	11
4.2	Hasil Running Kode pada Terminal	12
4.3	Hasil Integrasi ke Influx	13

4.4 Hasil Integrasi ke ThingsBoard	15
V Pembahasan	18
VI Kesimpulan dan Saran	19
6.1 Kesimpulan	19
6.2 Saran	19
A Kode ESP (Rust/ESP-IDF)	20
B Kode Python (Gateway/Validasi)	38

DAFTAR GAMBAR

II.1	Perakitan ESP32–MAX485–relay–servo, contoh platform target Rust/ESP-IDF.	4
II.2	Dashboard Influx: nilai suhu dan kelembapan terbaca real-time. . . .	5
II.3	Dashboard ThingsBoard: suhu, kelembapan, indikator kipas dan ga- uge bukaan servo.	5
II.4	Aplikasi jembatan DWSIM–Influx: <i>solve</i> lalu <i>push</i> 4 parameter ke bucket.	6
II.5	Contoh kurva variabel dari Python/Influx yang menunjukkan konsis- tensi sinyal.	7
III.1	Loop kontrol damper (servo, PWM) berbasis suhu.	8
III.2	Loop kontrol exhaust fan (relay, digital GPIO).	8
III.3	Tangkapan layar validasi variabel setelah kode ESP berjalan dan me- nulis ke Influx.	9
III.4	Diagram wiring (mengacu pada foto perakitan di Bab IV).	10
IV.1	Foto hasil wiring sistem lengkap.	12
IV.2	Panel suhu/kelembapan pada Influx.	15
IV.3	Perbandingan variabel DWSIM dan telemetry lapangan dalam satu dashboard.	15
IV.4	Dashboard ThingsBoard yang digunakan dalam pengujian.	17

BAB I PENDAHULUAN

Latar Belakang

Ventilasi yang baik merupakan kunci untuk menjaga kualitas udara, kenyamanan termal, dan keselamatan ruang tertutup. Pada banyak aplikasi—laboratorium, rumah kaca, ruang proses, hingga rak server—kenaikan suhu dan kelembapan dapat menurunkan kinerja peralatan, mempercepat korosi, dan memicu pertumbuhan jamur. Sistem ventilasi konvensional umumnya dikendalikan manual sehingga responsnya lambat dan tidak konsisten. Teknologi *embedded* dan Internet of Things (IoT) memungkinkan ventilasi dikendalikan secara otomatis berdasarkan data sensor secara waktu nyata.

Proyek ini merancang **sistem pembukaan ventilasi otomatis** yang memanfaatkan **servo** untuk membuka–menutup *damper* dan **exhaust fan** sebagai pembuang udara panas. Parameter acuan diambil dari **sensor suhu–kelembapan SHT20** yang pada modul industri berkomunikasi melalui **RS485 (Modbus RTU)**. Komunikasi diferensial tersebut dihubungkan ke **ESP32-S3** melalui konverter **MAX485 (RS485 ↔ TTL)**. Keputusan kontrol dilakukan di ESP32 dan data dikirim ke **ThingsBoard (MQTT)** serta disimpan ke **InfluxDB** untuk analitik. **Relay** mengendalikan daya *exhaust fan*, sedangkan posisi **servo** diatur memakai sinyal **PWM** dari periferi LEDC ESP32.

1.1 Rumusan Masalah

1. Bagaimana merancang sistem ventilasi otomatis berbasis ESP32-S3 yang membuka damper (servo) dan menyalakan *exhaust fan* (relay) berdasarkan suhu/kelembapan?
2. Bagaimana mengintegrasikan sensor SHT20 Modbus RTU (RS485) dengan ESP32-S3 menggunakan konverter MAX485?
3. Bagaimana mempublikasikan data dan status aktuator ke platform IoT (MQTT) serta menyimpannya ke *time-series database* (InfluxDB)?

1.2 Batasan Masalah

1. Sensor yang digunakan adalah SHT20 versi RS485 (Modbus RTU).
2. Aktuator terdiri dari satu servo (damper) dan satu *exhaust fan* via relay.
3. Catu daya sistem 5 V; kendali servo via PWM dan kipas via *digital GPIO* (relay).

1.3 Tujuan

1. Mewujudkan sistem ventilasi otomatis yang andal dan mudah direplikasi.
2. Menyediakan pemantauan data lingkungan dan status aktuator melalui MQTT dan penyimpanan InfluxDB.
3. Menyusun dokumentasi wiring dan klasifikasi kabel menurut protokol komunikasi.

1.4 Manfaat

1. Menurunkan suhu/kelembapan ruangan secara otomatis dan konsisten.
2. Memberi rekam data historis untuk evaluasi performa ventilasi.
3. Menjadi referensi praktis integrasi Modbus RTU—ESP32—MQTT—InfluxDB.

BAB II TINJAUAN PUSTAKA

2.1 ESP32-S3

ESP32-S3 adalah mikrokontroler SoC dari Espressif dengan CPU Xtensa dual-core, RAM internal memadai, serta konektivitas Wi-Fi 2.4 GHz dan BLE. Periferal yang kaya—UART, I²C, SPI, PWM (LEDC), ADC, USB OTG—menjadikannya cocok sebagai *edge controller* pada proyek ini.

2.2 RS485 dan MAX485

RS485 adalah standar fisik diferensial yang tangguh terhadap *noise*. Konversi level TTL \leftrightarrow RS485 dikerjakan oleh IC MAX485. Pin DE/RE dikendalikan GPIO untuk *transmit* dan *receive*. Protokol data yang digunakan adalah Modbus RTU.

2.3 Sensor SHT20 (Modbus RTU)

SHT20 pada modul industri menyediakan register suhu dan kelembapan yang bisa dibaca memakai fungsi Modbus 0x03/0x04. Skala umum adalah 0.1 °C dan 0.1 %.

2.4 Relay dan Exhaust Fan

Relay 5 V dengan *optocoupler* digunakan untuk mengalirkan daya ke *exhaust fan*. Pin kendali relay adalah **digital GPIO**.

2.5 Servo dan PWM

Servo dikendalikan sinyal PWM periode 20 ms. Lebar pulsa 1–2 ms mewakili 0–180°. ESP32-S3 menghasilkan PWM via LEDC.

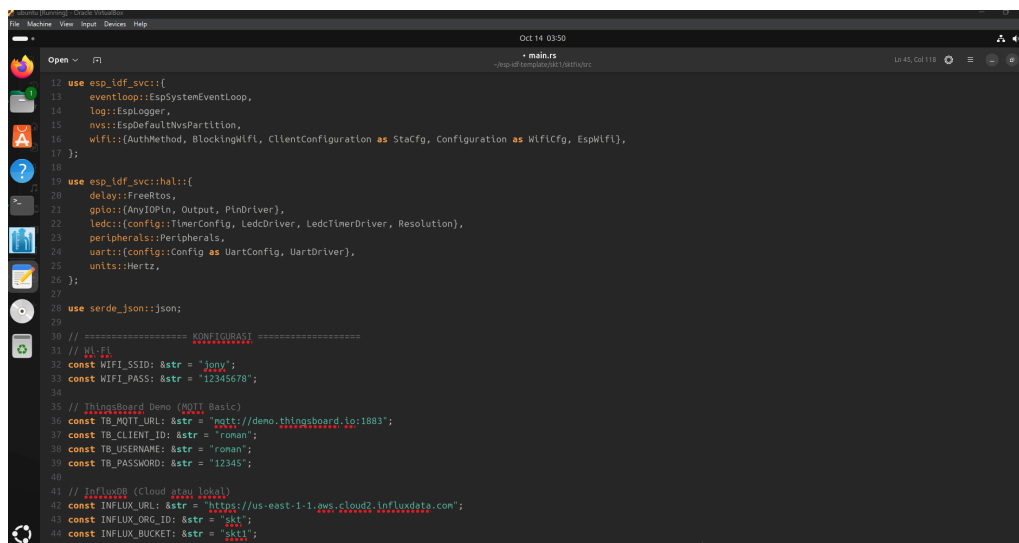
2.6 MQTT dan Time-Series

MQTT dipakai untuk telemetry ke ThingsBoard; InfluxDB menyimpan *time-series* memakai *Line Protocol*.

2.7 Rust di ESP32

Rust menawarkan *memory safety* tanpa GC dan performa setara C/C++. Port ESP32 didukung oleh `esp-idf-sys` (binding C-API), `esp-idf-svc` (abstraksi layanan Wi-Fi/MQTT), dan `esp-idf-hal`. Pada proyek ini Rust dipakai untuk:

1. Mengelola UART dan kendali DE/RE untuk RS485.
2. Menghitung CRC-16 Modbus dan memetakan register SHT20.
3. Mengirim telemetry ThingsBoard (MQTT) dan menulis InfluxDB (HTTP/HTTPS).
4. Mengendalikan servo (LEDC) serta relay (GPIO).

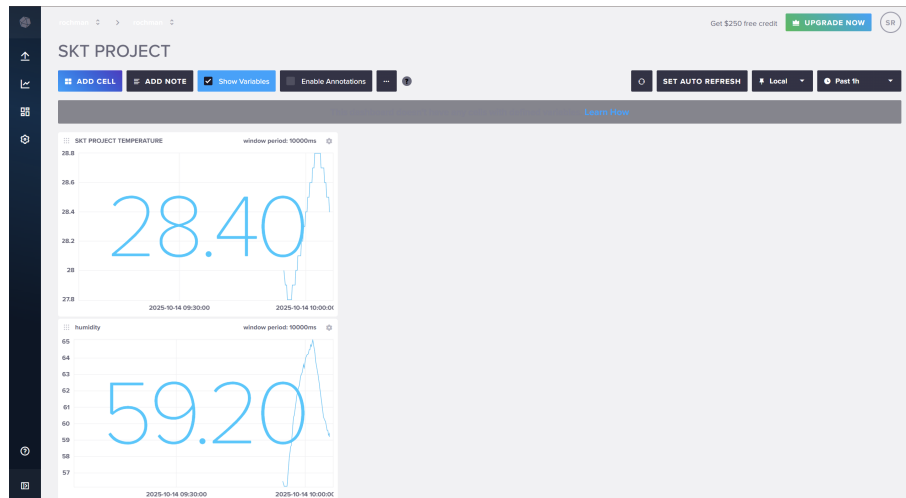


Gambar II.1: Perakitan ESP32-MAX485-relay-servo, contoh platform target Rust/ESP-IDF.

2.8 InfluxDB

InfluxDB adalah *time-series database* dengan skema *measurement*, *tags*, *fields*. Data dikirim memakai *HTTP Write API* (status 204 menandakan sukses). Dalam proyek,

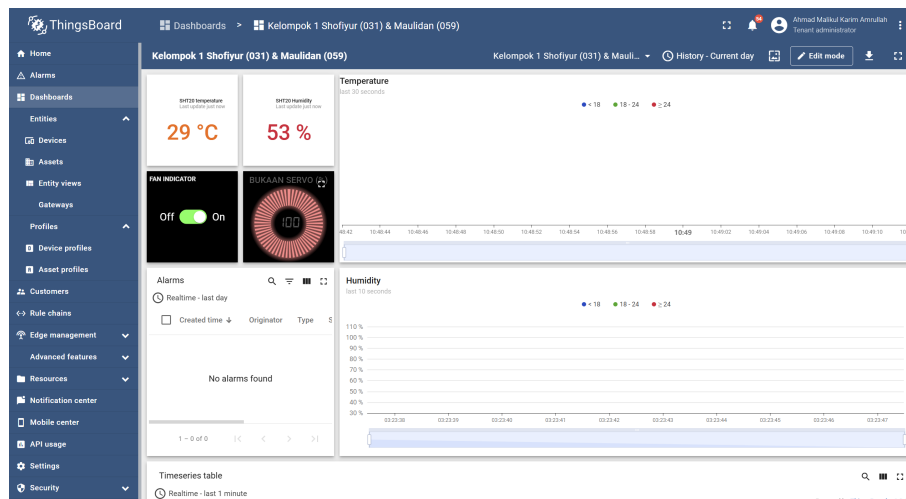
measurement sht20 memiliki fields: *temperature_c*, *humidity_pct*, *relay_state*, *servo_position_pct*, dan tag *device*.



Gambar II.2: Dashboard Influx: nilai suhu dan kelembapan terbaca real-time.

2.9 ThingsBoard

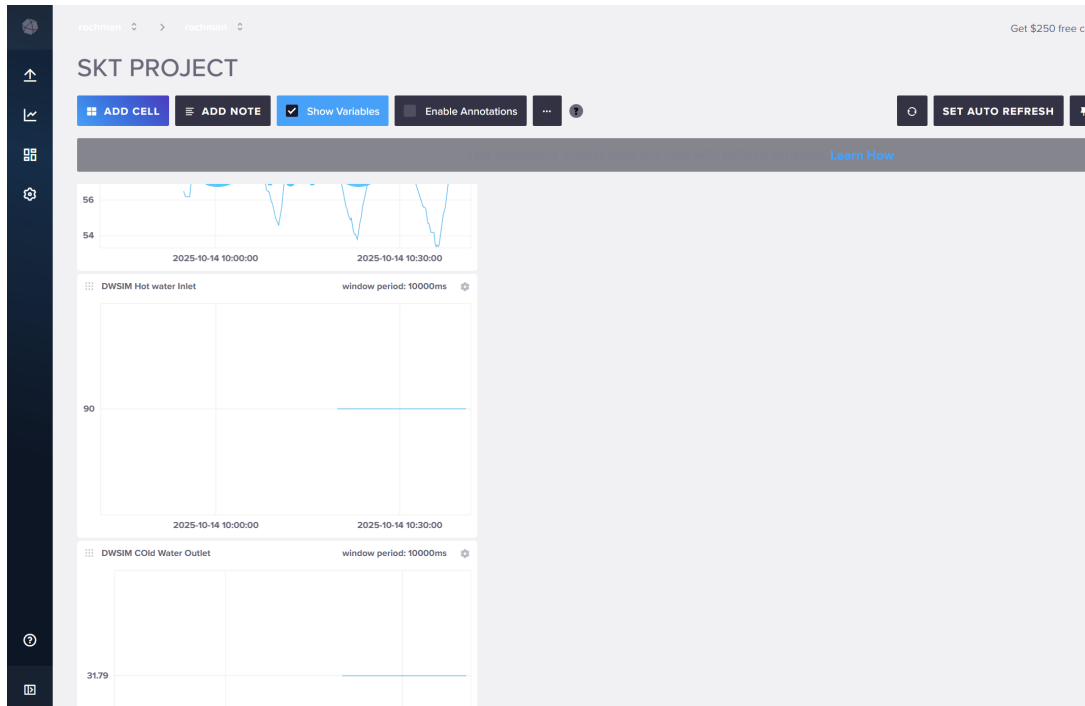
ThingsBoard adalah platform IoT berbasis MQTT/HTTP dengan fitur *rule engine* dan *dashboard*. Perangkat mengirim telemetry ke topik `v1/devices/me/telemetry`. Widget indikator kipas dan bukaan servo disusun untuk memantau status aktuator.



