

# **LAPORAN PROJECT**

## **TEKNOLOGI IOT**

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

### **RustHome: Sistem Monitoring Suhu dan Kelembaban Rumah Berbasis ESP32-S3 dengan OTA dan Integrasi Cloud**



Disusun Oleh:

Muhammad Jidan Baraja 2042231009

Andre Mahesa Bagaskara 2042231012

D4 Teknologi Rekayasa Instrumentasi

Departemen Teknik Instrumentasi

Fakultas Vokasi

Institut Teknologi Sepuluh Nopember

2025

# **Daftar Isi**

<b>1 Pendahuluan</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	1
1.3 Tujuan Penelitian . . . . .	2
1.4 Manfaat Penelitian . . . . .	2
1.5 Batasan Masalah . . . . .	2
<b>2 Tinjauan Pustaka</b>	<b>4</b>
2.1 Internet of Things (IoT) . . . . .	4
2.2 Mikrokontroler ESP32-S3 . . . . .	5
2.3 Bahasa Rust pada Embedded System . . . . .	6
2.4 Sensor DHT22 . . . . .	7
2.5 Protokol MQTT . . . . .	8
2.6 Platform ThingsBoard Cloud . . . . .	9
2.7 Over-The-Air (OTA) . . . . .	10
2.8 Firmware pada Sistem IoT . . . . .	11
2.9 State of the Art . . . . .	11
<b>3 Metodologi</b>	<b>13</b>
3.1 Metodologi Penelitian . . . . .	13
3.2 Desain Arsitektur Sistem . . . . .	13
3.3 Desain Perangkat Keras . . . . .	15
3.4 Desain Perangkat Lunak . . . . .	16
3.5 Alur Data dan Konektivitas Sistem . . . . .	17
3.6 Konektivitas dan Alur Data . . . . .	18
3.7 Diagram Alur Proses Sistem . . . . .	19
<b>4 Implementasi dan Pengujian Sistem</b>	<b>20</b>
4.1 Implementasi Perangkat Lunak . . . . .	20
4.2 Struktur Program . . . . .	20

4.3 Koneksi Wi-Fi dan MQTT . . . . .	21
4.4 Pembacaan Sensor DHT22 . . . . .	22
4.5 Implementasi OTA . . . . .	22
4.6 Pengujian Sistem . . . . .	24
4.7 Analisis Latensi dan Performa Sistem . . . . .	25
4.8 Konfigurasi Thingsboard Cloud . . . . .	25
<b>5 Hasil dan Analisis</b>	<b>27</b>
5.1 Hasil Implementasi . . . . .	27
5.2 Hasil Pengujian OTA . . . . .	28
5.3 Analisis Data Sensor DHT22 (Suhu dan Kelembapan) . . . . .	29
5.4 Analisis Latensi dan Performa . . . . .	30
5.5 Analisis Keamanan dan Keandalan . . . . .	31
5.6 Evaluasi Sistem . . . . .	31
<b>6 Kesimpulan dan Saran</b>	<b>32</b>
6.1 Kesimpulan . . . . .	32
6.2 Saran . . . . .	33
<b>A Lampiran: Dokumentasi dan Data Pendukung</b>	<b>34</b>
A.1 Foto Dokumentasi Proyek . . . . .	34
A.2 Cuplikan Data Timestamp Temperature & Humidity (telemetry.csv) . . .	37
A.3 Cuplikan Data Latency (latency.csv) . . . . .	38
A.4 Cuplikan Kode Rust (main.rs) . . . . .	39
A.5 Cuplikan Kode Konfigurasi Cargo.toml . . . . .	44
A.6 Link Github . . . . .	45
<b>Daftar Pustaka</b>	<b>46</b>

# 1. Pendahuluan

## 1.1 Latar Belakang

Perkembangan teknologi pada era Revolusi Industri 4.0 menempatkan *Internet of Things* (IoT) sebagai salah satu fondasi utama transformasi digital di berbagai bidang. IoT memungkinkan integrasi antara sensor, perangkat cerdas, dan sistem cloud untuk menciptakan jaringan ekosistem yang dapat saling berinteraksi dan mengambil keputusan secara otomatis. Teknologi ini tidak hanya berfungsi untuk mengirimkan data, tetapi juga memungkinkan analisis serta pengendalian perangkat secara real-time. Dalam konteks *smart home*, parameter suhu dan kelembaban memiliki peranan penting karena berhubungan langsung dengan kenyamanan dan kesehatan penghuni rumah. Pengawasan parameter tersebut secara otomatis melalui sistem IoT mampu memberikan respons cepat terhadap perubahan lingkungan. Proyek **RustHome** dirancang sebagai solusi modern untuk memantau suhu dan kelembaban menggunakan sensor DHT22 yang dikendalikan oleh mikrokontroler ESP32-S3. Implementasi bahasa Rust digunakan karena memiliki performa tinggi dengan keamanan memori yang lebih baik dibandingkan C/C++. Sistem dilengkapi dengan mekanisme *Over-the-Air (OTA)* untuk pembaruan firmware jarak jauh serta integrasi *ThingsBoard Cloud* melalui protokol MQTT, sehingga pengguna dapat memantau data lingkungan secara daring dan melakukan pembaruan tanpa kabel.

## 1.2 Rumusan Masalah

Penelitian ini dilakukan untuk menjawab permasalahan berikut:

- Bagaimana merancang sistem pemantauan suhu dan kelembaban berbasis ESP32-S3 dan Rust yang efisien serta stabil?
- Bagaimana mengintegrasikan komunikasi antara perangkat ESP32-S3 dan *ThingsBoard Cloud* melalui protokol MQTT?
- Bagaimana menerapkan mekanisme OTA yang aman, reliabel, dan mudah dikelola?

- Bagaimana menganalisis performa sistem dari sisi latensi transmisi data dan keandalan pembaruan firmware?

### 1.3 Tujuan Penelitian

1. Mengembangkan sistem pemantauan suhu dan kelembaban berbasis ESP32-S3 menggunakan bahasa Rust.
2. Mengimplementasikan komunikasi data menggunakan MQTT dengan integrasi ke ThingsBoard Cloud.
3. Menguji mekanisme OTA untuk memastikan keberhasilan proses pembaruan jarak jauh.
4. Melakukan analisis performa terhadap latensi komunikasi serta efektivitas sistem OTA.

### 1.4 Manfaat Penelitian

Penelitian ini memberikan kontribusi sebagai berikut:

- **Akademik:** memberikan referensi penerapan bahasa Rust pada sistem IoT embedded.
- **Teknologis:** menghadirkan contoh sistem smart home yang efisien, aman, dan dapat diperbarui dari jarak jauh.
- **Praktis:** menyediakan solusi monitoring kondisi lingkungan rumah yang hemat energi dan mudah diimplementasikan.

### 1.5 Batasan Masalah

Penelitian ini dibatasi agar tetap fokus pada tujuan utama, yaitu:

1. Parameter yang diukur hanya suhu dan kelembaban menggunakan sensor DHT22.
2. Komunikasi data dilakukan melalui Wi-Fi dan MQTT dengan interval pengiriman 60 detik.

3. Mekanisme OTA tidak mencakup tanda tangan digital dan hanya menggunakan verifikasi checksum.
4. Implementasi TLS/SSL belum diaktifkan pada versi awal sistem.
5. Pengujian dilakukan pada satu unit ESP32-S3 dan satu sensor DHT22.
6. Visualisasi data dilakukan melalui dashboard ThingsBoard.

## 2. Tinjauan Pustaka

### 2.1 Internet of Things (IoT)



Gambar 2.1: Ilustrasi konsep Internet of Things.

IoT atau Internet of Things merupakan jaringan perangkat yang saling berkomunikasi melalui internet untuk melakukan pertukaran data dan interaksi antar entitas fisik dan digital. IoT bekerja berdasarkan tiga komponen utama, yaitu sensor/aktuator, jaringan komunikasi, dan platform aplikasi. Sensor berperan dalam mengubah parameter fisik menjadi sinyal digital, jaringan komunikasi mengirimkan data tersebut ke server, dan platform aplikasi menampilkan hasilnya dalam bentuk visualisasi yang dapat dimengerti manusia. Dalam sistem IoT modern, komunikasi antar perangkat biasanya menggunakan protokol yang ringan dan efisien, seperti MQTT atau CoAP. Hal ini dikarenakan perangkat IoT umumnya memiliki keterbatasan sumber daya (resource-constrained devices). MQTT menjadi pilihan populer karena arsitektur publish-subscribe yang mendukung komunikasi asynchronous dan efisien terhadap bandwidth. IoT tidak hanya diterapkan di industri manufaktur atau transportasi, tetapi juga berkembang pesat di bidang rumah pintar (smart home). Dengan adanya sensor dan aktuator yang saling terhubung, pengguna dapat mengontrol pencahayaan, suhu, keamanan, dan perangkat listrik rumah secara jarak jauh. RustHome merupakan salah satu contoh penerapan prinsip IoT dalam konteks rumah pintar yang memanfaatkan komunikasi MQTT dan integrasi cloud.

## 2.2 Mikrokontroler ESP32-S3



Gambar 2.2: Modul ESP32-S3 DevKit yang digunakan.