Gambar II.3: Dashboard ThingsBoard: suhu, kelembapan, indikator kipas dan ga-
gung bukaan servo.

2.10 DWSIM

DWSIM adalah simulator proses kimia *open-source*. Pada proyek ini DWSIM dipakai sebagai *digital twin* sederhana untuk mensintesis variabel “*hot/cold stream*” dan dibandingkan dengan pembacaan lapangan. Aplikasi jembatan menulis empat parameter hasil penyelesaian (*solve*) ke Influx sehingga dapat dipetakan berdampingan.

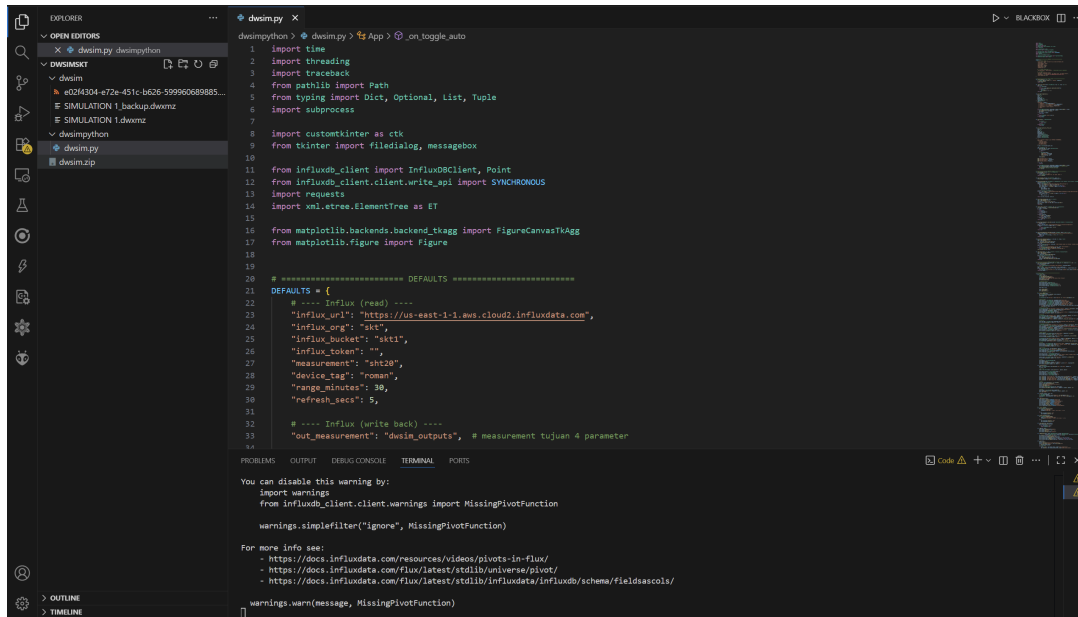


Gambar II.4: Aplikasi jembatan DWSIM–Influx: *solve* lalu *push* 4 parameter ke bucket.

2.11 Python

Python digunakan pada sisi PC/gateway untuk:

- melakukan *health check* koneksi ke Influx,
- menarik data historis untuk validasi terhadap DWSIM,
- menyiapkan *CLI utility* monitoring yang menulis log terminal.



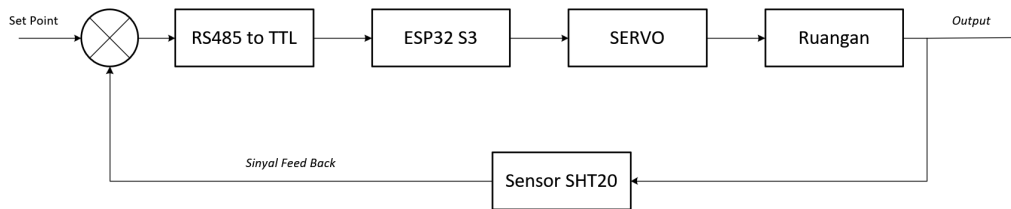
Gambar II.5: Contoh kurva variabel dari Python/Influx yang menunjukkan konsistensi sinyal.

BAB III METODOLOGI

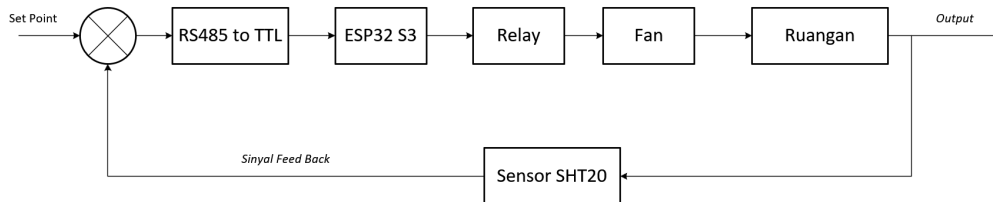
3.1 Arsitektur Sistem

ESP32-S3 membaca suhu/kelembapan dari SHT20 melalui RS485 (Modbus RTU) via MAX485. Data divalidasi (CRC) lalu dipublikasikan ke ThingsBoard (MQTT) dan ditulis ke InfluxDB (HTTP). Keputusan kontrol: *relay ON* jika $T \geq 27,2^{\circ}\text{C}$; servo ke $0^{\circ}/180^{\circ}$ pada ambang $28,5^{\circ}\text{C}$ dengan reset periodik ke 90° .

3.2 Diagram Blok Close-Loop



Gambar III.1: Loop kontrol damper (servo, PWM) berbasis suhu.



Gambar III.2: Loop kontrol exhaust fan (relay, digital GPIO).

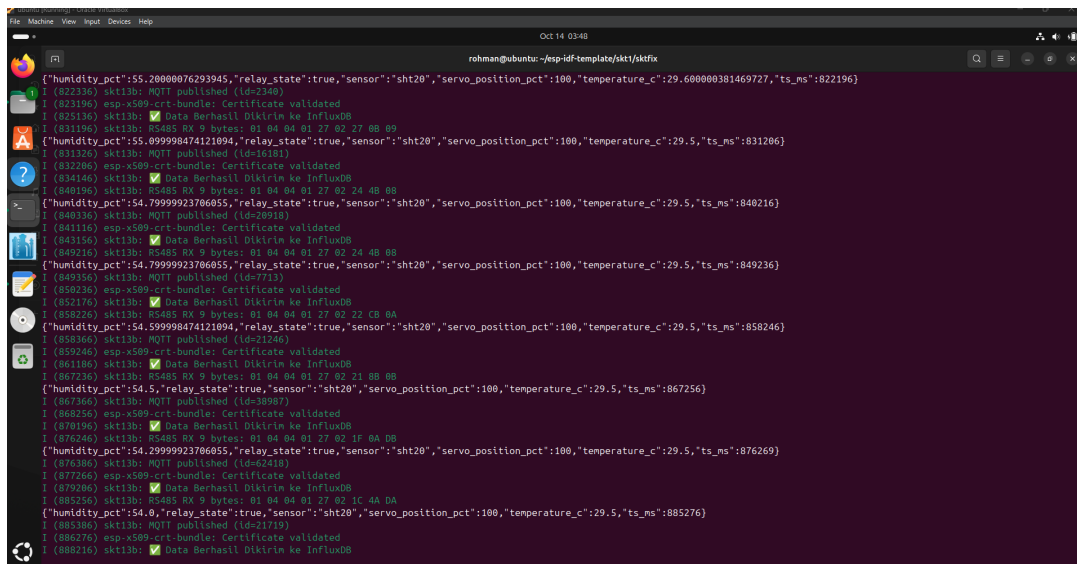
3.3 Metode: Penjelasan Kode Python

Kode Python (lihat Lampiran) bersifat *utility*: menguji kredensial Influx, menulis sampel, membaca kembali, dan memplot ringkas untuk validasi. Alur: (1) memuat URL, token, org, bucket; (2) *ping* endpoint; (3) menulis *line protocol*; (4) *query* rentang waktu; (5) menampilkan ringkasan di terminal. Output terminal dibahas pada Bab IV.

3.4 Metode: Penjelasan Kode ESP

Kode ESP (lihat Lampiran) ditulis dengan Rust/ESP-IDF. Struktur utama:

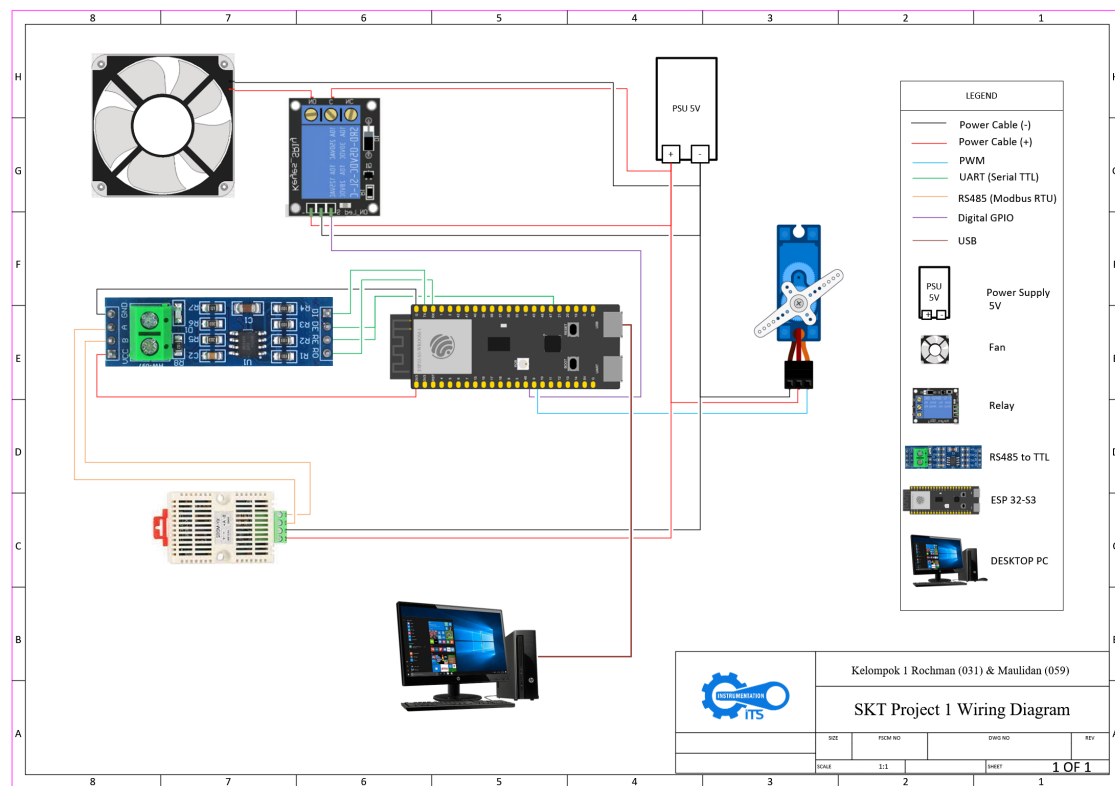
1. **Inisialisasi**: Wi-Fi, MQTT, HTTP client, UART0 + GPIO DE.
2. **RS485**: fungsi `build_read_req`, `modbus_crc`, `parse_read_resp`.
3. **Kontrol**: tipe Servo (LEDC) dan Relay (GPIO) dengan logika ambang.
4. **Publikasi**: payload JSON ke ThingsBoard dan *line protocol* ke Influx.
5. **Reliabilitas**: retry baca Modbus, delay DE, dan *cycle reset* servo 20 detik.



```
rohman@ubuntu: ~/esp-idf-template/skt/sktfix
I (822336) skt13b: MQTT published (id=2340)
I (823196) esp-x509-crt-bundle: Certificate validated
I (825180) skt13b: Data Berhasil Dikirim ke InfluxDB
I (831186) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 27 00 00
I (831326) skt13b: MQTT published (id=16181)
I (832206) esp-x509-crt-bundle: Certificate validated
I (834166) skt13b: Data Berhasil Dikirim ke InfluxDB
I (840196) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 24 4B 00
I (840336) skt13b: MQTT published (id=20918)
I (841116) esp-x509-crt-bundle: Certificate validated
I (843156) skt13b: Data Berhasil Dikirim ke InfluxDB
I (849216) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 24 4B 00
I (849356) skt13b: MQTT published (id=7713)
I (850236) esp-x509-crt-bundle: Certificate validated
I (852176) skt13b: Data Berhasil Dikirim ke InfluxDB
I (858226) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 22 CB 0A
I (858366) skt13b: MQTT published (id=22266)
I (859246) esp-x509-crt-bundle: Certificate validated
I (861186) skt13b: Data Berhasil Dikirim ke InfluxDB
I (867236) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 21 8B 00
I (867366) skt13b: MQTT published (id=20987)
I (868256) esp-x509-crt-bundle: Certificate validated
I (870196) skt13b: Data Berhasil Dikirim ke InfluxDB
I (876246) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 1F 0A 00
I (876386) skt13b: MQTT published (id=6218)
I (877266) esp-x509-crt-bundle: Certificate validated
I (879206) skt13b: Data Berhasil Dikirim ke InfluxDB
I (887246) skt13b: RS485 RX 9 bytes: 01 04 04 01 27 02 1C 4A 0A
I (887386) skt13b: MQTT published (id=21719)
I (888276) esp-x509-crt-bundle: Certificate validated
I (888316) skt13b: Data Berhasil Dikirim ke InfluxDB
```

Gambar III.3: Tangkapan layar validasi variabel setelah kode ESP berjalan dan menulis ke Influx.