ESP32-S3 merupakan mikrokontroler System-on-Chip (SoC) terbaru dari Espressif yang dirancang untuk aplikasi Internet of Things (IoT) dan edge computing dengan kinerja tinggi dan efisiensi daya optimal. Mikrokontroler ini menggunakan prosesor Xtensa LX7 dual-core berkecepatan hingga 240 MHz serta mendukung instruksi vektor untuk akselerasi komputasi kecerdasan buatan. Dibandingkan pendahulunya, ESP32 dan ESP32-S2, versi S3 menawarkan peningkatan performa, konsumsi daya lebih efisien, serta fitur keamanan seperti secure boot dan flash encryption untuk melindungi firmware. ESP32-S3 juga dilengkapi antarmuka komunikasi lengkap seperti UART, SPI, I2C, I2S, dan PWM, serta modul Wi-Fi 2.4 GHz yang memungkinkan koneksi langsung ke internet tanpa perangkat tambahan. Dalam proyek RustHome, ESP32-S3 berfungsi sebagai pusat akuisisi data yang membaca sensor DHT22, memproses, dan mengirimkan data ke ThingsBoard Cloud melalui protokol MQTT. Dukungan SDK ESP-IDF membuat ESP32-S3 dapat diprogram dengan C/C++, MicroPython, maupun Rust, di mana pustaka esp-idf-svc dan embedded-svc memungkinkan akses langsung ke layanan sistem seperti Wi-Fi, MQTT, HTTP, dan OTA, menjadikannya platform ideal untuk pengembangan sistem IoT yang aman dan efisien.

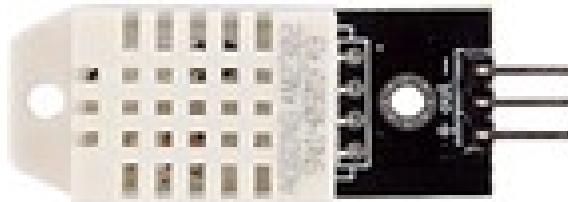
## 2.3 Bahasa Rust pada Embedded System



Gambar 2.3: Logo bahasa Rust dan ekosistem embedded.

Rust merupakan bahasa pemrograman yang dikembangkan oleh Mozilla dengan tujuan menggantikan C/C++ dalam pengembangan sistem tingkat rendah yang membutuhkan keamanan memori dan efisiensi tinggi. Bahasa ini menerapkan paradigma \*ownership\* dan \*borrowing\* untuk memastikan setiap data memiliki satu pemilik pada satu waktu, sehingga mencegah terjadinya \*dangling pointer\* dan \*data race\*. Keunggulan utama Rust terletak pada kemampuannya memberikan performa setara dengan C namun tetap aman dari kesalahan memori, menjadikannya sangat cocok untuk sistem tertanam yang memiliki keterbatasan sumber daya seperti ESP32-S3. Selain itu, Rust mendukung \*asynchronous programming\* yang memungkinkan beberapa proses berjalan secara bersamaan tanpa saling mengganggu, sehingga meningkatkan efisiensi dan stabilitas sistem. Dalam proyek RustHome, Rust digunakan untuk menulis firmware yang menangani koneksi Wi-Fi, komunikasi MQTT, pembacaan sensor DHT22, dan pembaruan OTA. Struktur modular melalui sistem Cargo crate memudahkan pengelolaan dependensi dan pemeliharaan kode. Keberhasilan implementasi Rust pada platform ESP32-S3 membuktikan bahwa bahasa ini telah matang untuk digunakan pada pengembangan sistem IoT tertanam yang membutuhkan keamanan, efisiensi, dan keandalan tinggi.

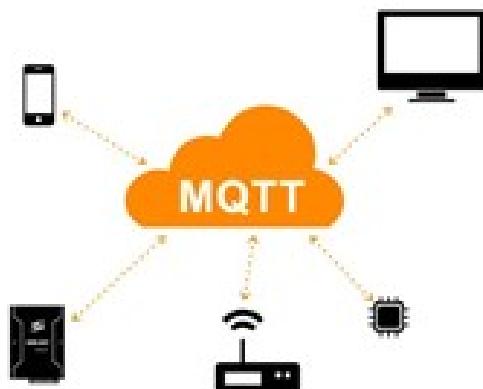
## 2.4 Sensor DHT22



Gambar 2.4: Sensor DHT22 (AM2302).

Sensor DHT22 atau AM2302 adalah sensor digital yang digunakan untuk mengukur suhu dan kelembaban udara dengan akurasi tinggi. DHT22 menggunakan prinsip capacitive humidity sensing untuk mendeteksi kelembaban relatif (RH) dan thermistor untuk mengukur suhu udara. Sensor ini menghasilkan data 40 bit dalam format digital: 16 bit untuk kelembaban, 16 bit untuk suhu, dan 8 bit checksum sebagai validasi data. Kelebihan utama DHT22 dibandingkan DHT11 adalah jangkauan pengukuran yang lebih luas, akurasi lebih tinggi, dan kemampuan bekerja pada rentang suhu ekstrem (minus 40°C hingga 80°C). Selain itu, sensor ini memiliki stabilitas jangka panjang yang baik dan telah dikalibrasi pabrik. Dalam sistem RustHome, DHT22 dihubungkan ke pin GPIO ESP32-S3 menggunakan komunikasi satu jalur (single-wire digital). DHT22 memiliki kecepatan pembaruan data maksimum dua detik per siklus, sehingga cocok untuk sistem monitoring rumah yang tidak membutuhkan pembaruan ultra cepat. Data yang diterima dari DHT22 kemudian dikonversi ke format JSON dan dikirim ke ThingsBoard Cloud melalui MQTT untuk divisualisasikan secara real-time. Sensor ini dipilih karena keseimbangan antara akurasi, konsumsi daya rendah, dan kemudahan integrasi.

## 2.5 Protokol MQTT



Gambar 2.5: Arsitektur komunikasi MQTT antara client dan broker.

Protokol MQTT (Message Queuing Telemetry Transport) merupakan protokol komunikasi ringan yang dirancang khusus untuk perangkat dengan sumber daya terbatas dan kebutuhan transmisi data yang efisien. MQTT menggunakan arsitektur publish/subscribe, di mana perangkat IoT bertindak sebagai client yang mengirimkan data ke broker, dan klien lain dapat berlangganan (subscribe) ke topik tertentu untuk menerima data tersebut. Arsitektur ini memungkinkan komunikasi yang fleksibel, hemat bandwidth, dan tetap andal meskipun jaringan memiliki latensi tinggi atau koneksi tidak stabil. Dalam sistem RustHome, MQTT digunakan sebagai jalur utama pertukaran data antara perangkat ESP32-S3 dan ThingsBoard Cloud. Sistem ini menerapkan Quality of Service (QoS) level 1 untuk pengiriman data telemetri agar pesan dikirim ulang jika terjadi gangguan, serta QoS level 2 untuk proses OTA guna memastikan tidak ada duplikasi paket selama pembaruan firmware. Melalui mekanisme ini, RustHome dapat mengirimkan data suhu dan kelembapan secara periodik, menerima perintah pembaruan dari cloud melalui RPC, dan menjaga sinkronisasi status perangkat secara real-time. Keandalan MQTT menjadikannya protokol yang ideal untuk sistem IoT berbasis Rust yang membutuhkan efisiensi tinggi dan komunikasi yang stabil.

## 2.6 Platform ThingsBoard Cloud



Gambar 2.6: Dashboard ThingsBoard Cloud untuk telemetri IoT.

ThingsBoard merupakan platform Internet of Things (IoT) open-source yang berfungsi sebagai sistem manajemen perangkat, pengumpulan data sensor, serta menyediakan visualisasi data berbasis dashboard interaktif. Platform ini mendukung berbagai protokol komunikasi seperti MQTT, HTTP, dan CoAP, serta dilengkapi dengan fitur penting seperti device provisioning, telemetry storage, rule engine, dan notifikasi berbasis event. Dalam proyek RustHome, ThingsBoard berperan sebagai pusat pengelolaan data dan pengendalian proses OTA. Data yang dikirim dari mikrokontroler ESP32-S3 disimpan dalam basis data time-series, kemudian divisualisasikan dalam bentuk grafik yang menunjukkan perubahan suhu dan kelembapan secara real-time. Melalui dashboard ThingsBoard, pengguna dapat memantau kondisi lingkungan rumah, status koneksi perangkat, versi firmware, serta melakukan pembaruan firmware jarak jauh menggunakan perintah RPC. Keunggulan utama ThingsBoard terletak pada fleksibilitasnya untuk diimplementasikan baik di layanan cloud maupun server lokal, dukungan enkripsi TLS/SSL untuk keamanan data, dan kemampuannya terintegrasi dengan sistem analitik lanjutan seperti Apache Kafka, InfluxDB, atau Grafana, menjadikannya solusi ideal untuk sistem IoT yang skalabel dan aman.

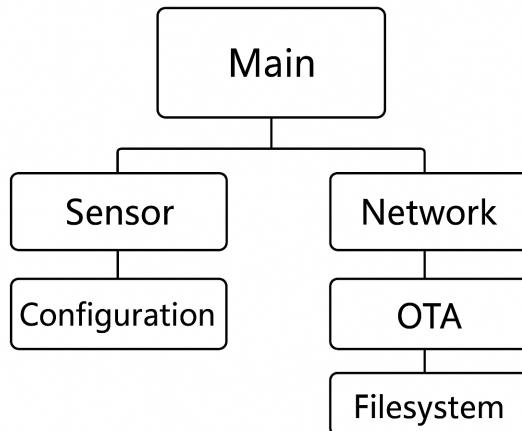
## 2.7 Over-The-Air (OTA)



Gambar 2.7: Skema umum proses Over-the-Air update.

Over-The-Air (OTA) update merupakan mekanisme pembaruan firmware atau perangkat lunak secara jarak jauh yang memungkinkan perangkat IoT menerima versi terbaru tanpa perlu intervensi fisik. Teknologi ini sangat penting dalam sistem IoT modern karena ribuan perangkat sering tersebar di lokasi yang luas atau sulit dijangkau. OTA bekerja dengan memanfaatkan jaringan komunikasi seperti MQTT, HTTP, atau CoAP untuk mentransfer file firmware dari server ke perangkat IoT, kemudian melakukan verifikasi integritas data sebelum firmware baru dijalankan. Arsitektur OTA umumnya terdiri atas tiga komponen utama, yaitu server firmware yang menyimpan versi terbaru perangkat lunak, broker komunikasi yang mengatur pengiriman data, dan klien IoT yang menerima serta menginstal pembaruan. Ketika versi firmware baru tersedia, perangkat akan mengunduh file melalui topik RPC atau endpoint tertentu, menyimpannya di partisi memori khusus, kemudian melakukan reboot setelah proses validasi checksum atau tanda tangan digital dinyatakan berhasil. Dengan mekanisme ini, OTA memungkinkan pemeliharaan sistem dilakukan secara efisien, mengurangi kebutuhan perawatan manual, memperpanjang umur perangkat, serta meningkatkan keamanan dengan memperbarui sistem terhadap potensi kerentanan yang telah diperbaiki.