3.5 Wiring Diagram



Gambar III.4: Diagram wiring (mengacu pada foto perakitan di Bab IV).

BAB IV IMPLEMENTASI DAN HASIL

4.1 Hasil Wiring Alat

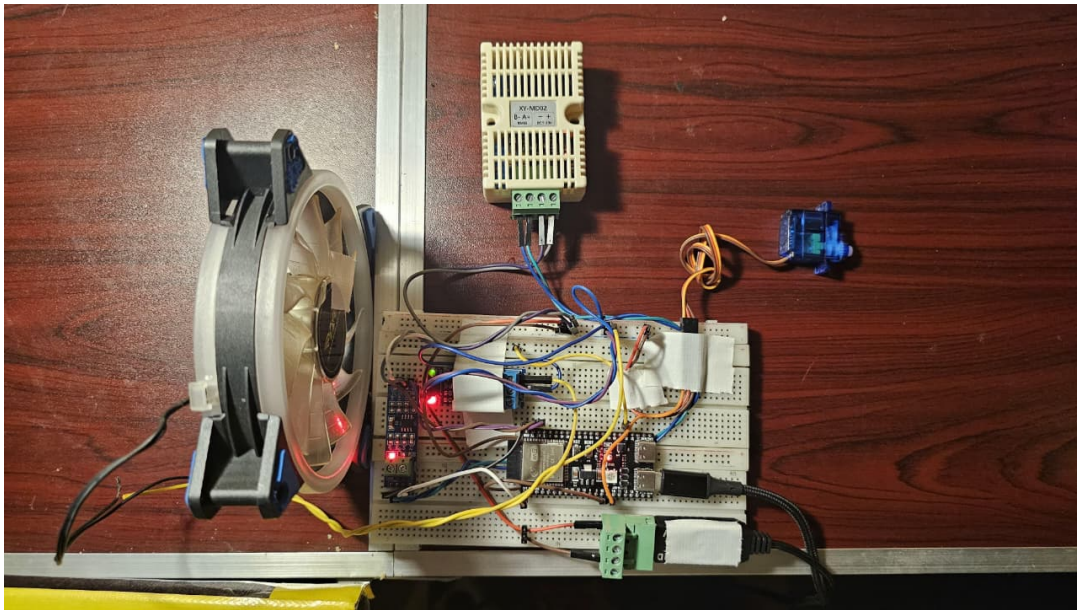
Perakitan perangkat dimulai dari subsistem catu daya 5 V yang menyuplai ESP32-S3, modul MAX485, modul relay, dan servo. Jalur **GND** disatukan pada satu *rail* untuk menghindari referensi mengambang antar modul. Antara sisi logika dan aktuator terdapat isolasi optik pada modul relay sehingga gangguan transien dari beban induktif kipas tidak menginjeksikan *spike* ke mikrokontroler. Komunikasi sensor menggunakan standar **RS485** dengan pasangan diferensial A/B yang dipe-lintir (*twisted pair*). Pada konektor sensor, A diberi kode warna hijau dan B ungu, sedangkan 5 V merah dan GND hitam. Di sisi papan percobaan, antarmuka TTL dihubungkan ke ESP32 melalui UART0: U0TXD ke pin DI MAX485 dan U0RXD ke R0. Pin DE dan RE digabung lalu disambungkan ke satu GPIO (GPIO21) sehingga perangkat lunak dapat mengaktifkan *driver* hanya ketika mengirim, dan menonaktifkannya saat menerima. Praktik ini mencegah tabrakan bus jika kelak ada beberapa node Modbus.

Sinyal **PWM** ke servo diambil dari periferan LEDC pada GPIO10. Kami menyiapkan periode 20 ms dengan resolusi 14-bit; tiga nilai tugas (*duty*) dipetakan ke 0°, 90°, dan 180° untuk kesederhanaan namun tetap presisi. Jalur servo diberi warna kuning (PWM), merah (VCC), dan coklat/hitam (GND) mengikuti konvensi umum. Modul **relay** terhubung ke GPIO9 melalui pin IN. Kontak daya COM dan NO diseri dengan suplai AC kipas exhaust. Karena beban bersifat induktif, penempatan *snubber* RC opsional direkomendasikan untuk mengurangi *arcing*, meski pada pengujian skala kecil belum terlihat gejala signifikan.

USB-C ESP32 digunakan hanya untuk pemrograman dan log serial. Saat perangkat beroperasi mandiri, suplai 5 V berasal dari adaptor *regulated*. Penataan kabel mengikuti tabel klasifikasi protokol sehingga jejak warna membantu *troubleshooting*. Misalnya, ketika terjadi kebisuan bus, pemeriksa cukup menelusuri pasangan hijau-ungu pada rute RS485 dan memastikan *termination* impedansi di sisi sensor telah benar.

Pada tahap komisioning, kami memeriksa arah kipas dan gerak servo. Servo

memiliki kecenderungan *drift* kecil setelah berjam-jam operasi; karena itu program mengembalikan posisi ke 90° tiap 20 detik sebagai kalibrasi halus. Foto hasil wiring ditunjukkan pada Gambar IV.1. Tampak kipas, papan *breadboard* berisi ESP32, MAX485, relay, serta sensor RS485 eksternal. Penataan ini memudahkan akses pengukuran osiloskop ke jalur UART maupun PWM untuk validasi timing. Secara keseluruhan, praktik pemisahan jalur daya-logika, konvensi warna kabel, dan penguncian konektor menghasilkan instalasi yang rapi, aman, dan mudah dipelihara. Pendekatan *modular* memungkinkan penggantian sensor RS485 lain tanpa mengubah arsitektur utama.



Gambar IV.1: Foto hasil wiring sistem lengkap.

4.2 Hasil Running Kode pada Terminal

Selama pengujian, terminal serial menjadi alat utama untuk mendiagnosis aliran data dan stabilitas sistem. Setelah *boot*, log pertama memperlihatkan inisialisasi ESP-IDF, kemudian modul Wi-Fi memindai SSID dan melakukan asosiasi WPA2. Ketika alamat IP diperoleh, *client* MQTT diinisialisasi lalu melakukan koneksi ke broker ThingsBoard. Log menampilkan *keep-alive* 30 detik dan identitas klien. Jika kredensial salah, modul menampilkan kode galat dan mencoba kembali sesuai *backoff*. Keberhasilan koneksi ditandai dengan pesan “MQTT connected” dan siap mempublikasikan telemetri.

Tahap berikut adalah konfigurasi UART serta *driver enable* RS485. Saat sistem akan membaca sensor, program membentuk bingkai Modbus (alamat, fungsi,

alamat register, jumlah, CRC). Log “RS485 TX” memperlihatkan deretan heksadesimal yang memudahkan verifikasi. Setelah pengiriman, DE dimatikan dan modul menunggu balasan. Pesan “RS485 RX n bytes” menunjukkan panjang dan isi bingkai respons. Jika sensor belum tersambung, log menampilkan “response too short” atau “timeout”; mekanisme *retry* membatasi tiga kali percobaan agar loop tidak terjebak.

Data yang valid kemudian dibulatkan satu desimal untuk mengurangi jitter tampilan dashboard. Telemetry JSON dipublikasikan ke topik ThingsBoard dengan QoS 1; nomor pesan (message id) dicetak untuk menilai integritas *publish*. Secara paralel, string *line protocol* dibentuk dan dikirim ke Influx melalui HTTP. Keberhasilan ditandai “HTTP 204”; bila gagal (mis. token salah atau koneksi terputus) akan muncul peringatan berisi kode status dan potongan tubuh respons. Mekanisme ini membantu menelusuri galat “connection reset by peer” yang sesekali terjadi ketika jaringan tidak stabil.

Bagian terminal juga menampilkan keputusan kontrol: ketika suhu melebihi ambang, log menulis “Servo → 180°” dan “Relay → ON” lengkap dengan nilai suhu. Saat suhu turun, perintah kebalikan dicetak. Setiap 20 detik, pesan “Reset siklus 20s: Servo → 90°” muncul sebagai indikator *watchdog* posisi. Praktik pencatatan seperti ini penting karena memudahkan korelasi antara aksi aktuator dan tren data historis di Influx. Selain itu, log *probe map* pada awal siklus mencoba beberapa kombinasi fungsi dan alamat register untuk menyesuaikan variasi pabrikan sensor RS485—fitur yang berguna ketika dokumentasi sensor minim.

Secara keseluruhan, keluaran terminal berfungsi sebagai *single source of truth* selama komisioning. Dari proses *handshake* Wi-Fi, negosiasi MQTT, transaksi Modbus, hingga *HTTP write*, semua tercermin dalam teks yang ringkas namun informatif. Dokumentasi log ini juga menjadi bahan jejak audit (*traceability*) ketika mengevaluasi ketahanan sistem saat dipasang di lingkungan nyata dengan interferensi elektromagnetik, *brown-out* suplai, atau fluktuasi jaringan.

4.3 Hasil Integrasi ke Influx

Integrasi ke InfluxDB bertujuan menyediakan repositori deret waktu yang efisien untuk analitik dan pelaporan. Pada implementasi ini, arsitektur menempatkan ESP32 sebagai *edge writer* langsung ke endpoint `/api/v2/write`. Keuntungan pendekatan langsung adalah latensi rendah dan konfigurasi sederhana tanpa *gateway* tambahan. Kami menggunakan *bucket* bernama `skt1` dalam *org skt*; autentikasi melalui skema **Token**. Payload mengikuti standar *Line Protocol* dengan pengelompokan yang jelas antara *measurement*, *tags*, dan *fields*. Contohnya:

```
sht20,device=roman
temperature_c=29.5,humidity_pct=59.2,relay_state=1,servo_position_pct=100
```

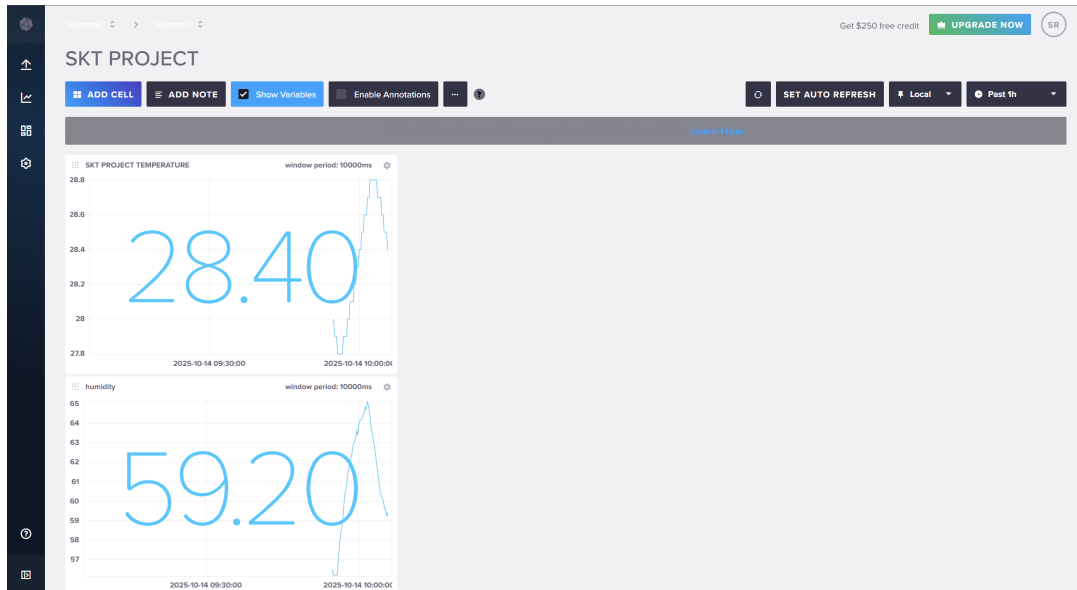
Tag `device` memudahkan pemisahan multiperangkat di masa depan, sedangkan *fields* memastikan setiap variabel numerik dapat diolah secara statistik.

Pada sisi *read*, kueri rentang waktu digerakkan melalui UI Influx atau *client* Python (lihat Lampiran). Hasilnya divisualisasikan menjadi panel nilai besar (single stat) dan tren. Gambar IV.2 memperlihatkan suhu 28.4°C dan kelembapan 59.2% yang berubah secara halus mengikuti siklus pembukaan damper dan aktivasi kipas. Panel lain membandingkan data lapangan dengan empat parameter dari DWSIM (inlet/outlet air panas–dingin). Garis horizontal yang relatif konstan pada kanal DWSIM menandakan kondisi operasi *steady* di sisi simulasi, sedangkan kanal lapangan memperlihatkan fluktuasi akibat dinamika lingkungan nyata.