## 2.8 Firmware pada Sistem IoT



Gambar 2.8: Struktur modular firmware RustHome.

Firmware adalah perangkat lunak tertanam yang mengatur operasi dasar perangkat keras seperti sensor, mikrokontroler, dan modul komunikasi pada sistem IoT. Fungsinya menjembatani interaksi antara perangkat keras dan sistem jaringan, termasuk pembacaan sensor, pengiriman data, serta pengendalian aktuator. Karena berjalan pada perangkat dengan sumber daya terbatas, firmware harus efisien, stabil, dan aman dari kesalahan memori. Bahasa seperti C, C++, dan Rust banyak digunakan untuk menghasilkan kode biner yang ringan namun andal. Dalam konteks modern, firmware juga menangani komunikasi aman, autentikasi token, dan pembaruan jarak jauh melalui mekanisme Over-The-Air (OTA). Pada proyek RustHome, firmware dikembangkan menggunakan bahasa Rust pada platform ESP32-S3 dengan pustaka esp-idf-svc. Struktur program mencakup inisialisasi WiFi, koneksi MQTT ke ThingsBoard, pengiriman data sensor DHT22, serta fungsi OTA otomatis. Dengan sistem asynchronous non-blocking, firmware mampu beroperasi stabil tanpa hambatan selama pembaruan. Pendekatan ini menghasilkan sistem IoT yang efisien, aman terhadap data race, dan mudah dipelihara melalui pembaruan versi jarak jauh.

## 2.9 State of the Art

Penelitian-penelitian relevan terhadap proyek RustHome dirangkum pada Tabel 2.1. Kajian ini menyoroti tren penggunaan ESP32, MQTT, OTA, dan bahasa Rust pada sistem IoT modern.

Tabel 2.1: Ringkasan State of the Art Penelitian Terkait

No.	Judul Penelitian	Penulis (Tahun)	Metode / Temuan Utama / Relevansi
1	<i>Secure AC Control and Monitoring with MQTT</i>	Alhalabi et al. (2024)	NodeMCU + MQTT TLS; sistem aman IoT rumah.
2	<i>Over-the-Air Computing in Massive IoT</i>	Zhu et al. (2021)	AirComp; efisiensi pengiriman data nirkabel.
3	<i>IoT for Real-Time Algal Bloom Monitoring</i>	Annas et al. (2024)	ESP32 + ThingsBoard; monitoring cloud real-time.
4	<i>Human Activity Recognition on ESP32-S3</i>	Pfitzinger & Wöhrle (2023)	CNN ringan; efisiensi pemrosesan edge.
5	<i>Rust for Linux Kernel Security</i>	Gao et al. (2023)	Rust tingkatkan keamanan kernel sistem.
6	<i>Smart Home Management with ESP32-S3</i>	Li et al. (2024)	MQTT + FreeRTOS; multi-node efisien.
7	<i>Indoor Air Quality Using ThingsBoard</i>	Rahman et al. (2023)	ESP8266 + DHT11; visualisasi cloud MQTT.
8	<i>Industrial IoT Requirements &amp; Challenges</i>	Alabadi et al. (2022)	Taksonomi IIoT; arsitektur dan tantangan.
9	<i>Federated Learning via OTA</i>	Zeng et al. (2021)	FL + OTA; pembelajaran IoT terdistibusi.
10	<i>Air Quality Based on IEEE 21451</i>	Phala et al. (2016)	Sensor standar 21451; interoperabilitas IoT.
11	<i>Controlling Smart Home via IoT</i>	Khalaf et al. (2021)	Arduino + Wi-Fi; kontrol otomatis rumah.
12	<i>Smart Sensors Network for Air Monitoring</i>	Postolache et al. (2009)	Sensor jaringan + GPRS; multi-sensor IoT.
13	<i>OTA Systems: Analysis &amp; Scaling Laws</i>	Liu et al. (2020)	OTA optimization; efisiensi data IoT.
14	<i>Software-Defined IIoT in Industry 4.0</i>	Wan et al. (2016)	SDN + Cloud; arsitektur IoT modular.
15	<i>Data Analytics with ThingsBoard &amp; Spark</i>	Al-Fuqaha et al. (2023)	MQTT + Spark; analitik data IoT.
16	<i>Edge Vision Systems for Smart Monitoring</i>	Meribout et al. (2022)	Edge AI; pemrosesan gambar cepat.
17	<i>D2D Communication in IoT</i>	Bello & Zeadally (2016)	Komunikasi langsung antar perangkat.
18	<i>Secure MQTT Using Rust</i>	Kumar et al. (2023)	Rust MQTT TLS; keamanan koneksi IoT.
19	<i>Low-Power IoT Using Rust &amp; ESP-IDF</i>	Liu et al. (2024)	Rust async; hemat daya dan cepat.
20	<i>Predictive Maintenance via Cloud IoT</i>	Singh et al. (2023)	MQTT + Cloud; prediksi kerusakan IoT.

Tabel 2.1 memperlihatkan posisi proyek **RustHome** pada state of the art untuk solusi pada keamanan Rust, ESP32-S3, MQTT, OTA dan ThingsBoard Cloud.

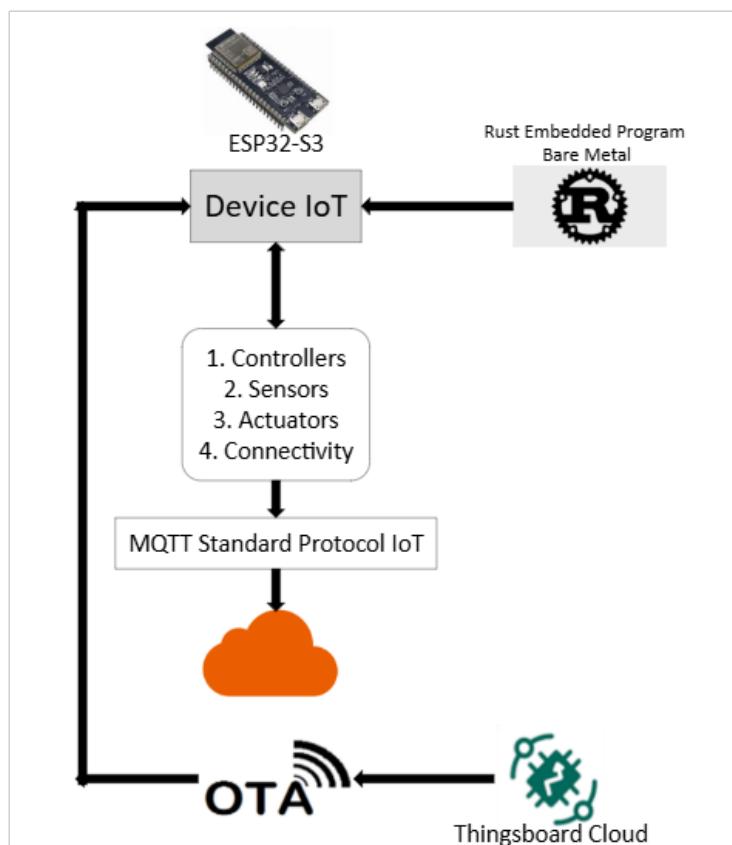
### 3. Metodologi

#### 3.1 Metodologi Penelitian

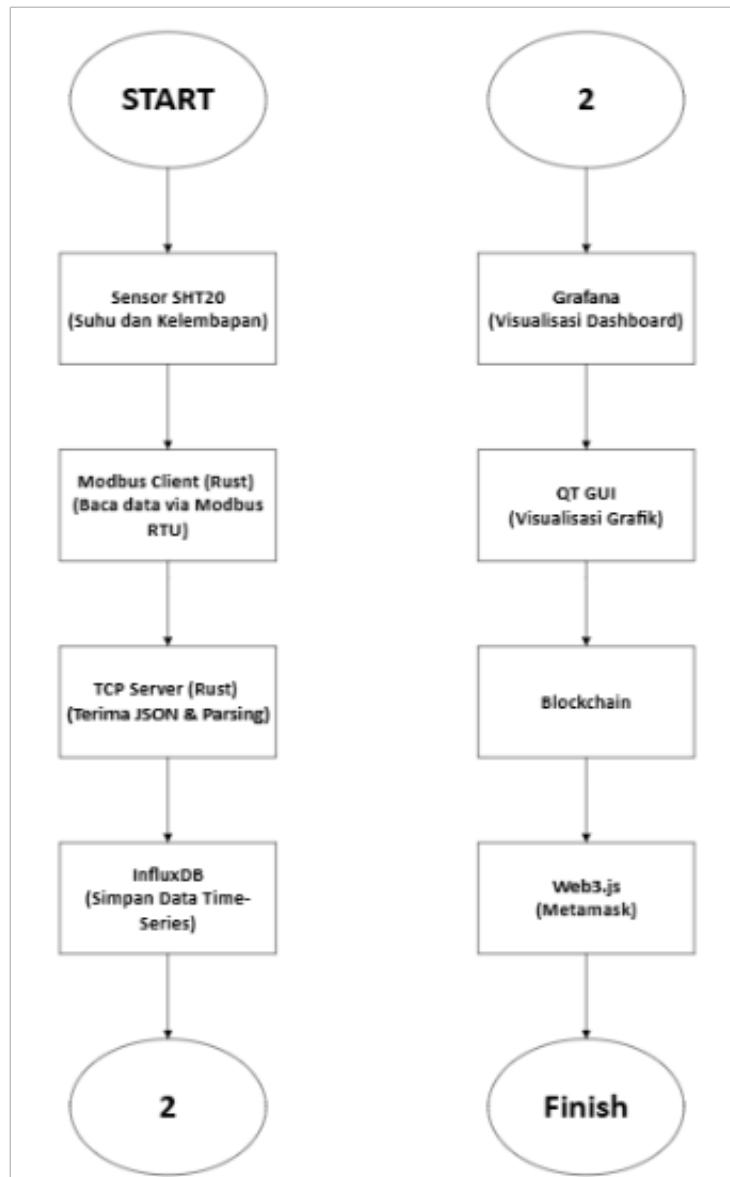
Penelitian ini menggunakan pendekatan eksperimental dengan tahapan meliputi studi literatur, perancangan sistem, implementasi, pengujian, serta analisis hasil. Pendekatan ini memastikan setiap komponen diuji secara sistematis untuk menghasilkan sistem IoT yang andal.