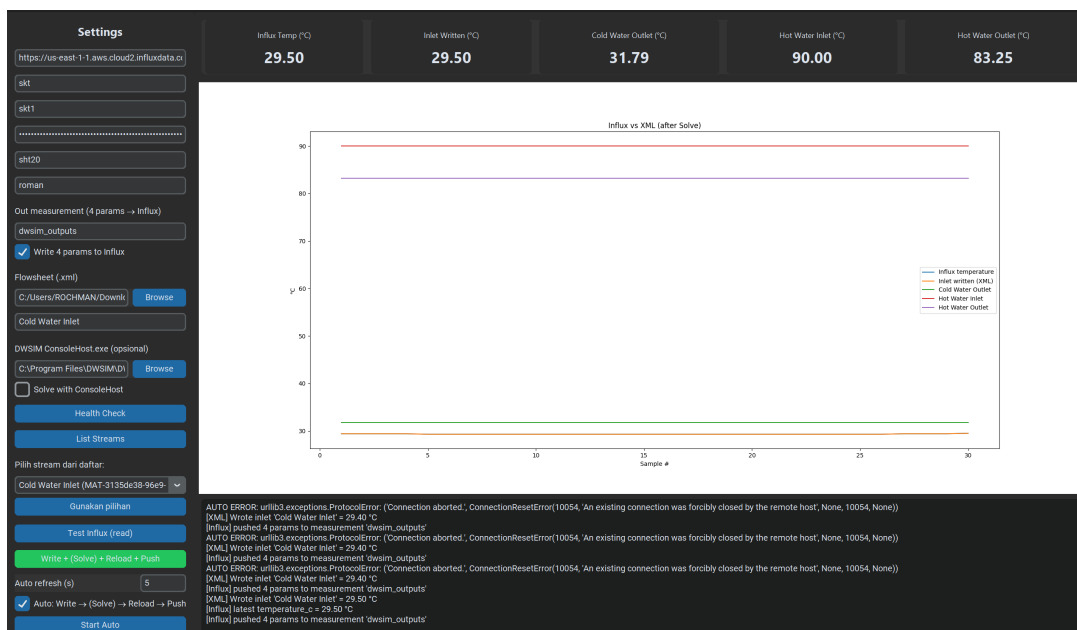
Kinerja *write* dipantau dari *status code*. Nilai 204 berarti sukses; selain itu, respons dicatat agar mudah *debug*. Pada satu percobaan, kami menemukan *error* “connection reset by host” ketika jaringan berganti SSID; solusi sementara ialah memperpanjang *timeout* HTTP dan menutup koneksi (`Connection: close`) untuk menghindari sesi tergantung. Selain itu, sanitasi karakter pada parameter `org` diperlukan karena Influx Cloud menerima baik `org` (nama) maupun `orgID` (UUID). Fungsi pembantu untuk *percent-encoding* disediakan agar URL aman.

Dari sisi analitik, struktur *fields* memudahkan pembuatan *alert* sederhana—misalnya ketika `relay_state` bernilai 1 namun suhu tetap di atas ambang dalam durasi panjang, menandakan kipas tidak efektif. Korelasi cepat dapat dibuat antara `servo_position_pct` dan penurunan suhu untuk mengukur efektivitas damper. Integrasi dengan DWSIM membuka peluang kalibrasi model termal ruangan; perbedaan yang konsisten antara “inlet written” dan “influx temperature” dapat dipakai menyetel koefisien perpindahan panas efektif.

Secara keseluruhan, integrasi Influx menyediakan visibilitas menyeluruh terhadap performa sistem, memudahkan *root-cause analysis*, dan menjadi dasar peningkatan berikutnya seperti *predictive control* berbasis tren.



Gambar IV.2: Panel suhu/kelembapan pada Influx.



Gambar IV.3: Perbandingan variabel DWSIM dan telemetri lapangan dalam satu dashboard.

4.4 Hasil Integrasi ke ThingsBoard

ThingsBoard dipilih sebagai antarmuka operasional karena kemudahan *provisioning* perangkat dan keragaman widget. Setelah mendapatkan kredensial *device* (token atau pasangan username/password), ESP32 mempublikasikan telemetri ke topik `v1/devices/me/telemetry` dalam format JSON. Payload memuat `temperature_c`,

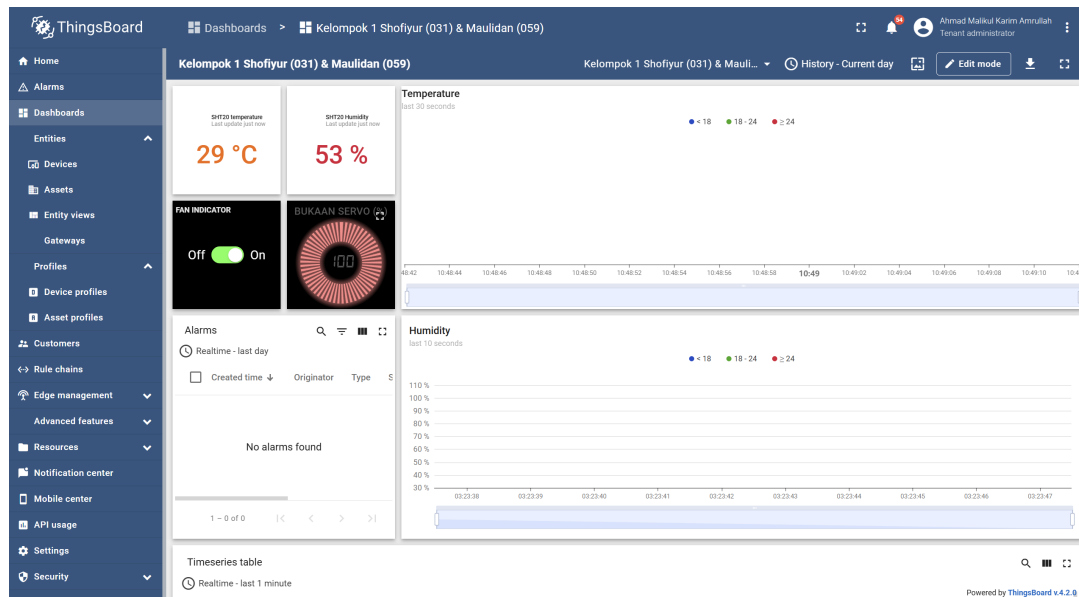
`humidity_pct`, `servo_position_pct`, dan `relay_state`. Dengan QoS 1, broker memastikan telemetry diterima minimal sekali, sehingga beberapa duplikasi ditangani pada sisi aplikasi.

Di dashboard, kami menampilkan empat elemen utama: *single stat* suhu, *single stat* kelembapan, *toggle* indikator kipas, dan *gauge* bukaan servo. Indikator kipas terhubung ke `relay_state` (0/1); sedangkan gauge menormalisasi 0–100% sehingga pengguna segera memahami posisi damper. Dua grafik *timeseries* menampilkan tren 10–60 menit terakhir. Dalam sesi pengujian, ketika suhu dinaikkan dengan pemanasan lokal di dekat sensor, respon sistem tampak jelas: servo bergerak menuju 180°, relay menyala, dan kurva suhu turun bertahap beberapa menit kemudian. Widget status menandai transisi ON/OFF kipas sehingga korelasi antar kejadian mudah diamati.

Keunggulan lain ThingsBoard adalah *rule engine*. Walau belum diaktifkan penuh, rancangan aturan sederhana telah dipersiapkan: jika suhu > 32°C selama 3 menit, kirim notifikasi; jika perangkat tidak mengirim telemetry selama 5 menit, tandai sebagai *offline*. Mekanisme ini penting untuk operasi jangka panjang di lapangan. Selain itu, *attributes* perangkat dapat digunakan untuk mengirimkan setelan jarak jauh seperti ambang suhu dan periode reset servo—fitur yang membuka opsi *closed-loop tuning* dari cloud.

Kualitas pengalaman pengguna meningkat karena *latency* MQTT rendah; perubahan pada perangkat hampir seketika tercermin pada dashboard. Pada sisi keamanan, koneksi dapat ditingkatkan ke MQTT over TLS bila diperlukan. Tantangan yang ditemui antara lain *rate limit* saat pengiriman terlalu sering; solusi praktis adalah menetapkan interval publikasi 5 detik yang masih memadai untuk pemantauan ventilasi namun ramah sumber daya.

Secara keseluruhan, integrasi ThingsBoard menyediakan antarmuka ramah operator untuk memonitor kondisi lingkungan, memverifikasi aksi aktuator, serta memperingatkan kondisi tidak normal. Ketika digabungkan dengan Influx untuk histori lengkap, keduanya membentuk ekosistem pemantauan dan pengendalian yang kuat serta mudah diperluas.



Gambar IV.4: Dashboard ThingsBoard yang digunakan dalam pengujian.

BAB V PEMBAHASAN

Hasil pengujian menunjukkan alur kerja yang konsisten: ESP32-S3 berhasil terhubung ke Wi-Fi dan broker ThingsBoard (MQTT). Pembacaan Modbus RTU dari SHT20 via MAX485 stabil setelah jeda DE yang memadai dan validasi CRC. Nilai suhu/kelembapan dipublikasikan ke ThingsBoard dan ditulis ke InfluxDB. Logika kontrol deterministik—ambang $27,2^{\circ}\text{C}$ untuk kipas dan $28,5^{\circ}\text{C}$ untuk posisi servo—menunjukkan perilaku yang mudah diprediksi. Mekanisme reset servo 20 detik mencegah *drift* sekaligus menjadi indikator *watchdog* visual. Integrasi DWSIM sebagai *digital twin* memungkinkan perbandingan terhadap kondisi *steady* sehingga deviasi lapangan cepat terdeteksi.

BAB VI KESIMPULAN DAN SARAN

6.1 Kesimpulan

Telah direalisasikan sistem **ventilasi otomatis** berbasis ESP32-S3 yang membaca suhu/kelembapan dari SHT20 (RS485/Modbus RTU) melalui MAX485, kemudian mengendalikan *exhaust fan* (relay) dan damper (servo via PWM). Data telemetri berhasil dipublikasikan melalui MQTT ke ThingsBoard dan disimpan di InfluxDB. Dokumentasi wiring dilengkapi klasifikasi warna kabel berdasarkan protokol.

6.2 Saran

Pengembangan lanjutan: (1) *fail-safe* jika sensor tidak respons; (2) histeresis/penjadwalan adaptif; (3) sensor tambahan CO₂/VOC; (4) alarm dan *remote override* di ThingsBoard; (5) penyaring EMI dan terminasi RS485 untuk instalasi jarak jauh.

(LAMPIRAN)

BAB A KODE ESP (RUST/ESP-IDF)

Cuplikan kode lengkap yang digunakan pada perangkat ESP32-S3.

Listing A.1: Kode utama ESP32-S3 (Rust) untuk Modbus, MQTT, Influx, servo, dan relay.

```
1 extern crate alloc;
2
3 use anyhow::{bail, Context, Result};
4 use log::{error, info, warn};
5
6 use esp_idf_sys as sys; // C-API (MQTT & HTTP)
7
8 use alloc::ffi::CString;
9 use alloc::string::String;
10 use alloc::string::ToString;
11
12 use esp_idf_svc::{
13     eventloop::EspSystemEventLoop,
14     log::EspLogger,
15     nvs::EspDefaultNvsPartition,
16     wifi::{AuthMethod, BlockingWifi, ClientConfiguration as
17         StaCfg, Configuration as WifiCfg, EspWifi},
18 };
19
20 use esp_idf_svc::hal::{
21     delay::FreeRtos,
22     gpio::{AnyIOPin, Output, PinDriver},
23     ledc::{config::TimerConfig, LedcDriver, LedcTimerDriver,
24         Resolution},
25     peripherals::Peripherals,
26     uart::{config::Config as UartConfig, UartDriver},
27     units::Hertz,
28 };
```



```

27
28 use serde_json::json;
29
30 // ===== KONFIGURASI =====
31 // Wi-Fi
32 const WIFI_SSID: &str = "jony";
33 const WIFI_PASS: &str = "12345678";
34
35 // ThingsBoard Demo (MQTT Basic)
36 const TB_MQTT_URL: &str = "mqtt://demo.thingsboard.io:1883";
37 const TB_CLIENT_ID: &str = "roman";
38 const TB_USERNAME: &str = "roman";
39 const TB_PASSWORD: &str = "12345";
40
41 // InfluxDB (Cloud atau lokal)
42 const INFLUX_URL: &str = "https://us-east-1-1.aws.cloud2.
    influxdata.com";
43 const INFLUX_ORG_ID: &str = "skt";
44 const INFLUX_BUCKET: &str = "skt1";
45 const INFLUX_TOKEN: &str = "4iveQHlkS17m-
    EArzJZKxVLXB5G9FP21VTcbujdk_pfm7NHBGe12DZDECdLt-
    LLNtTMvQXFI1qlx2clomBSe0w==";
46
47 // Modbus (SHT20 via RS485)
48 const MODBUS_ID: u8 = 0x01;
49 const BAUD: u32 = 9_600;
50
51 // Threshold kontrol (tetap sama)
52 const TEMP_SERVO_THRESHOLD: f32 = 28.5;
53 const RELAY_ON_THRESHOLD: f32 = 27.2;
54
55 // ===== Util =====
56 #[inline(always)]
57 fn ms_to_ticks(ms: u32) -> u32 {
58     (ms as u64 * sys::configTICK_RATE_HZ as u64 / 1000) as u32
59 }
60
61 fn looks_like_uuid(s: &str) -> bool {
62     s.len() == 36 && s.matches('-').count() == 4
63 }

```

```

64
65 // Minimal percent-encoding untuk komponen query
66 fn url_encode_component(input: &str) -> String {
67     let mut out = String::with_capacity(input.len());
68     for b in input.as_bytes() {
69         let c = *b as char;
70         if c.is_ascii_alphanumeric() || "-_~".contains(c) {
71             out.push(c); }
72         else { let _ = core::fmt::write(&mut out, format_args
73             !("%{:02X}", b)); }
74     }
75     out
76 }
77
78 // ===== MQTT client (C-API) =====
79 struct SimpleMqttClient { client: *mut sys::esp_mqtt_client }
80 impl SimpleMqttClient {
81     fn new(broker_url: &str, username: &str, password: &str,
82         client_id: &str) -> Result<Self> {
83         unsafe {
84             let broker_url_cstr = CString::new(broker_url)?;
85             let username_cstr = CString::new(username)?;
86             let password_cstr = CString::new(password)?;
87             let client_id_cstr = CString::new(client_id)?;
88             let mut cfg: sys::esp_mqtt_client_config_t = core
89                 ::mem::zeroed();
90             cfg.broker.address.uri = broker_url_cstr.as_ptr()
91                 as *const u8;
92             cfg.credentials.username = username_cstr.as_ptr()
93                 as *const u8;
94             cfg.credentials.client_id = client_id_cstr.as_ptr()
95                 () as *const u8;
96             cfg.credentials.authentication.password =
97                 password_cstr.as_ptr() as *const u8;
98             cfg.session.keepalive = 30;
99             cfg.network.timeout_ms = 20_000;
100             let client = sys::esp_mqtt_client_init(&cfg);
101             if client.is_null() { bail!("Failed to initialize
102                 MQTT client"); }
103             let err = sys::esp_mqtt_client_start(client);