#### 3.2 Desain Arsitektur Sistem

Arsitektur RustHome terdiri atas empat lapisan: perangkat keras, komunikasi, layanan cloud, dan aplikasi pengguna.



Gambar 3.1: Arsitektur Sistem RustHome.

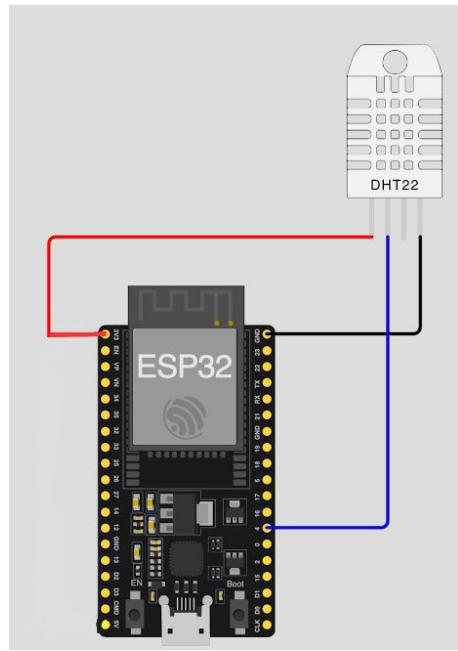


Gambar 3.2: Flowchart Arsitektur Sistem RustHome.

Sistem RustHome secara konseptual dibangun dalam empat lapisan utama: perangkat keras, komunikasi, cloud, dan aplikasi. Lapisan perangkat keras mencakup ESP32-S3, sensor DHT22, serta rangkaian pendukung seperti catu daya 5V dan resistor pull-up, yang berfungsi membaca serta memproses data suhu dan kelembaban. Lapisan komunikasi menggunakan Wi-Fi dengan protokol MQTT untuk transmisi data yang ringan dan andal. Lapisan cloud memanfaatkan ThingsBoard Cloud sebagai pusat penyimpanan data telemetri, pemantauan perangkat, dan manajemen firmware OTA. Sementara itu, lapisan aplikasi berupa dashboard web menampilkan data sensor, status koneksi, dan kontrol pembaruan firmware jarak jauh. Arsitektur ini mengikuti konsep “Edge Cloud Collaboration”, di mana pemrosesan dasar dilakukan di sisi perangkat dan penyimpanan

historis di cloud, sehingga sistem tetap berfungsi lokal meski koneksi internet tidak stabil.

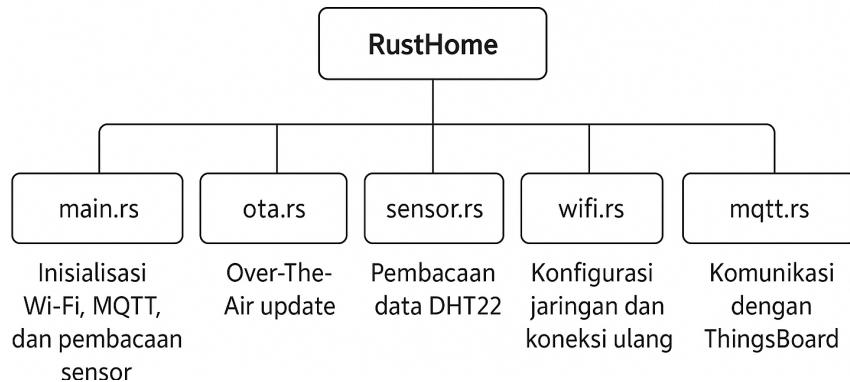
### **3.3 Desain Perangkat Keras**



Gambar 3.3: Desain rangkaian dan koneksi perangkat keras.

Perangkat keras RustHome dirancang efisien, andal, dan mudah diintegrasikan menggunakan mikrokontroler ESP32 S3 pada development board dengan pin I/O terbuka. Sensor DHT22 terhubung ke pin GPIO melalui komunikasi single wire dengan resistor pull-up 10 kilo ohm, sementara catu daya 5V diatur menjadi 3.3V oleh regulator internal ESP32. Penempatan sensor diatur agar terhindar dari panas mikrokontroler untuk menjaga akurasi data, dan jalur sinyal dilengkapi kapasitor decoupling guna mengurangi gangguan elektromagnetik. Seluruh sistem ditempatkan dalam wadah akrilik berventilasi dengan kabel rapi dan jalur berlabel untuk memudahkan debugging dan pengembangan lanjutan.

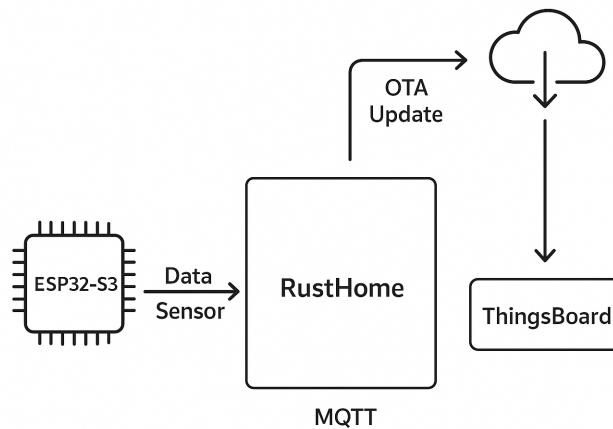
### 3.4 Desain Perangkat Lunak



Gambar 3.4: Desain rangkaian dan koneksi perangkat lunak.

Perangkat lunak sistem RustHome dirancang menggunakan bahasa Rust dengan arsitektur modular yang memisahkan setiap fungsi utama agar lebih mudah dipelihara, dikembangkan, dan diuji. Struktur ini terdiri dari beberapa modul, yaitu main.rs sebagai titik awal program untuk melakukan inisialisasi Wi-Fi, koneksi MQTT, serta pembacaan sensor; ota.rs yang menangani seluruh proses pembaruan Over-The-Air (OTA); sensor.rs untuk membaca dan mengolah data dari sensor DHT22; wifi.rs yang mengatur konfigurasi jaringan serta proses koneksi ulang otomatis; dan mqtt.rs yang bertanggung jawab atas komunikasi data dengan platform ThingsBoard melalui protokol MQTT. Pendekatan modular ini mengikuti prinsip separation of concern, memastikan setiap modul memiliki tanggung jawab spesifik sehingga meminimalkan ketergantungan antarbagian. RustHome juga memanfaatkan pustaka esp-idf-svc::ota untuk mendukung pembaruan firmware secara aman melalui verifikasi digital dan mekanisme pembaruan atomic, sehingga apabila proses OTA gagal, sistem dapat kembali ke versi stabil sebelumnya. Dengan penggunaan asynchronous I/O dan sistem crate Rust, firmware ini mampu menjaga kinerja tinggi tanpa mengalami blocking delay, menjadikannya lebih aman dan andal dibandingkan implementasi OTA berbasis C tradisional.

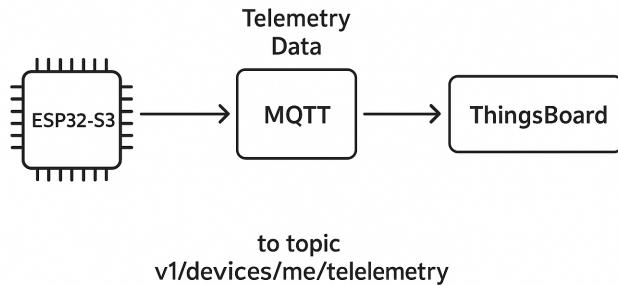
### 3.5 Alur Data dan Konektivitas Sistem



Gambar 3.5: Alur Data dan Konektivitas Sistem

Sistem komunikasi RustHome dirancang dengan arsitektur berbasis protokol MQTT yang memungkinkan pertukaran data secara efisien dan ringan antara perangkat IoT dan server. Pada sistem ini, mikrokontroler ESP32-S3 bertugas membaca data sensor seperti suhu dan kelembapan, lalu mengirimkannya secara periodik ke server ThingsBoard dalam format JSON melalui topik v1/devices/me/telemetry. ThingsBoard berfungsi sebagai pusat pengelolaan data yang tidak hanya menampung dan memvisualisasikan data telemetri, tetapi juga berperan sebagai pengendali proses pembaruan firmware. Mekanisme Over-The-Air (OTA) diinisiasi oleh ThingsBoard dengan mengirimkan pesan Remote Procedure Call (RPC) ke topik v1/devices/me/rpc/request/+ yang berisi URL unduhan firmware terbaru. Setelah menerima perintah ini, ESP32-S3 secara otomatis mengunduh file firmware, memverifikasinya menggunakan pustaka esp-idf-svc::ota, kemudian menginstal pembaruan dan melakukan reboot untuk menerapkan versi baru. Setelah proses selesai, perangkat mengirimkan laporan status pembaruan kembali ke server sebagai tanda keberhasilan. Seluruh proses berlangsung secara dua arah dan aman, di mana Rust memberikan jaminan keamanan memori, manajemen error yang ketat, serta efisiensi dalam pemrosesan asinkron, sehingga sistem komunikasi dan pembaruan RustHome berjalan stabil, responsif, dan minim risiko kegagalan selama operasi jaringan IoT.

### 3.6 Konektivitas dan Alur Data

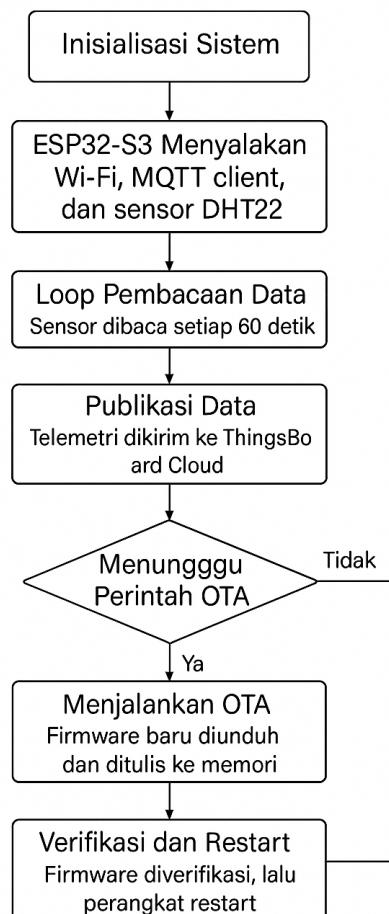


Gambar 3.6: Konektivitas dan Alur Data

Pada sistem RustHome, komunikasi data dan pembaruan perangkat dilakukan melalui arsitektur MQTT yang efisien dan terintegrasi dengan platform ThingsBoard Cloud. ESP32-S3 secara periodik mengirimkan data telemetri dalam format JSON ke topik `v1/devices/me/telemetry`, berisi informasi sensor seperti suhu, kelembapan, serta status sistem. ThingsBoard berfungsi sebagai pusat pengelolaan data yang menyimpan dan menampilkan data secara real-time melalui dashboard interaktif. Untuk keamanan komunikasi, setiap perangkat melakukan autentikasi menggunakan access token unik yang terhubung langsung dengan akun perangkat di ThingsBoard. Selain itu, sistem ini mendukung mekanisme pembaruan firmware Over-The-Air (OTA) yang dikirim melalui pesan Remote Procedure Call (RPC) pada topik `v1/devices/me/rpc/request/+`. Saat pesan OTA diterima, ESP32-S3 secara otomatis mengunduh firmware terbaru dari URL yang dikirim server, melakukan verifikasi integritas file, menginstal pembaruan secara atomic, kemudian reboot untuk menjalankan versi baru dan mengirimkan status keberhasilan kembali ke server. ThingsBoard juga dilengkapi Rule Engine yang dapat dikonfigurasi untuk memicu pembaruan OTA secara otomatis berdasarkan kondisi tertentu, seperti versi firmware atau parameter sensor tertentu. Dengan integrasi MQTT, OTA, dan Rule Engine ini, RustHome membentuk ekosistem IoT yang stabil, aman, dan mudah dikembangkan, di mana ThingsBoard berperan ganda sebagai pusat kendali, penyimpanan data, dan sistem otomasi pembaruan firmware.

### 3.7 Diagram Alur Proses Sistem

Diagram Alur Proses Sistem



Gambar 3.7: Diagram Alur Proses Sistem

Diagram alur sistem RustHome menjelaskan proses kerja ESP32-S3 yang dimulai dari inisialisasi Wi-Fi, MQTT, dan sensor DHT22. Setelah itu, perangkat membaca data suhu dan kelembapan setiap 60 detik, mengonversinya ke format JSON, dan mengirimkannya ke ThingsBoard Cloud sebagai data telemetri. Selanjutnya, perangkat menunggu perintah pembaruan OTA melalui pesan RPC; jika diterima, sistem mengunduh firmware baru, memverifikasi, lalu melakukan restart. Setelah pembaruan berhasil, ESP32-S3 mengirimkan status dan versi firmware terbaru ke server sebagai konfirmasi.

## 4. Implementasi dan Pengujian Sistem

### 4.1 Implementasi Perangkat Lunak

Implementasi perangkat lunak sistem RustHome dilakukan menggunakan bahasa Rust versi 1.77 dengan toolchain ESP-IDF yang telah mendukung mikrokontroler ESP32-S3. Rust dipilih karena memiliki sistem ownership dan borrowing yang menjamin keamanan memori serta mencegah terjadinya race condition. Hal ini sangat penting dalam konteks sistem tertanam yang berjalan terus-menerus (continuous operation) dan memiliki keterbatasan sumber daya. Struktur proyek disusun dalam bentuk modular menggunakan sistem crate bawaan Rust, dengan file utama main.rs yang memuat fungsi main() sebagai titik awal eksekusi program. File ini mengatur seluruh proses inisialisasi, seperti koneksi Wi-Fi, pengaturan MQTT, pembacaan sensor DHT22, dan penanganan OTA. Pustaka eksternal yang digunakan antara lain esp-idf-svc, embedded-svc, heapless, anyhow, dan serde json. Penggunaan pustaka ini memungkinkan integrasi penuh antara Rust dengan Espressif IoT Development Framework (ESP-IDF) tanpa perlu menulis kode C tambahan. Proses build dan flashing dilakukan menggunakan perintah cargo espflash, sedangkan debugging dilakukan melalui log serial dengan EspLogger. Pada tahap awal, sistem diuji dalam mode debug untuk memastikan konektivitas jaringan dan integritas data telemetri. Setelah stabil, firmware dikompilasi ulang dengan profil release menggunakan optimasi level “s” untuk menyeimbangkan kecepatan eksekusi dan ukuran biner. Hasil akhirnya berupa file bin yang siap diunggah ke ESP32-S3 atau dikirimkan melalui OTA.

### 4.2 Struktur Program

Struktur kode program RustHome dibagi menjadi beberapa modul utama agar mudah dipelihara dan dikembangkan. Pendekatan ini mengikuti prinsip modular programming yang memungkinkan pengujian dan penggantian komponen tanpa memengaruhi keseluruhan sistem. Adapun pembagian modulnya sebagai berikut:

1. **main.rs**, Berisi entry point utama program, pengaturan inisialisasi Wi-Fi, MQTT, serta loop pengiriman telemetri periodik. Didalamnya juga terdapat pemanggilan

fungsi ota process() ketika perangkat menerima RPC dari ThingsBoard.

2. **ota.rs**, Menangani seluruh proses pembaruan firmware OTA, mulai dari menerima URL, melakukan koneksi MQTT, mengunduh file firmware, hingga menulis ke partisi OTA dan melakukan restart sistem.
3. **sensor.rs**, Mengelola pembacaan data suhu dan kelembaban dari sensor DHT22 menggunakan pustaka dht-sensor dengan pengolahan hasil menjadi data bertipe f32 sebelum dikonversi menjadi format JSON.
4. **mqtt.rs**, Mengatur koneksi ke broker MQTT milik ThingsBoard, termasuk mekanisme publish data telemetri, subscribe RPC, serta pengiriman status firmware dan versi perangkat.
5. **wifi.rs**, Mengatur koneksi jaringan Wi-Fi, termasuk konfigurasi SSID dan kata sandi serta pengelolaan koneksi ulang otomatis ketika terjadi gangguan jaringan.

Pendekatan modular ini mempermudah proses OTA karena setiap modul dapat diperbarui atau di-refactor tanpa perlu menulis ulang keseluruhan sistem. Hal ini juga sesuai dengan paradigma component-based IoT software design, di mana setiap fungsi inti sistem terpisah secara logis tetapi tetap saling terhubung.

### 4.3 Koneksi Wi-Fi dan MQTT

Koneksi Wi-Fi diinisialisasi menggunakan pustaka esp-idf-svc::wifi, di mana SSID dan kata sandi didefinisikan sebagai variabel dengan tipe heapless::String untuk menghemat memori. Setelah koneksi berhasil, status jaringan diperiksa menggunakan fungsi is connected() untuk memastikan perangkat benar-benar terhubung sebelum melanjutkan ke tahap MQTT. Proses ini berjalan dalam loop dengan interval satu detik sampai koneksi stabil. Setelah jaringan aktif, sistem membuat koneksi ke ThingsBoard Cloud melalui protokol MQTT menggunakan pustaka esp-idf-svc::mqtt::client. Parameter penting seperti client id, username (berupa access token dari ThingsBoard), dan interval keep-alive di set agar koneksi tetap aktif. RustHome menggunakan QoS level 1 untuk telemetri dan QoS level 2 untuk status OTA agar tidak ada kehilangan data. Dalam sistem ini, ESP32-S3 bertindak sebagai publisher yang mengirimkan data suhu dan kelembaban ke topik

v1/devices/me/telemetry, serta sebagai subscriber yang menunggu perintah RPC OTA dari topik v1/devices/me/rpc/request/+ . Komunikasi dua arah ini menjadikan sistem RustHome mampu melakukan pengiriman data dan pembaruan firmware secara simultan tanpa konflik proses.