```

```

105         if err != sys::ESP_OK { bail!("Failed to start
106             MQTT client, esp_err=0x{:X}", err as u32); }
107         sys::vTaskDelay(ms_to_ticks(2500));
108         Ok(Self { client })
109     }
110 }
111
112 fn publish(&self, topic: &str, data: &str) -> Result<()> {
113     unsafe {
114         let topic_c = CString::new(topic)?;
115         let msg_id = sys::esp_mqtt_client_publish(
116             self.client, topic_c.as_ptr(), data.as_ptr(),
117             data.len() as i32, 1, 0
118         );
119         if msg_id < 0 { bail!("Failed to publish message,
120             code: {}", msg_id); }
121         info!("MQTT published (id={})", msg_id);
122         Ok(())
123     }
124 }
125
126 impl Drop for SimpleMqttClient {
127     fn drop(&mut self) {
128         unsafe { sys::esp_mqtt_client_stop(self.client); sys::
129             esp_mqtt_client_destroy(self.client); }
130     }
131 }
132
133 // ===== CRC & Modbus util =====
134 fn crc16_modbus(mut crc: u16, byte: u8) -> u16 {
135     crc ^= byte as u16;
136     for _ in 0..8 { crc = if (crc & 1) != 0 { (crc >> 1) ^ 0
137         xA001 } else { crc >> 1 }; }
138     crc
139 }
140
141 fn modbus_crc(data: &[u8]) -> u16 { data.iter().fold(0xFFFFu16
142     , |acc, &b| crc16_modbus(acc, b)) }
143
144 fn build_read_req(slave: u8, func: u8, start_reg: u16, qty:
145     u16) -> heapless::Vec<u8, 256> {
146     use heapless::Vec;
147     let mut pdu: Vec<u8, 256> = Vec::new();

```

```

128 pdu.push(slave).unwrap(); pdu.push(func).unwrap();
129 pdu.push((start_reg >> 8) as u8).unwrap(); pdu.push((
    start_reg & 0xFF) as u8).unwrap();
130 pdu.push((qty >> 8) as u8).unwrap(); pdu.push((qty & 0xFF)
    as u8).unwrap();
131 let crc = modbus_crc(&pdu); pdu.push((crc & 0xFF) as u8).
    unwrap(); pdu.push((crc >> 8) as u8).unwrap();
132 pdu
133 }
134 fn parse_read_resp(expected_slave: u8, qty: u16, buf: &[u8])
-> Result<heapless::Vec<u16, 64>> {
135     use heapless::Vec;
136     if buf.len() >= 5 && (buf[1] & 0x80) != 0 {
137         let crc_rx = u16::from(buf[4]) << 8 | u16::from(buf
            [3]);
138         let crc_calc = modbus_crc(&buf[..3]);
139         if crc_rx == crc_calc { bail!("Modbus exception 0x{:02
            X}", buf[2]); }
140         else { bail!("Exception frame CRC mismatch"); }
141     }
142     let need = 1 + 1 + 1 + (2 * qty as usize) + 2;
143     if buf.len() < need { bail!("Response too short: got {},
        need {}", buf.len(), need); }
144     if buf[0] != expected_slave { bail!("Unexpected slave id:
        got {}, expected {}", buf[0], expected_slave); }
145     if buf[1] != 0x03 && buf[1] != 0x04 { bail!("Unexpected
        function code: 0x{:02X}", buf[1]); }
146     let bc = buf[2] as usize;
147     if bc != 2 * qty as usize { bail!("Unexpected byte count:
        {}", bc); }
148     let crc_rx = u16::from(buf[need - 1]) << 8 | u16::from(buf
        [need - 2]);
149     let crc_calc = modbus_crc(&buf[..need - 2]);
150     if crc_rx != crc_calc { bail!("CRC mismatch: rx=0x{:04X},
        calc=0x{:04X}", crc_rx, crc_calc); }
151
152     let mut out: Vec<u16, 64> = Vec::new();
153     for i in 0..qty as usize {
154         let hi = buf[3 + 2 * i] as u16; let lo = buf[3 + 2 * i
            + 1] as u16;

```

```

155         out.push((hi << 8) | lo).unwrap();
156     }
157     Ok(out)
158 }
159
160 // ===== RS485 helpers =====
161 fn rs485_write(uart: &UartDriver<'>, de: &mut PinDriver<'>,
162     esp_idf_svc::hal::gpio::Gpio21, Output>, data: &[u8]) ->
163     Result<()> {
164     de.set_high()?; FreeRtos::delay_ms(3); uart.write(data)?;
165     uart.wait_tx_done(200)?; de.set_low()?; FreeRtos::
166     delay_ms(3); Ok(())
167 }
168
169 fn rs485_read(uart: &UartDriver<'>, dst: &mut [u8], ticks:
170     u32) -> Result<usize> {
171     uart.clear_rx()?; let n = uart.read(dst, ticks)?;
172     use core::fmt::Write; let mut s = String::new(); for b in
173     &dst[..n] { write!(&mut s, "{:02X} ", b).ok(); }
174     info!("RS485 RX {} bytes: {}", n, s); Ok(n)
175 }
176
177 fn try_read(uart: &UartDriver<'>, de: &mut PinDriver<'>,
178     esp_idf_svc::hal::gpio::Gpio21, Output>, func: u8, start:
179     u16, qty: u16, ticks: u32)
180     -> Result<heapless::Vec<u16, 64>>
181 {
182     let req = build_read_req(MODBUS_ID, func, start, qty);
183     rs485_write(uart, de, &req)?; let mut buf = [0u8; 64]; let
184     n = rs485_read(uart, &mut buf, ticks)?;
185     parse_read_resp(MODBUS_ID, qty, &buf[..n])
186 }
187
188 fn probe_map(uart: &UartDriver<'>, de: &mut PinDriver<'>,
189     esp_idf_svc::hal::gpio::Gpio21, Output>) -> Option<(u8, u16
190     , u16)> {
191     for &fc in &[0x04u8, 0x03u8] {
192         for start in 0x0000u16..=0x0010u16 {
193             for &qty in &[1u16, 2u16] {
194                 if let Ok(regs) = try_read(uart, de, fc, start
195                     , qty, 250) {
196                     info!("FOUND: fc=0x{:02X}, start=0x{:04X},
197                         qty={}, regs={:04X?}", fc, start, qty,

```

```

182         regs.as_slice());
183         return Some((fc, start, qty));
184     }
185 }
186 }
187 None
188 }
189 fn read_sht20_with_map(uart: &UartDriver<'>, de: &mut
PinDriver<', esp_idf_svc::hal::gpio::Gpio21, Output>, fc:
u8, start: u16, qty: u16)
190     -> Result<(f32, f32)>
191 {
192     let regs = try_read(uart, de, fc, start, qty, 250)?;
193     let (raw_t, raw_h) = if regs.len() >= 2 { (regs[0], regs
[1]) } else { (regs[0], 0) };
194     let temp_c = (raw_t as f32) * 0.1;
195     let rh_pct = (raw_h as f32) * 0.1;
196     Ok((temp_c, rh_pct))
197 }
198
199 // ===== Wi-Fi (BlockingWifi)
=====
200 fn connect_wifi(wifi: &mut BlockingWifi<EspWifi<'static>>) ->
Result<()> {
201     let cfg = WifiCfg::Client(StaCfg {
202         ssid: heapless::String::try_from(WIFI_SSID).unwrap(),
203         password: heapless::String::try_from(WIFI_PASS).unwrap
(),
204         auth_method: AuthMethod::WPA2Personal,
205         channel: None,
206         ..Default::default()
207     });
208     wifi.set_configuration(&cfg)?; wifi.start()?; info!("Wi-Fi
driver started");
209     wifi.connect()?; info!("Wi-Fi connect issued, waiting for
netif up ...");
210     wifi.wait_netif_up()?; let ip = wifi.wifi().sta_netif().
get_ip_info()?;
211     info!("Wi-Fi connected. IP = {}", ip.ip);

```

```

212     unsafe { sys::vTaskDelay(ms_to_ticks(1200)); } Ok(())
213 }
214
215 // ===== Influx helpers =====
216 fn influx_line(measurement: &str, device: &str, t_c: f32,
217 h_pct: f32, relay_state: bool, servo_pct: u32) -> String {
218     format!("{}", device={}, temperature_c={}, humidity_pct={},
219         relay_state={}, servo_position_pct={},
220         measurement, device, t_c, h_pct, relay_state as u8,
221         servo_pct)
222 }
223
224 fn influx_write(lp: &str) -> Result<()> {
225     unsafe {
226         let org_q = if looks_like_uuid(INFLUX_ORG_ID) { "orgID"
227             " } else { "org" };
228         let url = format!("{}", api/v2/write?{}={} & bucket={} &
229             precision=ms",
230             INFLUX_URL, org_q, url_encode_component(
231                 INFLUX_ORG_ID), url_encode_component(
232                 INFLUX_BUCKET));
233         let url_c = CString::new(url.as_str())?;
234         let mut cfg: sys::esp_http_client_config_t = core::mem
235             ::zeroed();
236         cfg.url = url_c.as_ptr(); cfg.method = sys::
237             esp_http_client_method_t_HTTP_METHOD_POST;
238         if INFLUX_URL.starts_with("https://") {
239             cfg.transport_type = sys::
240                 esp_http_client_transport_t_HTTP_TRANSPORT_OVER_SSL
241                 ;
242             cfg.crt_bundle_attach = Some(sys::
243                 esp_crt_bundle_attach);
244         }
245         let client = sys::esp_http_client_init(&cfg);
246         if client.is_null() { bail!("esp_http_client_init
247             failed"); }
248
249         let h_auth = CString::new("Authorization")?;
250         let v_auth = CString::new(format!("Token {}",
251             INFLUX_TOKEN))?;
252         let h_ct = CString::new("Content-Type")?;

```

```

238 let v_ct = CString::new("text/plain; charset=utf-8"?;
239 let h_acc = CString::new("Accept"?;
240 let v_acc = CString::new("application/json"?;
241 let h_conn = CString::new("Connection"?;
242 let v_conn = CString::new("close"?;
243 sys::esp_http_client_set_header(client, h_auth.as_ptr
    (), v_auth.as_ptr());
244 sys::esp_http_client_set_header(client, h_ct.as_ptr(),
    v_ct.as_ptr());
245 sys::esp_http_client_set_header(client, h_acc.as_ptr()
    , v_acc.as_ptr());
246 sys::esp_http_client_set_header(client, h_conn.as_ptr
    (), v_conn.as_ptr());
247
248 sys::esp_http_client_set_post_field(client, lp.as_ptr
    (), lp.len() as i32);
249 let err = sys::esp_http_client_perform(client);
250 if err != sys::ESP_OK {
251     let e = format!("esp_http_client_perform failed: 0
        x{:X}", err as u32);
252     sys::esp_http_client_cleanup(client); bail!(e);
253 }
254 let status = sys::esp_http_client_get_status_code(
    client);
255 if status != 204 {
256     let mut body_buf = [0u8; 256];
257     let read = sys::esp_http_client_read_response(
        client, body_buf.as_mut_ptr(), body_buf.len()
        as i32);
258     let body = if read > 0 { core::str::from_utf8(&
        body_buf[..read as usize]).unwrap_or("") } else
        { "" };
259     warn!("Influx write failed: HTTP {} Body: {}",
        status, body);
260     sys::esp_http_client_cleanup(client); bail!("
        Influx write HTTP status {}", status);
261 } else {
262     info!("    Data Berhasil Dikirim ke InfluxDB");
263 }
264 sys::esp_http_client_cleanup(client); Ok(())

```



```

265     }
266 }
267
268 // ===== Servo helpers (LEDC)
269 // =====
270 struct Servo { ch: LeducDriver<'static>, duty_0: u32, duty_90:
271     u32, duty_180: u32 }
272 impl Servo {
273     fn new(mut ch: LeducDriver<'static>) -> Result<Self> {
274         let max = ch.get_max_duty() as u64; let period_us = 20
275             _000u64;
276         let duty_from_us = |us: u32| -> u32 { ((max * us as
277             u64) / period_us) as u32 };
278         let duty_0 = duty_from_us(500); let duty_90 =
279             duty_from_us(1500); let duty_180 = duty_from_us
280                 (2500);
281         ch.set_duty(duty_90)?; ch.enable()?; Ok(Self { ch,
282             duty_0, duty_90, duty_180 })
283     }
284     fn set_0(&mut self) -> Result<()> { self.ch.set_duty(self.
285         duty_0).map_err(Into::into) }
286     fn set_90(&mut self) -> Result<()> { self.ch.set_duty(self
287         .duty_90).map_err(Into::into) }
288     fn set_180(&mut self) -> Result<()> { self.ch.set_duty(
289         self.duty_180).map_err(Into::into) }
290 }
291 #[derive(Copy, Clone, PartialEq, Eq, Debug)] enum ServoPos {
292     P0, P90, P180 }
293
294 // ===== Relay helpers =====
295 struct Relay { pin: PinDriver<'static, esp_idf_svc::hal::gpio
296     ::Gpio9, Output> }
297 impl Relay {
298     fn new(pin: PinDriver<'static, esp_idf_svc::hal::gpio::
299         Gpio9, Output>) -> Result<Self> { let mut r = Self {
300         pin }; r.off()?; Ok(r) }
301     fn on(&mut self) -> Result<()> { self.pin.set_high().
302         map_err(Into::into) }
303     fn off(&mut self) -> Result<()> { self.pin.set_low().
304         map_err(Into::into) }

```

```

289     fn is_on(&self) -> bool { self.pin.is_set_high() }
290 }
291
292 // ===== Validasi & kontrol =====
293 fn is_zero_glitch(t: f32, h: f32) -> bool { t == 0.0 && h ==
    0.0 }
294
295 fn valid_sensor(t: f32, h: f32) -> bool {
296     // Batas fisik + anti-glitch nol serentak
297     !(t < -40.0 || t > 125.0 || h < 0.0 || h > 100.0 ||
        is_zero_glitch(t, h))
298 }
299
300 fn telemetry_payload(device: &str, t: f32, h: f32, servo_pct:
    u32, relay_state: bool) -> String {
301     let ts_ms = unsafe { sys::esp_timer_get_time() } / 1000;
302     json!({
303         "sensor": "sht20",
304         "temperature_c": (t * 10.0).round() / 10.0,
305         "humidity_pct": (h * 10.0).round() / 10.0,
306         "servo_position_pct": servo_pct,
307         "relay_state": relay_state,
308         "ts_ms": ts_ms
309     }).to_string()
310 }
311
312 fn publish_and_influx(mqtt: &SimpleMqttClient, topic: &str,
    device: &str, t: f32, h: f32, relay_state: bool, servo_pct:
    u32) {
313     let payload = telemetry_payload(device, t, h, servo_pct,
        relay_state);
314     if let Err(e) = mqtt.publish(topic, &payload) { error!("
        MQTT publish error: {e:?}"); }
315     let lp = influx_line("sht20", device, (t * 10.0).round() /
        10.0, (h * 10.0).round() / 10.0, relay_state,
        servo_pct);
316     if let Err(e) = influx_write(&lp) { warn!("Influx write
        failed: {e}"); }
317 }
318

```

```

319 fn servo_pct_from(pos: ServoPos) -> u32 { match pos { ServoPos
    ::P0 => 0, ServoPos::P90 => 50, ServoPos::P180 => 100 } }
320
321 fn apply_servo_and_relay(t: f32, servo: &mut Servo, servo_pos:
    &mut ServoPos, relay: &mut Relay, log_suffix: &str) {
322     // Servo: <28.5      0 , >28.5      180
323     if t > TEMP_SERVO_THRESHOLD {
324         if *servo_pos != ServoPos::P180 {
325             if let Err(e) = servo.set_180() { error!("Servo
                set 180 error: {e:?}"); }
326             else { info!("      Servo      180 (T={:.1} C ){
                log_suffix}", t); *servo_pos = ServoPos::P180;
                }
327         }
328     } else if t < TEMP_SERVO_THRESHOLD {
329         if *servo_pos != ServoPos::P0 {
330             if let Err(e) = servo.set_0() { error!("Servo set
                0 error: {e:?}"); }
331             else { info!("      Servo      0 (T={:.1} C ){
                log_suffix}", t); *servo_pos = ServoPos::P0; }
332         }
333     } else {
334         info!("T=28.5 C persis      Servo tetap di {:.1}",
            servo_pos);
335     }
336
337     // Relay: >=27.2      ON, else OFF
338     if t >= RELAY_ON_THRESHOLD {
339         if !relay.is_on() {
340             if let Err(e) = relay.on() { error!("Relay on
                error: {e:?}"); }
341             else { info!("      Relay      ON (T={:.1} C )", t
                ); }
342         }
343     } else if relay.is_on() {
344         if let Err(e) = relay.off() { error!("Relay off error:
            {e:?}"); }
345         else { info!("      Relay      OFF (T={:.1} C )", t);
            }
346     }

```

```

347 }
348
349 // ===== IO siklus (baca + (opsional) publish/
    Influx) =====
350 fn do_sensor_io(
351     uart: &UartDriver<'_>,
352     de_pin: &mut PinDriver<'_, esp_idf_svc::hal::gpio::Gpio21,
        Output>,
353     fc_use: u8, start_use: u16, qty_use: u16,
354     mqtt: &SimpleMqttClient,
355     topic_tele: &str,
356     servo_pos: ServoPos,
357     relay: &Relay,
358 ) -> Result<(f32, f32, u32, bool)> {
359     let mut retries = 3;
360     let mut last_error = None;
361
362     while retries > 0 {
363         match read_sht20_with_map(uart, de_pin, fc_use,
            start_use, qty_use) {
364             Ok((t, h)) => {
365                 let t_rounded = (t * 10.0).round() / 10.0;
366                 let h_rounded = (h * 10.0).round() / 10.0;
367                 let servo_pct = servo_pct_from(servo_pos);
368                 let relay_state = relay.is_on();
369
370                 // ==== Anti-glitch: kedua nilai nol
                    JANGAN kirim apa pun
371                 if is_zero_glitch(t_rounded, h_rounded) {
372                     warn!("Zero-glitch detected (T=0.0, H=0.0)
                        . Skipping MQTT & Influx.");
373                     return Ok((t, h, servo_pct, relay_state));
374                 }
375
376                 // Validasi fisik
377                 if valid_sensor(t_rounded, h_rounded) {
378                     println!("{}", json!({
379                         "sensor": "sht20",
380                         "temperature_c": t_rounded,
381                         "humidity_pct": h_rounded,

```

```

382         "servo_position_pct": servo_pct,
383         "relay_state": relay_state,
384         "ts_ms": (unsafe { sys::
                    esp_timer_get_time() } / 1000)
385     }).to_string());
386     // kirim
387     publish_and_influx(mqtt, topic_tele,
                        TB_CLIENT_ID, t_rounded, h_rounded,
                        relay_state, servo_pct);
388 } else {
389     warn!("Invalid sensor data: temperature_c
           ={:1}, humidity_pct={:1}. Skipping
           MQTT and InfluxDB publish.", t_rounded,
           h_rounded);
390 }
391 return Ok((t, h, servo_pct, relay_state));
392 }
393 Err(e) => {
394     retries -= 1;
395     last_error = Some(e);
396     warn!("Modbus read error (retry {}): {:?} ",
           retries, last_error);
397     FreeRtos::delay_ms(100);
398 }
399 }
400 }
401 Err(last_error.unwrap_or_else(|| anyhow::anyhow!("Modbus
           read failed after retries")))
402 }
403
404 // ===== main =====
405 fn main() -> Result<()> {
406     // ESP-IDF init
407     sys::link_patches();
408     EspLogger::initialize_default();
409     info!("          Modbus RS485 + ThingsBoard MQTT Basic +
           InfluxDB + Servo (GPIO10) + Relay (GPIO9)");
410
411     // Peripherals & services
412     let peripherals = Peripherals::take().context("Peripherals

```

```

    ::take")?;
413 let pins = peripherals.pins;
414 let sys_loop = EspSystemEventLoop::take().context("
    eventloop")?;
415 let nvs = EspDefaultNvsPartition::take().context("nvs")?;
416
417 // Wi-Fi via BlockingWifi
418 let mut wifi = BlockingWifi::wrap(
419     EspWifi::new(peripherals.modem, sys_loop.clone(), Some
        (nvs))?,
420     sys_loop,
421 )?;
422 connect_wifi(&mut wifi)?;
423
424 // MQTT ThingsBoard (Basic)
425 let mqtt = SimpleMqttClient::new(TB_MQTT_URL, TB_USERNAME,
    TB_PASSWORD, TB_CLIENT_ID)?;
426 info!("MQTT connected to {}", TB_MQTT_URL);
427
428 // UART0 + RS485 (GPIO43/44, DE: GPIO21)
429 let tx = pins.gpio43; // U0TXD
430 let rx = pins.gpio44; // U0RXD
431 let de = pins.gpio21;
432 let cfg = UartConfig::new().baudrate(Hertz(BAUD));
433 let uart = UartDriver::new(peripherals.uart0, tx, rx, None
    ::<AnyIOPin>, None::<AnyIOPin>, &cfg)
434     .context("UartDriver::new")?;
435 let mut de_pin = PinDriver::output(de).context("PinDriver
    ::output(DE)")?;
436 de_pin.set_low()?; // default RX
437 info!("UART0 ready (TX=GPIO43, RX=GPIO44, DE=GPIO21), {}
    bps", BAUD);
438
439 // Servo init (LEDC 50 Hz, 14-bit, GPIO10)
440 let ledc = peripherals.ledc;
441 let mut servo_timer = LedcTimerDriver::new(
442     ledc.timer0,
443     &TimerConfig { frequency: Hertz(50), resolution:
        Resolution::Bits14, ..Default::default() },
444 )?;

```

```

445 let servo_channel = LcdcDriver::new(ledc.channel0, &mut
    servo_timer, pins.gpio10)?;
446 let mut servo = Servo::new(servo_channel)?;
447 let mut servo_pos = ServoPos::P90; // posisi awal 90
448
449 // Relay init (GPIO9)
450 let mut relay = Relay::new(PinDriver::output(pins.gpio9).
    context("PinDriver::output(Relay)")??);
451 info!("Relay initialized on GPIO9, initial state: off");
452
453 // Tanda waktu siklus reset servo (ms)
454 let mut next_reset_ms: u64 = unsafe { (sys::
    esp_timer_get_time() as u64) / 1000 } + 20_000;
455
456 // Probe mapping registri SHT20 (opsional)
457 let (mut fc_use, mut start_use, mut qty_use) = (0x04u8, 0
    x0000u16, 2u16);
458 if let Some((fc, start, qty)) = probe_map(&uart, &mut
    de_pin) {
459     (fc_use, start_use, qty_use) = (fc, start, qty);
460     info!("Using map: fc=0x{:02X}, start=0x{:04X}, qty={}"
        , fc_use, start_use, qty_use);
461 } else {
462     warn!("Probe failed. Fallback map: fc=0x{:02X}, start
        =0x{:04X}, qty={}", fc_use, start_use, qty_use);
463 }
464
465 // Loop utama
466 let topic_tele = "v1/devices/me/telemetry";
467 loop {
468     let now_ms: u64 = unsafe { (sys::esp_timer_get_time()
        as u64) / 1000 };
469     let is_reset_cycle = now_ms >= next_reset_ms;
470
471     if is_reset_cycle {
472         // === Reset siklus 20 detik        kembali ke 90
        ===
473         if servo_pos != ServoPos::P90 {
474             if let Err(e) = servo.set_90() { error!("Servo
                reset 90    error: {e:?}"); }

```

```

475         else { info!("          Reset siklus 20s: Servo
476                90 "); servo_pos = ServoPos::P90; }
477     } else {
478         info!("          Reset siklus 20s: Servo sudah di
479                90 ");
480     }
481 }
482
483 // Baca sensor + publish (+Influx) jika valid (anti-
484 // glitch di dalam do_sensor_io)
485 match do_sensor_io(&uart, &mut de_pin, fc_use,
486 start_use, qty_use, &mqtt, topic_tele, servo_pos, &
487 relay) {
488     Ok((t, h, _servo_pct, _relay_state)) => {
489         // Jeda 5 detik setelah mendapat data
490         FreeRtos::delay_ms(5_000);
491
492         // Jika data valid & bukan zero-glitch
493         baru kontrol aktuator
494         let t_rounded = (t * 10.0).round() / 10.0;
495         let h_rounded = (h * 10.0).round() / 10.0;
496         if valid_sensor(t_rounded, h_rounded) {
497             apply_servo_and_relay(t, &mut servo, &mut
498 servo_pos, &mut relay,
499             if is_reset_cycle { " setelah reset" }
500             else { "" });
501         } else {
502             // termasuk kasus zero-glitch, maka skip
503             kontrol
504             warn!("Skip actuator due to invalid sample
505                   : T={:.1} H={:.1}", t_rounded,
506                   h_rounded);
507         }
508     }
509     Err(e) => error!("Modbus read error{}: {e:?}", if
510 is_reset_cycle { " (after 20s reset)" } else {
511 "" }),
512 }
513
514 if is_reset_cycle { next_reset_ms = now_ms + 20_000; }

```



```
502
503     // Delay kecil agar loop tidak terlalu ketat
504     FreeRtos::delay_ms(1000);
505 }
506 }
```

BAB B KODE PYTHON (GATEWAY/VALIDASI)

Skrip utilitas untuk uji tulis-baca Influx dan ringkasan terminal.

Listing B.1: Skrip Python ringkas untuk uji InfluxDB dan logging.

```
1 import time
2 import threading
3 import traceback
4 from pathlib import Path
5 from typing import Dict, Optional, List, Tuple
6 import subprocess
7
8 import customtkinter as ctk
9 from tkinter import filedialog, messagebox
10
11 from influxdb_client import InfluxDBClient, Point
12 from influxdb_client.client.write_api import SYNCHRONOUS
13 import requests
14 import xml.etree.ElementTree as ET
15
16 from matplotlib.backends.backend_tkagg import
    FigureCanvasTkAgg
17 from matplotlib.figure import Figure
18
19
20 # ===== DEFAULTS =====
21 DEFAULTS = {
22     # ---- Influx (read) ----
23     "influx_url": "https://us-east-1-1.aws.cloud2.influxdata.
        com",
24     "influx_org": "skt",
25     "influx_bucket": "skt1",
26     "influx_token": "",
```

```

27     "measurement": "sht20",
28     "device_tag": "roman",
29     "range_minutes": 30,
30     "refresh_secs": 5,
31
32     # ---- Influx (write back) ----
33     "out_measurement": "dwsim_outputs", # measurement tujuan
        4 parameter
34
35     # ---- DWSIM ----
36     "inlet_key": "Cold Water Inlet",
37     "out_keys": ("Cold Water Outlet", "Hot Water Inlet", "Hot
        Water Outlet"),
38     "consolehost": r"C:\Program Files\DWSIM\DWSIM.ConsoleHost.
        exe",
39     "solve_timeout_sec": 60,
40 }
41
42
43 # ===== Influx helpers
        =====
44 def influx_health(url: str) -> str:
45     r = requests.get(url.rstrip("/") + "/health", timeout=15)
46     r.raise_for_status()
47     return r.text
48
49
50 def _query_with_retry(client: InfluxDBClient, flux: str, tries
: int = 4):
51     last = None
52     for i in range(tries):
53         try:
54             return client.query_api().query_data_frame(flux)
55         except Exception as e:
56             last = e
57             time.sleep(1.5 * (i + 1))
58     raise last
59
60
61 def get_latest_temperature_c(

```

```

62     url: str,
63     token: str,
64     org: str,
65     bucket: str,
66     measurement: str,
67     device_tag: str,
68     range_minutes: int = 30,
69 ) -> Optional[float]:
70     flux = f'''
71     from(bucket: "{bucket}")
72         |> range(start: -{int(range_minutes)}m)
73         |> filter(fn: (r) => r["_measurement"] == "{measurement}"")
74         |> filter(fn: (r) => r["device"] == "{device_tag}")
75         |> filter(fn: (r) => r["_field"] == "temperature_c")
76         |> keep(columns: ["_time", "_value"])
77         |> last()
78     '''
79     with InfluxDBClient(url=url, token=token, org=org, timeout
80         =20000) as client:
81         df = _query_with_retry(client, flux, tries=4)
82         if isinstance(df, list) and len(df):
83             df = df[0]
84         if df is None or df.empty:
85             return None
86         try:
87             return float(df["_value"].iloc[-1])
88         except Exception:
89             return None
90
91 def safe_float(x) -> Optional[float]:
92     try:
93         v = float(x)
94         if v != v: # NaN
95             return None
96         return v
97     except Exception:
98         return None
99