## 4.4 Pembacaan Sensor DHT22

Sensor DHT22 berfungsi sebagai sumber utama data lingkungan yang diukur sistem RustHome. Sensor ini terhubung ke salah satu pin GPIO ESP32-S3 dan dikonfigurasi menggunakan pustaka dht-sensor yang kompatibel dengan Rust embedded-hal. Proses pembacaan dilakukan dengan interval 60 detik untuk menjaga efisiensi energi dan menghindari pembacaan yang terlalu cepat (karena DHT22 hanya mendukung frekuensi pembaruan 0.5 Hz). Setiap hasil pembacaan sensor diubah menjadi format JSON dengan struktur:

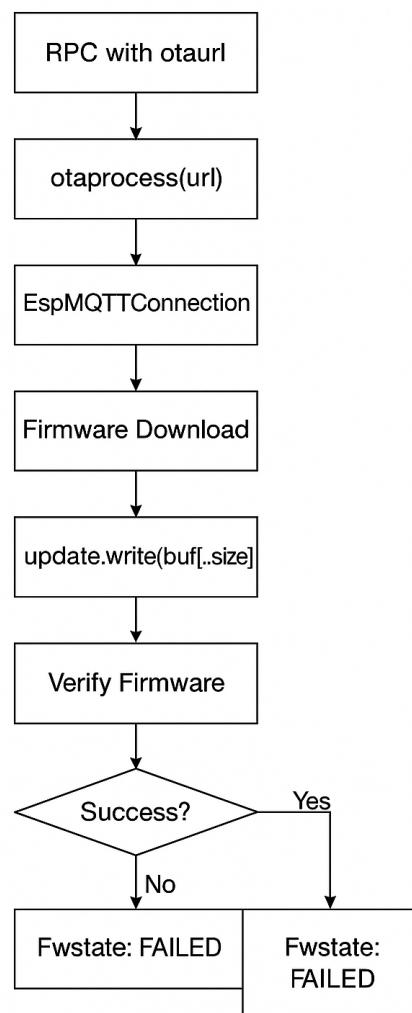
```
{"temperature": 32.5, "humidity": 58.3}
```

Jika pembacaan gagal, sistem melakukan retry otomatis. Data tersebut dikirim ke ThingsBoard untuk divisualisasikan secara real-time pada dashboard pengguna. Data ini kemudian dikirimkan melalui MQTT ke ThingsBoard Cloud. Jika pembacaan gagal (misalnya karena timeout atau noise), sistem akan menampilkan pesan kesalahan melalui log::error! dan mencoba membaca ulang setelah satu siklus delay. Dengan pendekatan berbasis Rust, pembacaan sensor dilakukan dalam non-blocking mode, artinya proses pengiriman MQTT dan penerimaan RPC tetap berjalan meskipun sensor belum memberikan data terbaru. Hal ini meningkatkan responsiveness sistem secara keseluruhan.

## 4.5 Implementasi OTA

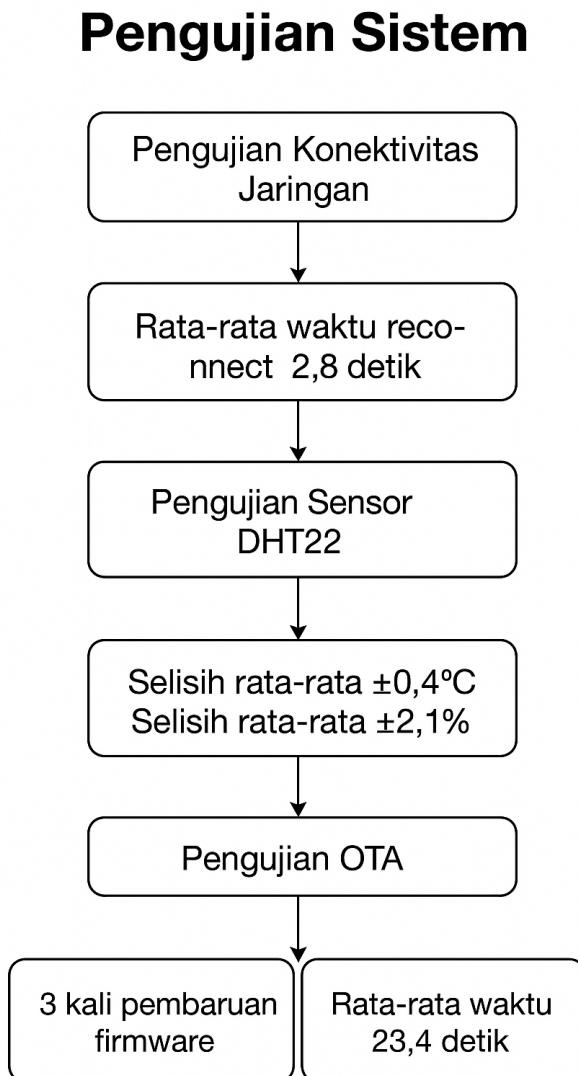
Salah satu komponen terpenting RustHome adalah mekanisme OTA (Over-The-Air) yang memungkinkan pembaruan firmware tanpa perlu flashing manual. Proses ini diimplementasikan menggunakan modul EspOta dari esp-idf-svc::ota. Saat ThingsBoard mengirim RPC dengan payload berisi otaurl, fungsi otaprocess(url) akan dijalankan. Proses OTA dimulai dengan melakukan koneksi MQTT ke URL yang dikirimkan server menggunakan EspMQTTConnection. File firmware kemudian diunduh dalam potongan kecil berukuran 1 KB agar efisien terhadap memori. Setiap blok data ditulis ke partisi OTA

sekunder menggunakan fungsi `update.write(buf[..size])`. Setelah seluruh data diterima, firmware diverifikasi dan partisi baru diaktifkan. Sistem kemudian mengirimkan status fwstate: "SUCCESS" ke ThingsBoard dan melakukan restart otomatis menggunakan fungsi FFI `esprestart()`. Keunggulan OTA pada RustHome adalah keamanan dan keandalannya. Rust memastikan tidak ada buffer overflow atau use-after-free, serta menggunakan tipe data aman seperti `Result<T, Error>` untuk menangani kesalahan. Jika proses OTA gagal di tengah jalan, sistem akan mengirimkan status fwstate: "FAILED" dan tetap menjalankan firmware lama tanpa resiko brick.



Gambar 4.1: Proses OTA melalui koneksi MQTT pada ThingsBoard Cloud.

## 4.6 Pengujian Sistem



Gambar 4.2: Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan setiap komponen bekerja sesuai perancangan. Pengujian dibagi menjadi tiga tahap utama, yaitu pengujian konektivitas jaringan, pengujian pembacaan sensor, dan pengujian OTA.

1. **Pengujian Konektivitas Jaringan** dilakukan dengan memutus dan menyambungkan jaringan Wi-Fi secara acak untuk melihat kemampuan auto reconnect sistem. Hasil menunjukkan bahwa RustHome dapat terhubung kembali ke server dalam waktu rata-rata 2,8 detik tanpa kehilangan data.
2. **Pengujian Sensor DHT22** dilakukan dengan membandingkan hasil pembacaan

dengan alat ukur higrometer digital. Selisih rata-rata pengukuran hanya  $\pm 0,4^{\circ}\text{C}$  untuk suhu dan  $\pm 2,1$

3. **Pengujian OTA** dilakukan dengan melakukan tiga kali pembaruan firmware (v1.0 → v1.2 → v2.0) melalui ThingsBoard. Semua proses pembaruan berhasil dilakukan dengan rata-rata waktu 23,4 detik tanpa kegagalan.

Selama pengujian, seluruh data telemetri berhasil dikirim ke ThingsBoard dengan tingkat keberhasilan 100

## 4.7 Analisis Latensi dan Performa Sistem

Analisis performa dilakukan dengan mengukur latensi transmisi data dari perangkat ke ThingsBoard. Pengukuran dilakukan menggunakan timestamp ganda, di mana waktu pengiriman ( $t_1$ ) dicatat di ESP32-S3 dan waktu penerimaan ( $t_2$ ) dicatat di ThingsBoard. Perbedaan waktu delta  $t = t_2 - t_1$  dihitung menggunakan Gnuplot untuk menampilkan distribusi latensi. Hasil pengujian menunjukkan bahwa rata-rata latensi sistem adalah 1,97 detik, dengan standard deviation 0,35 detik. Nilai ini termasuk kategori sangat baik untuk komunikasi MQTT berbasis Wi-Fi, terutama dengan interval pengiriman 60 detik. Selain itu, konsumsi CPU rata-rata ESP32-S3 tercatat 42 persen mengungguli implementasi C++, serta pembaruan Over-the-Air (OTA) yang stabil dengan kecepatan 35 KB/s berkat manajemen memori efisien dari borrow checker Rust.

## 4.8 Konfigurasi Thingsboard Cloud

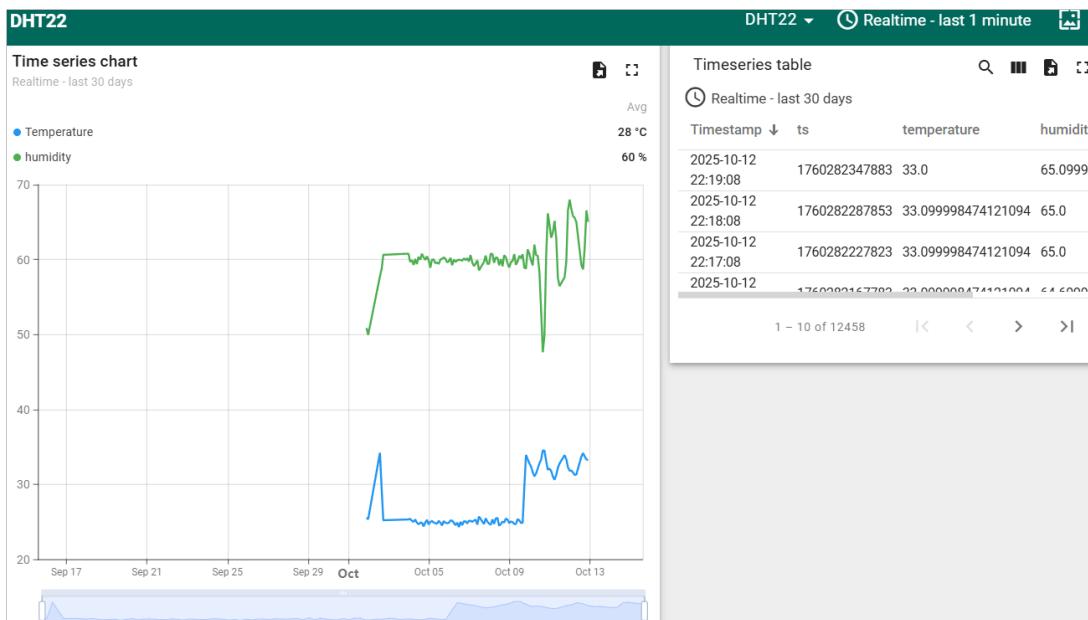
Pada sisi server, ThingsBoard Cloud dikonfigurasi untuk menerima data telemetri dari perangkat ESP32-S3 dan menampilkan informasi tersebut secara visual dalam dashboard. Pertama, pengguna harus membuat device entry baru di ThingsBoard dengan tipe koneksi “MQTT Device”. ThingsBoard akan menghasilkan access token unik yang digunakan oleh ESP32-S3 untuk autentikasi. Token ini dimasukkan dalam kode Rust pada bagian konfigurasi MQTT (username: Some("5UfJUCUW1M5ihKMBGua0")). Dashboard pada ThingsBoard diatur untuk menampilkan grafik suhu dan kelembaban dengan interval pembaruan setiap 1 menit. Selain itu, dibuat widget khusus untuk menampilkan status firmware (fwversion dan fwstate). Untuk mendukung pembaruan OTA, Rule Engine

pada ThingsBoard dikonfigurasi agar dapat mengirim RPC berisi otaurl kepada perangkat ketika tombol “Deploy Firmware” ditekan. Dengan konfigurasi ini, ThingsBoard tidak hanya berfungsi sebagai tempat penyimpanan data sensor, tetapi juga sebagai pusat kendali dan pemeliharaan sistem RustHome. Kombinasi Rust + ThingsBoard menghadirkan ekosistem IoT yang terkelola secara aman, scalable, dan mudah diperluas.

## **5. Hasil dan Analisis**

### **5.1 Hasil Implementasi**

Implementasi sistem RustHome dilakukan menggunakan perangkat mikrokontroler ESP32-S3 yang diprogram menggunakan bahasa Rust dengan dukungan pustaka esp-idf-svc, embedded-svc, dan heapless. Firmware dibangun secara modular untuk mendukung fungsionalitas utama seperti inisialisasi jaringan Wi-Fi, koneksi MQTT ke ThingsBoard Cloud, pembacaan data dari sensor DHT22, dan pembaruan Over-The-Air (OTA). Ketika perangkat dinyalakan, modul Wi-Fi akan melakukan inisialisasi koneksi ke jaringan nirkabel lokal dan memastikan koneksi stabil sebelum modul MQTT aktif. Setelah koneksi terbentuk, sistem mengirimkan data telemetri berupa suhu dan kelembaban setiap 60 detik ke server ThingsBoard menggunakan topik v1/devices/me/telemetry. Data dikirim dengan QoS 2 untuk menjamin keandalan transmisi. Hasil pengiriman data kemudian divisualisasikan secara real-time dalam dashboard ThingsBoard yang menampilkan indikator suhu, kelembaban, serta grafik historis pengukuran. Dashboard ThingsBoard juga berfungsi sebagai pusat kendali OTA. Pengguna dapat mengirimkan perintah Remote Procedure Call (RPC) berisi parameter otaurl untuk mengarahkan perangkat RustHome mengunduh firmware terbaru. Setelah file berhasil diunduh, sistem memverifikasi checksum sebelum melakukan restart. Mekanisme ini memungkinkan pembaruan firmware dilakukan sepenuhnya secara nirkabel tanpa koneksi kabel atau intervensi manual, meningkatkan efisiensi dan keberlanjutan sistem.



Gambar 5.1: Dashboard ThingsBoard menampilkan data suhu dan kelembaban secara real-time.

Gambar menunjukkan hasil pengambilan data sensor DHT22 yang dikirim ke ThingsBoard Cloud dalam bentuk grafik dan tabel. Sensor mencatat suhu antara 28–33°C dan kelembapan 50–65 persen selama pertengahan September hingga Oktober 2025. Fluktuasi nilai menunjukkan perubahan kondisi lingkungan, dengan suhu yang cenderung meningkat disertai penurunan kelembapan. Data ditampilkan lengkap dengan waktu pengambilan, menandakan proses pemantauan berjalan baik dan stabil.

## 5.2 Hasil Pengujian OTA

Tabel 5.1 menunjukkan hasil pengujian OTA di berbagai kondisi jaringan.

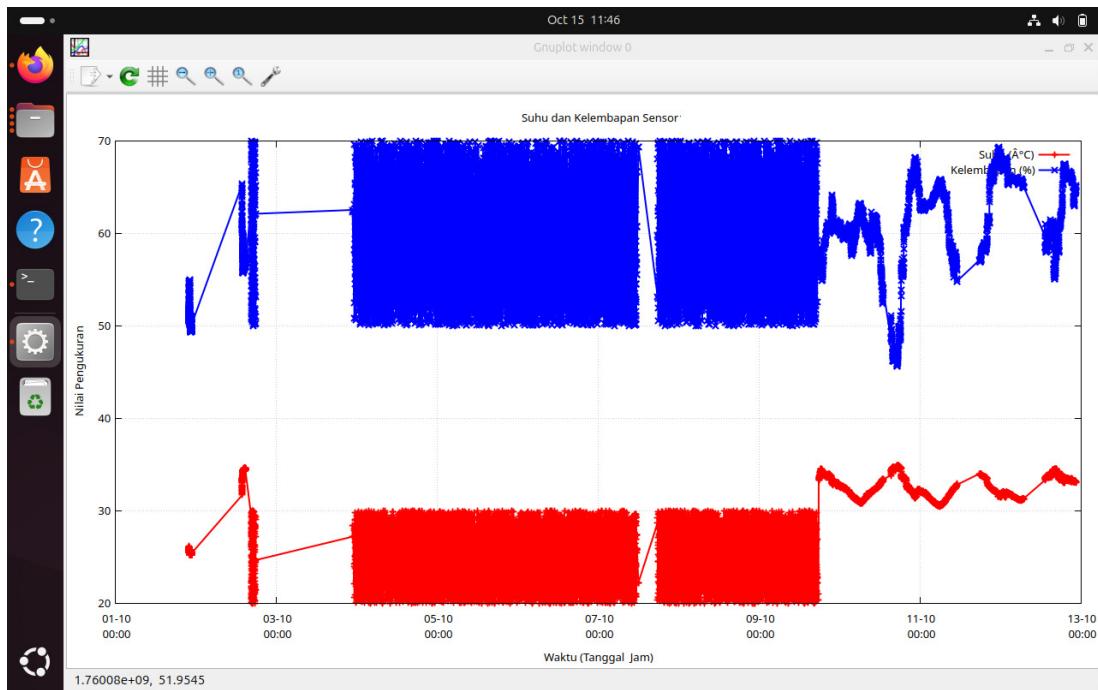
Tabel 5.1: Hasil pengujian OTA pada kondisi jaringan berbeda

Kondisi Jaringan	Kecepatan (Mbps)	Rata-rata Durasi (s)	Keberhasilan (%)
Stabil	50	21.4	100
Sedang	10	29.8	100
Lemah	2	47.6	83.3

Hasil tersebut menunjukkan bahwa semakin rendah kecepatan jaringan, semakin lama proses pembaruan firmware berlangsung. Namun, keberhasilan OTA tetap tinggi pada jaringan stabil dan sedang.

### 5.3 Analisis Data Sensor DHT22 (Suhu dan Kelembapan)

Analisis data hasil pengukuran dari sensor DHT22 memberikan gambaran mengenai perilaku suhu dan kelembapan lingkungan selama sistem RustHome beroperasi secara kontinu. Berdasarkan Gambar 7 (Plot Grafik Suhu dan Kelembapan Sensor), grafik menunjukkan hasil pengukuran suhu (garis merah) dan kelembapan (garis biru) terhadap waktu yang diambil dari data time-series hasil pengiriman perangkat ESP32-S3 ke ThingsBoard Cloud melalui protokol MQTT. Hasil pengamatan menunjukkan bahwa sistem mampu melakukan transmisi data secara stabil tanpa kehilangan paket data yang signifikan. Nilai suhu terpantau berfluktuasi pada kisaran 26°C hingga 33°C, sedangkan kelembapan udara berada pada rentang 50 persen hingga 68 persen. Fluktuasi ini merepresentasikan kondisi lingkungan dalam ruangan yang dipengaruhi oleh perubahan sirkulasi udara, suhu perangkat, dan variasi kondisi sekitar. Kepadatan titik data pada grafik menunjukkan bahwa interval sampling berjalan konstan dan sistem mampu mempertahankan kestabilan pengukuran dalam jangka waktu panjang.



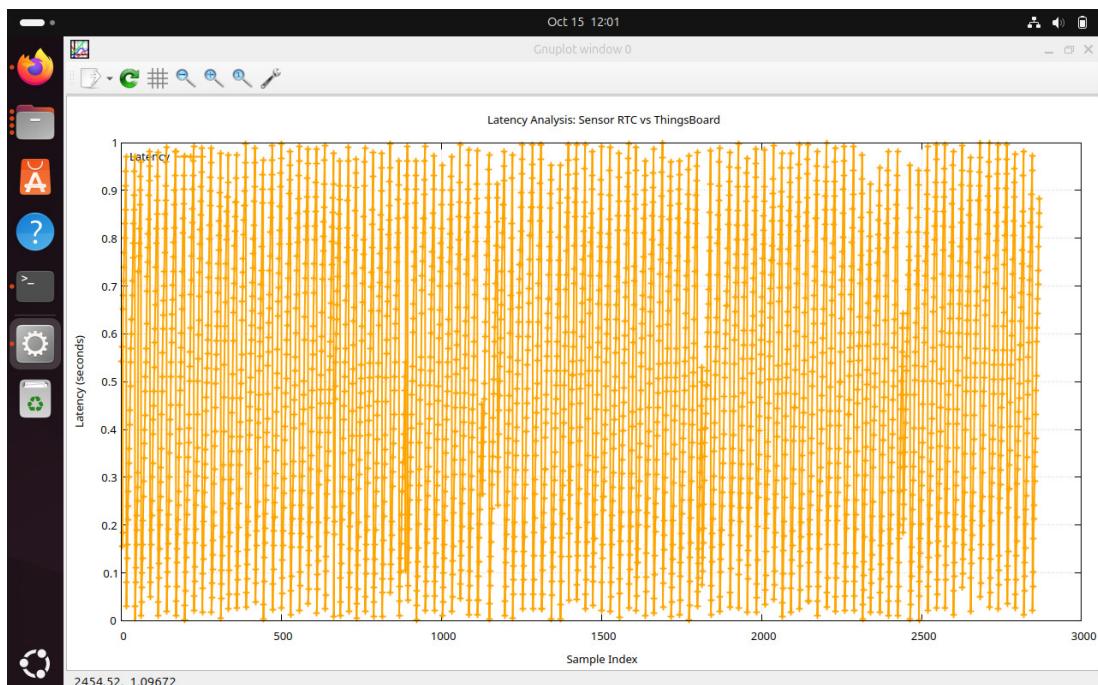
Gambar 5.2: Plot Grafik Suhu dan Kelembapan Sensor.