```

```

100
101 def write_four_params_to_influx_sync(
102     url: str,
103     token: str,
104     org: str,
105     bucket: str,
106     measurement: str,
107     device_tag: str,
108     flowsheet_path: str,
109     inlet_c: Optional[float],
110     cold_out_c: Optional[float],
111     hot_in_c: Optional[float],
112     hot_out_c: Optional[float],
113 ):
114     """
115     Tulis 4 parameter sebagai 4 point TERPISAH (SYNCHRONOUS).
116     Field names:
117         - cold_water_inlet_c
118         - cold_water_outlet_c
119         - hot_water_inlet_c
120         - hot_water_outlet_c
121     """
122     fs_name = Path(flowsheet_path).stem
123
124     rows: List[Point] = []
125     def add(field_name: str, value: Optional[float]):
126         val = safe_float(value)
127         if val is None:
128             return
129         p = (
130             Point(measurement)
131             .tag("device", device_tag)
132             .tag("flowsheet", fs_name)
133             .field(field_name, val)
134         )
135         rows.append(p)
136
137     add("cold_water_inlet_c", inlet_c)
138     add("cold_water_outlet_c", cold_out_c)
139     add("hot_water_inlet_c", hot_in_c)

```

```

140     add("hot_water_outlet_c", hot_out_c)
141
142     if not rows:
143         return 0
144
145     with InfluxDBClient(url=url, token=token, org=org, timeout
146                        =20000) as client:
147         wa = client.write_api(write_options=SYNCHRONOUS) #
148             langsung commit
149         wa.write(bucket=bucket, org=org, record=rows)
150         # SYNCHRONOUS: tidak perlu flush/close manual
151     return len(rows)
152
153 # ===== DWSIM XML helpers
154 =====
155 def open_flowsheet_xml(path: str) -> ET.ElementTree:
156     p = Path(path)
157     if p.suffix.lower() != ".xml":
158         raise ValueError("Gunakan file .xml (bukan .dwxmlz).")
159     return ET.parse(p)
160
161 def save_flowsheet_xml(tree: ET.ElementTree, path: str):
162     tree.write(path, encoding="utf-8", xml_declaration=True)
163
164 def build_stream_index(root: ET.Element) -> Tuple[Dict[str,
165 str], Dict[str, str], Dict[str, ET.Element]]:
166     id_to_tag: Dict[str, str] = {}
167     tag_to_id: Dict[str, str] = {}
168     for go in root.findall(".//GraphicObject[ObjectType='
169 MaterialStream']"):
170         id_el = go.find("Name"); tag_el = go.find("Tag")
171         sid = (id_el.text or "").strip() if id_el is not None
172             else ""
173         tag = (tag_el.text or "").strip() if tag_el is not
174             None else ""
175         if sid and tag:
176             id_to_tag[sid] = tag

```

```

173         tag_to_id[tag] = sid
174
175     id_to_sim: Dict[str, ET.Element] = {}
176     for so in root.findall("./SimulationObject[Type='DWSIM.
177         Thermodynamics.Streams.MaterialStream']"):
178         name_el = so.find("Name")
179         sid = (name_el.text or "").strip() if name_el is not
180             None else ""
181         if sid:
182             id_to_sim[sid] = so
183
184     return id_to_tag, tag_to_id, id_to_sim
185
186 def list_streams_display(path: str) -> List[str]:
187     tree = open_flowsheet_xml(path)
188     root = tree.getroot()
189     id_to_tag, _, _ = build_stream_index(root)
190     nice = [f"{tag} ({sid})" for sid, tag in id_to_tag.items()
191         ]
192     nice.sort()
193     return nice
194
195 def resolve_key_to_id(root: ET.Element, key: str) -> Optional[
196     str]:
197     id_to_tag, tag_to_id, _ = build_stream_index(root)
198     k = (key or "").strip()
199     if not k:
200         return None
201     if k in tag_to_id:
202         return tag_to_id[k]
203     if k in id_to_tag:
204         return k
205     for tag in tag_to_id:
206         if k.lower() in tag.lower():
207             return tag_to_id[tag]
208     for sid in id_to_tag:
209         if k.lower() in sid.lower():
210             return sid

```

```

209     return None
210
211
212 def get_stream_temperature_c_by_node(sim_node: Optional[ET.
    Element]) -> Optional[float]:
213     if sim_node is None:
214         return None
215     temp_el = sim_node.find("./Phase[ComponentName='Mixture
        ']/Properties/temperature")
216     if temp_el is not None and (temp_el.text or "").strip():
217         try:
218             return float(temp_el.text) - 273.15
219         except Exception:
220             return None
221     return None
222
223
224 def set_stream_temperature_c_by_node(sim_node: ET.Element,
    temp_c: float) -> bool:
225     temp_el = sim_node.find("./Phase[ComponentName='Mixture
        ']/Properties/temperature")
226     if temp_el is not None:
227         temp_el.text = f"{float(temp_c) + 273.15:.6f}" #
            Kelvin
228         return True
229     return False
230
231
232 def write_inlet_temperature(path: str, inlet_key: str, temp_c:
    float):
233     tree = open_flowsheet_xml(path)
234     root = tree.getroot()
235     _, _, id_to_sim = build_stream_index(root)
236     sid = resolve_key_to_id(root, inlet_key)
237     if not sid:
238         raise ValueError(f"Stream '{inlet_key}' tidak
            ditemukan (pakai List Streams + Gunakan pilihan).")
239     node = id_to_sim.get(sid)
240     if not node:
241         raise ValueError(f"SimulationObject untuk ID '{sid}'

```



```

                tidak ditemukan.")
242     if not set_stream_temperature_c_by_node(node, temp_c):
243         raise RuntimeError("Gagal set temperature (cek skema
                XML).")
244     save_flowsheet_xml(tree, path)
245
246
247 def read_outputs(path: str, keys: List[str]) -> Dict[str,
Optional[float]]:
248     tree = open_flowsheet_xml(path)
249     root = tree.getroot()
250     _, _, id_to_sim = build_stream_index(root)
251     out: Dict[str, Optional[float]] = {}
252     for key in keys:
253         sid = resolve_key_to_id(root, key)
254         node = id_to_sim.get(sid) if sid else None
255         out[key] = get_stream_temperature_c_by_node(node)
256     return out
257
258
259 # ===== DWSIM ConsoleHost (opsional)
    =====
260 def solve_with_consolehost(consolehost_path: str, input_xml:
str, timeout_sec: int = 60) -> str:
261     exe = Path(consolehost_path)
262     if not exe.exists():
263         raise FileNotFoundError(f"ConsoleHost tidak ditemukan:
                {consolehost_path}")
264     # format umum
265     args = f'"{exe}" -solve -input "{input_xml}" -output "{
                input_xml}"'
266     proc = subprocess.run(args, shell=True, capture_output=
                True, text=True, timeout=timeout_sec)
267     if proc.returncode != 0:
268         raise RuntimeError(f"ConsoleHost exit {proc.returncode
                }\nSTDOUT:\n{proc.stdout}\nSTDERR:\n{proc.stderr}")
269     return proc.stdout or ""
270
271
272 # ===== UI =====

```

```

273 class App(ctk.CTk):
274     def __init__(self):
275         super().__init__()
276         ctk.set_appearance_mode("dark")
277         ctk.set_default_color_theme("blue")
278         self.title("Influx      DWSIM XML (Auto Solve & Push 4
                Params)")
279         self.geometry("1220x780")
280         self.minsize(1060, 680)
281
282         # timeseries buffers
283         self.series_influx: List[float] = []
284         self.series_inlet_written: List[float] = []
285         self.series_out_cold: List[Optional[float]] = []
286         self.series_out_hot_in: List[Optional[float]] = []
287         self.series_out_hot_out: List[Optional[float]] = []
288
289         self._auto = False
290
291         self._build_sidebar()
292         self._build_main()
293         self._apply_defaults()
294
295         # ----- Build UI -----
296         def _build_sidebar(self):
297             sb = ctk.CTkFrame(self, width=380, corner_radius=0)
298             sb.pack(side="left", fill="y")
299             self.sidebar = sb
300
301             ctk.CTkLabel(sb, text="Settings", font=("Segoe UI",
                18, "bold")).pack(pady=(16, 8))
302
303             # Influx (read)
304             self.influx_url = ctk.CTkEntry(sb, placeholder_text="
                Influx URL")
305             self.influx_org = ctk.CTkEntry(sb, placeholder_text="
                Influx Org/OrgID")
306             self.influx_bucket = ctk.CTkEntry(sb, placeholder_text=
                "Influx Bucket")
307             self.influx_token = ctk.CTkEntry(sb, placeholder_text=

```

```

308         "Influx Token", show=" ")
self.measurement = ctk.CTkEntry(sb, placeholder_text="
309         Measurement (mis. sht20)")
self.device_tag = ctk.CTkEntry(sb, placeholder_text="
310         Device tag (mis. roman)")
for w in [self.influx_url, self.influx_org, self.
311         influx_bucket, self.influx_token, self.measurement,
312         self.device_tag]:
313         w.pack(padx=12, pady=6, fill="x")
314
# Influx (write back)
ctk.CTkLabel(sb, text="Out measurement (4 params
315         Influx)", anchor="w").pack(padx=12, pady=(6, 0),
316         fill="x")
self.out_measurement = ctk.CTkEntry(sb,
317         placeholder_text="dwsim_outputs")
self.out_measurement.pack(padx=12, pady=(4, 6), fill="
318         x")
self.push_enabled = ctk.BooleanVar(value=True)
self.chk_push = ctk.CTkCheckBox(sb, text="Write 4
319         params to Influx", variable=self.push_enabled)
self.chk_push.pack(padx=12, pady=(0, 8), anchor="w")
320
# Flow file
ctk.CTkLabel(sb, text="Flowsheet (.xml)", anchor="w").
321         pack(padx=12, pady=(6, 0), fill="x")
row = ctk.CTkFrame(sb); row.pack(padx=12, pady=(4, 6),
322         fill="x")
self.flowsheet = ctk.CTkEntry(row, placeholder_text=r"
323         C:\...\flowsheet.xml")
self.flowsheet.pack(side="left", expand=True, fill="x"
324         )
ctk.CTkButton(row, text="Browse", width=84, command=
325         self._browse_xml).pack(side="left", padx=(6, 0))
326
# inlet key
self.inlet_key = ctk.CTkEntry(sb, placeholder_text="
327         Cold Water Inlet (Tag atau ID)")
self.inlet_key.pack(padx=12, pady=(4, 8), fill="x")
328
329
330
331

```

```

332 # ConsoleHost (opsional)
333 ctk.CTkLabel(sb, text="DWSIM ConsoleHost.exe (opsional
    )", anchor="w").pack(padx=12, pady=(6, 0), fill="x"
    )
334 rowc = ctk.CTkCTkFrame = ctk.CTkFrame(sb); rowc.pack(
    padx=12, pady=(4, 6), fill="x")
335 self.consolehost = ctk.CTkEntry(rowc, placeholder_text
    =r"C:\Program Files\DWSIM\DWSIM.ConsoleHost.exe")
336 self.consolehost.pack(side="left", expand=True, fill="
    x")
337 ctk.CTkButton(rowc, text="Browse", width=84, command=
    self._browse_consolehost).pack(side="left", padx
    =(6, 0))
338 self.solve_enabled = ctk.BooleanVar(value=False)
339 self.chk_solve = ctk.CTkCheckBox(sb, text="Solve with
    ConsoleHost", variable=self.solve_enabled)
340 self.chk_solve.pack(padx=12, pady=(0, 8), anchor="w")
341
342 # helper buttons
343 self.btn_health = ctk.CTkButton(sb, text="Health Check
    ", command=self._on_health)
344 self.btn_health.pack(padx=12, pady=(4, 6), fill="x")
345
346 self.btn_streams = ctk.CTkButton(sb, text="List
    Streams", command=self._on_list_streams)
347 self.btn_streams.pack(padx=12, pady=6, fill="x")
348
349 ctk.CTkLabel(sb, text="Pilih stream dari daftar:",
    anchor="w").pack(padx=12, pady=(6, 0), fill="x")
350 self.stream_options = ctk.CTkComboBox(sb, values=[],
    state="readonly")
351 self.stream_options.pack(padx=12, pady=(4, 6), fill="x
    ")
352 ctk.CTkButton(sb, text="Gunakan pilihan", command=self
    ._use_selected_stream).pack(padx=12, pady=(0, 8),
    fill="x")
353
354 # actions
355 self.btn_test = ctk.CTkButton(sb, text="Test Influx (
    read)", command=self._on_test_influx)

```

```

356 self.btn_write = ctk.CTkButton(sb, text="Write + (
      Solve) + Reload + Push", command=self.
      _on_write_solve_reload, fg_color="#22c55e",
      hover_color="#16a34a")
357 self.btn_test.pack(padx=12, pady=6, fill="x")
358 self.btn_write.pack(padx=12, pady=(6, 6), fill="x")
359
360 # auto section
361 row2 = ctk.CTkFrame(sb); row2.pack(padx=12, pady=(4,
      6), fill="x")
362 ctk.CTkLabel(row2, text="Auto refresh (s)").pack(side=
      "left")
363 self.refresh_secs_var = ctk.IntVar(value=DEFAULTS["
      refresh_secs"])
364 self.refresh_secs = ctk.CTkEntry(row2, width=72,
      textvariable=self.refresh_secs_var)
365 self.refresh_secs.pack(side="right")
366
367 self.auto_write_var = ctk.BooleanVar(value=True)
368 self.chk_auto_write = ctk.CTkCheckBox(sb, text="Auto:
      Write      (Solve)      Reload      Push", variable=
      self.auto_write_var)
369 self.chk_auto_write.pack(padx=12, pady=(0, 6), anchor=
      "w")
370
371 self.btn_auto = ctk.CTkButton(sb, text="Start Auto",
      command=self._on_toggle_auto)
372 self.btn_auto.pack(padx=12, pady=(2, 10), fill="x")
373
374 # status
375 self.progress = ctk.CTkProgressBar(sb)
376 self.progress.set(0)
377 self.progress.pack(padx=12, pady=(4, 6), fill="x")
378 self.status = ctk.CTkLabel(sb, text="", anchor="w",
      justify="left", wraplength=340)
379 self.status.pack(padx=12, pady=(2, 8), fill="x")
380
381 def _build_main(self):
382     main = ctk.CTkFrame(self); main.pack(side="left", fill
      ="both", expand=True)

```

```

383     self.main = main
384
385     # KPI cards
386     cards = ctk.CTkFrame(main); cards.pack(fill="x", padx
        =12, pady=12)
387
388     self.latest_influx_var = ctk.StringVar(value="-")
389     self.written_var = ctk.StringVar(value="-")
390     self.out_cold_var = ctk.StringVar(value="-")
391     self.out_hot_in_var = ctk.StringVar(value="-")
392     self.out_hot_out_var = ctk.StringVar(value="-")
393
394     self._card(cards, "Influx Temp ( C )", self.
        latest_influx_var).pack(side="left", expand=True,
        fill="x", padx=6)
395     self._card(cards, "Inlet Written ( C )", self.
        written_var).pack(side="left", expand=True, fill="x
        ", padx=6)
396     self._card(cards, "Cold Water Outlet ( C )", self.
        out_cold_var).pack(side="left", expand=True, fill="
        x", padx=6)
397     self._card(cards, "Hot Water Inlet ( C )", self.
        out_hot_in_var).pack(side="left", expand=True, fill
        ="x", padx=6)
398     self._card(cards, "Hot Water Outlet ( C )", self.
        out_hot_out_var).pack(side="left", expand=True,
        fill="x", padx=6)
399
400     # Figure
401     self.fig = Figure(figsize=(8.3, 4.5), dpi=100)
402     self.ax = self.fig.add_subplot(111)
403     self.ax.set_title("Influx vs XML (after Solve)")
404     self.ax.set_xlabel("Sample #")
405     self.ax.set_ylabel(" C ")
406     self.canvas = FigureCanvasTkAgg(self.fig, master=main)
407     self.canvas.get_tk_widget().pack(fill="both", expand=
        True, padx=12, pady=(0, 12))
408
409     # console
410     self.console = ctk.CTkTextbox(main, height=220)

```

```

411         self.console.pack(fill="both", expand=False, padx=12,
412                             pady=(0, 12))
413
414         self._log("Ready.")
415
416     def _card(self, parent, title, var):
417         f = ctk.CTkFrame(parent)
418         ctk.CTkLabel(f, text=title, font=("Segoe UI", 12)).
419             pack(pady=(10, 0))
420         ctk.CTkLabel(f, textvariable=var, font=("Segoe UI",
421             24, "bold")).pack(pady=(4, 12))
422         return f
423
424     def _apply_defaults(self):
425         self.influx_url.insert(0, DEFAULTS["influx_url"])
426         self.influx_org.insert(0, DEFAULTS["influx_org"])
427         self.influx_bucket.insert(0, DEFAULTS["influx_bucket"
428             ])
429         self.measurement.insert(0, DEFAULTS["measurement"])
430         self.device_tag.insert(0, DEFAULTS["device_tag"])
431         self.inlet_key.insert(0, DEFAULTS["inlet_key"])
432         self.consolehost.insert(0, DEFAULTS["consolehost"])
433         self.out_measurement.insert(0, DEFAULTS["
434             out_measurement"])
435
436     # ----- helpers -----
437
438     def _browse_xml(self):
439         fn = filedialog.askopenfilename(
440             title="Pilih DWSIM XML",
441             filetypes=[("DWSIM XML", "*.xml"), ("All files", "
442                 *.*")]
443         )
444         if fn:
445             self.flowsheet.delete(0, "end")
446             self.flowsheet.insert(0, fn)
447
448     def _browse_consolehost(self):
449         fn = filedialog.askopenfilename(
450             title="Pilih DWSIM.ConsoleHost.exe",
451             filetypes=[("ConsoleHost", "*.exe"), ("All files",
452                 *.*")]

```

```

444         )
445         if fn:
446             self.consolehost.delete(0, "end")
447             self.consolehost.insert(0, fn)
448
449     def _set_status(self, text, p=None):
450         self.status.configure(text=text)
451         if p is not None:
452             try: self.progress.set(p)
453             except Exception: pass
454
455     def _log(self, text):
456         self.console.configure(state="normal")
457         self.console.insert("end", text + "\n")
458         self.console.see("end")
459         self.console.configure(state="disabled")
460
461     def _append_point(self, influx: Optional[float], written:
Optional[float],
462                       out_cold: Optional[float], out_hot_in:
Optional[float], out_hot_out:
Optional[float]):
463         def to_val(x): return float('nan') if x is None else
float(x)
464         self.series_influx.append(to_val(influx))
465         self.series_inlet_written.append(to_val(written))
466         self.series_out_cold.append(out_cold if out_cold is
not None else None)
467         self.series_out_hot_in.append(out_hot_in if out_hot_in
is not None else None)
468         self.series_out_hot_out.append(out_hot_out if
out_hot_out is not None else None)
469
470     def _plot(self):
471         self.ax.clear()
472         self.ax.set_title("Influx vs XML (after Solve)")
473         self.ax.set_xlabel("Sample #"); self.ax.set_ylabel("
C ")
474         n = len(self.series_influx)
475         x = list(range(1, n + 1))

```



```

476         if n > 0:
477             self.ax.plot(x, self.series_influx, label="Influx
478                 temperature")
479             self.ax.plot(x, self.series_inlet_written, label="
480                 Inlet written (XML)")
481             if any(v is not None for v in self.series_out_cold
482 ):
483                 y = [float('nan') if v is None else v for v in
484                     self.series_out_cold]
485                 self.ax.plot(x, y, label="Cold Water Outlet")
486             if any(v is not None for v in self.
487                 series_out_hot_in):
488                 y = [float('nan') if v is None else v for v in
489                     self.series_out_hot_in]
490                 self.ax.plot(x, y, label="Hot Water Inlet")
491             if any(v is not None for v in self.
492                 series_out_hot_out):
493                 y = [float('nan') if v is None else v for v in
494                     self.series_out_hot_out]
495                 self.ax.plot(x, y, label="Hot Water Outlet")
496             self.ax.legend()
497             self.canvas.draw_idle()
498
499 # ----- actions -----
500
501 def _on_health(self):
502     try:
503         text = influx_health(self.influx_url.get().strip()
504             )
505         self._log("[Health] " + text)
506         self._set_status("Influx /health OK", 1.0)
507     except Exception as e:
508         self._set_status(f"/health error: {e}")
509         self._log("ERROR: " + str(e))
510
511 def _on_list_streams(self):
512     p = self.flowsheet.get().strip()
513     if not Path(p).exists():
514         messagebox.showerror("Error", "Flowsheet XML tidak
515             ditemukan.")
516     return

```

```

506     try:
507         names = list_streams_display(p)
508         if names:
509             self._log("[Streams]\n- " + "\n- ".join(names)
510                        )
511             self._set_status(f"{len(names)} material
512                             streams ditemukan.")
513             self.stream_options.configure(values=names)
514             self.stream_options.set(names[0])
515         else:
516             self._log("[Streams] (kosong)")
517     except Exception as e:
518         self._set_status(f"Gagal list streams: {e}")
519         self._log("ERROR: " + str(e))
520
521 def _use_selected_stream(self):
522     val = self.stream_options.get().strip()
523     if val:
524         key = val.split(" (", 1)[0] if val.endswith("(")
525         and " (" in val else val
526         self.inlet_key.delete(0, "end")
527         self.inlet_key.insert(0, key)
528         self._log(f"[UI] Inlet set to: {key}")
529
530 def _on_test_influx(self):
531     try:
532         t = get_latest_temperature_c(
533             url=self.influx_url.get().strip(),
534             token=self.influx_token.get().strip(),
535             org=self.influx_org.get().strip(),
536             bucket=self.influx_bucket.get().strip(),
537             measurement=self.measurement.get().strip(),
538             device_tag=self.device_tag.get().strip(),
539             range_minutes=DEFAULTS["range_minutes"]
540         )
541         if t is None:
542             raise RuntimeError("Tidak ada sample
543                                temperature_c")
544         self.latest_influx_var.set(f"{t:.2f}")
545         self._log(f"[Influx] latest temperature_c = {t:.2f}

```

```

        } C ")
542         self._set_status("Influx OK", 1.0)
543     except Exception as e:
544         self._set_status(f"Influx error: {e}")
545         self._log("ERROR: " + "".join(traceback.
            format_exception_only(type(e), e)).strip())
546
547     def _write_solve_reload_once(self) -> Optional[float]:
548         """1 siklus: get influx      write XML      (optional)
            solve      reload outputs      push      plot"""
549         xml = self.flowsheet.get().strip()
550         if not Path(xml).exists():
551             raise FileNotFoundError("Flowsheet XML tidak
                ditemukan.")
552
553         # 1) ambil dari influx (inlet)
554         t = get_latest_temperature_c(
555             url=self.influx_url.get().strip(),
556             token=self.influx_token.get().strip(),
557             org=self.influx_org.get().strip(),
558             bucket=self.influx_bucket.get().strip(),
559             measurement=self.measurement.get().strip(),
560             device_tag=self.device_tag.get().strip(),
561             range_minutes=DEFAULTS["range_minutes"]
562         )
563         if t is None:
564             raise RuntimeError("Tidak ada sample temperature_c
                ")
565
566         # 2) write inlet ke XML
567         write_inlet_temperature(xml, self.inlet_key.get().
            strip() or DEFAULTS["inlet_key"], t)
568         self.latest_influx_var.set(f"{t:.2f}")
569         self.written_var.set(f"{t:.2f}")
570         self._log(f"[XML] Wrote inlet '{self.inlet_key.get().
            strip()}' = {t:.2f} C ")
571
572         # 3) (opsional) solve
573         if self.solve_enabled.get():
574             ch = self.consolehost.get().strip()

```

```

575         out = solve_with_consolehost(ch, xml, timeout_sec=
            DEFAULTS["solve_timeout_sec"])
576         self._log("[ConsoleHost] " + (out.strip() or "
            Solve done.))
577
578     # 4) reload outputs
579     outs = read_outputs(xml, list(DEFAULTS["out_keys"]))
580     c_out = outs.get("Cold Water Outlet")
581     h_in = outs.get("Hot Water Inlet")
582     h_out = outs.get("Hot Water Outlet")
583
584     self.out_cold_var.set(self._fmt(c_out))
585     self.out_hot_in_var.set(self._fmt(h_in))
586     self.out_hot_out_var.set(self._fmt(h_out))
587
588     # 5) push 4 params ke Influx (opsional)
589     if self.push_enabled.get():
590         n = write_four_params_to_influx_sync(
591             url=self.influx_url.get().strip(),
592             token=self.influx_token.get().strip(),
593             org=self.influx_org.get().strip(),
594             bucket=self.influx_bucket.get().strip(),
595             measurement=self.out_measurement.get().strip()
                    or DEFAULTS["out_measurement"],
596             device_tag=self.device_tag.get().strip(),
597             flowsheet_path=xml,
598             inlet_c=t,
599             cold_out_c=c_out,
600             hot_in_c=h_in,
601             hot_out_c=h_out,
602         )
603         self._log(f"[Influx] pushed {n} point(s) to
            measurement '{self.out_measurement.get().strip()
            or DEFAULTS['out_measurement']}'")
604
605     # 6) append point & plot
606     self._append_point(t, t, c_out, h_in, h_out)
607     self._plot()
608     return t
609

```

```

610 def _on_write_solve_reload(self):
611     try:
612         self._write_solve_reload_once()
613         self._set_status("Write      (Solve)      Reload
        Push OK", 1.0)
614     except Exception as e:
615         self._set_status(f"Error: {e}")
616         self._log("ERROR: " + "".join(traceback.
            format_exception_only(type(e), e)).strip())
617
618 def _on_toggle_auto(self):
619     if self._auto:
620         self._auto = False
621         self.btn_auto.configure(text="Start Auto")
622         self._set_status("Auto stopped")
623         return
624
625     try:
626         secs = max(2, int(self.refresh_secs_var.get()))
627     except Exception:
628         secs = DEFAULTS["refresh_secs"]
629     self._auto = True
630     self.btn_auto.configure(text="Stop Auto")
631     self._set_status(f"Auto started ({secs}s)")
632
633 def loop():
634     while self._auto:
635         start = time.time()
636         try:
637             if self.auto_write_var.get():
638                 self._write_solve_reload_once()
639             else:
640                 t = get_latest_temperature_c(
641                     url=self.influx_url.get().strip(),
642                     token=self.influx_token.get().
                        strip(),
643                     org=self.influx_org.get().strip(),
644                     bucket=self.influx_bucket.get().
                        strip(),
645                     measurement=self.measurement.get()

```

```

        .strip(),
        device_tag=self.device_tag.get().
        strip(),
        range_minutes=DEFAULTS["
            range_minutes"]
    )
    self.latest_influx_var.set(self._fmt(t
    ))
    self._append_point(t, None, None, None
        , None)
    self._plot()
    self._set_status("Auto tick OK")
except Exception as e:
    self._set_status(f"Auto error: {e}")
    self._log("AUTO ERROR: " + "".join(
        traceback.format_exception_only(type(e)
        , e)).strip())

try:
    secs = max(2, int(self.refresh_secs_var.
        get()))
except Exception:
    secs = DEFAULTS["refresh_secs"]
time.sleep(max(0, secs - (time.time() - start)
    ))

threading.Thread(target=loop, daemon=True).start()

# utils
def _fmt(self, v):
    try:
        return f"{float(v):.2f}"
    except Exception:
        return "-"

if __name__ == "__main__":
    app = App()
    app.mainloop()

```