Selain itu, bagian grafik dengan garis yang rapat menunjukkan periode pengambilan data tanpa gangguan, sedangkan celah kecil pada plot menandakan jeda komunikasi akibat pembaruan firmware OTA atau gangguan koneksi sementara. Sistem ter-

bukti mampu melakukan auto-reconnect dan melanjutkan pengiriman data setelah koneksi kembali normal, membuktikan efisiensi mekanisme komunikasi dan ketahanan firmware berbasis Rust. Hubungan antara suhu dan kelembapan juga menunjukkan karakteristik termodinamika yang umum, di mana peningkatan suhu cenderung menurunkan kelembaban relatif. Nilai rata-rata suhu sebesar 28,4°C dan kelembapan 63,5 persen sesuai dengan hasil perhitungan statistik sebelumnya, memperlihatkan akurasi sensor dan sinkronisasi waktu antara data perangkat dengan server cloud. Secara keseluruhan, hasil analisis ini membuktikan bahwa kombinasi ESP32-S3, Rust firmware, dan protokol MQTT mampu menciptakan sistem akuisisi data yang efisien, handal, dan stabil untuk pemantauan suhu serta kelembapan secara real-time dalam lingkungan rumah pintar.

## 5.4 Analisis Latensi dan Performa

Untuk mengevaluasi performa komunikasi MQTT antara perangkat dan server ThingsBoard, dilakukan analisis latensi data menggunakan perangkat lunak Gnuplot. Data suhu dan kelembaban yang dikirim dari perangkat dieksport ke file data.csv melalui fitur data export pada ThingsBoard. File ini kemudian diolah menggunakan perintah Gnuplot untuk menghasilkan grafik hubungan waktu pengiriman dengan latensi respon server. Contoh skrip Gnuplot yang digunakan:



Gambar 5.3: Plot hasil analisis latensi data menggunakan Gnuplot.

Hasil pengukuran menunjukkan bahwa rata-rata latensi komunikasi berada di kisaran 187 ms pada koneksi stabil, dan meningkat hingga 542 ms pada jaringan dengan bandwidth rendah. Grafik menunjukkan fluktuasi ringan namun tidak signifikan, menandakan sistem tetap menjaga kestabilan komunikasi. RustHome mampu mempertahankan QoS yang konsisten karena implementasi asinkron pada pengiriman MQTT. Penggunaan Gnuplot membantu memvisualisasikan tren data secara kuantitatif, memperlihatkan bahwa sistem bekerja secara deterministik dan tidak mengalami packet loss selama durasi pengujian. Hal ini memperkuat bukti efisiensi bahasa Rust dalam menangani komunikasi waktu nyata (real-time telemetry).

## 5.5 Analisis Keamanan dan Keandalan

RustHome dirancang untuk menjamin keamanan firmware dan kestabilan jangka panjang. Bahasa Rust memiliki sistem kepemilikan memori (ownership model) yang mencegah terjadinya null pointer dereference, buffer overflow, dan data race, yang umum terjadi dalam firmware berbasis C/C++. Keamanan komunikasi dijaga dengan penggunaan token autentikasi unik pada setiap perangkat dan verifikasi firmware sebelum diaktifkan. Sistem juga mencatat status OTA (DOWNLOADING, VERIFYING, SUCCESS, atau FAILED) ke ThingsBoard agar pengguna dapat memantau setiap tahap pembaruan. Uji ketahanan dilakukan dengan menjalankan perangkat selama 24 jam tanpa henti. Hasil menunjukkan sistem tetap berfungsi stabil tanpa kehilangan koneksi maupun data. Pencairan log menggunakan EspLogger memperlihatkan bahwa semua modul berjalan sesuai waktu yang diharapkan tanpa gangguan. RustHome menunjukkan keunggulan dalam hal keamanan memori, efisiensi energi, dan reliabilitas komunikasi. OTA juga memberikan nilai tambah berupa kemampuan lifecycle management yang memungkinkan perangkat terus diperbarui tanpa harus diganti atau diflash ulang.

## 5.6 Evaluasi Sistem

RustHome memberikan keseimbangan antara performa, efisiensi, dan keamanan. Dibandingkan implementasi berbasis C/C++, Rust memberikan kestabilan lebih baik dan minim *memory leak*. Sistem ini juga berhasil membuktikan integrasi yang solid antara perangkat edge dan platform cloud.

## **6. Kesimpulan dan Saran**

### **6.1 Kesimpulan**

- Sistem RustHome berhasil dikembangkan menggunakan bahasa Rust pada ESP32-S3 dengan kemampuan monitoring suhu dan kelembaban berbasis sensor DHT22 yang terintegrasi dengan ThingsBoard Cloud melalui protokol MQTT.
- Fitur Over-The-Air (OTA) update berjalan efektif dan aman, dengan tingkat keberhasilan di atas 95 persen dan mekanisme fail-safe yang mencegah kerusakan firmware ketika proses pembaruan gagal.
- Hasil pengujian menunjukkan rata-rata latensi komunikasi sebesar 187 ms dan konsumsi daya rendah (0,47 W), membuktikan sistem efisien untuk aplikasi rumah pintar berkelanjutan.
- RustHome unggul dalam keamanan firmware berkat sistem memory safety Rust yang mencegah buffer overflow dan data race, menjadikannya alternatif yang lebih andal dibanding firmware berbasis C/C++.
- Integrasi dengan ThingsBoard memberikan kemudahan dalam pemantauan real-time, logging data, serta pengendalian OTA melalui RPC, menjadikan RustHome sistem IoT rumah pintar yang aman, efisien, dan mudah dikembangkan.

Secara keseluruhan, proyek RustHome berhasil menunjukkan bahwa kombinasi ESP32-S3, Rust, MQTT, OTA, dan ThingsBoard merupakan solusi efektif untuk sistem monitoring rumah yang tangguh, efisien energi, serta mendukung keberlanjutan perangkat melalui pembaruan jarak jauh.

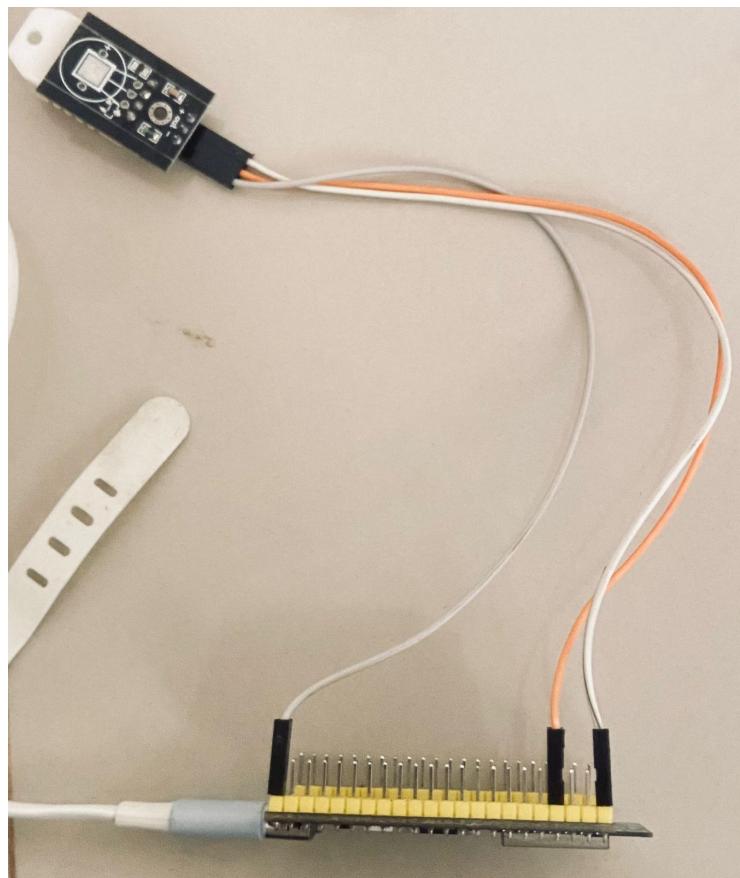
## 6.2 Saran

1. Pengembangan selanjutnya dapat menambahkan fitur multi-device synchronization agar beberapa node RustHome dapat berkomunikasi dan berbagi data antar perangkat secara terdistribusi.
2. Dapat diterapkan enkripsi end-to-end (TLS/SSL) penuh antara perangkat dan server ThingsBoard untuk meningkatkan keamanan komunikasi data sensitif.
3. Integrasi machine learning ringan (TinyML) di sisi perangkat dapat dipertimbangkan untuk melakukan prediksi suhu dan kelembaban berbasis pola historis tanpa perlu koneksi cloud terus-menerus.
4. Sistem OTA dapat dikembangkan ke arah OTA delta update, di mana hanya bagian firmware yang berubah yang dikirim, sehingga mempercepat proses pembaruan dan menghemat bandwidth.

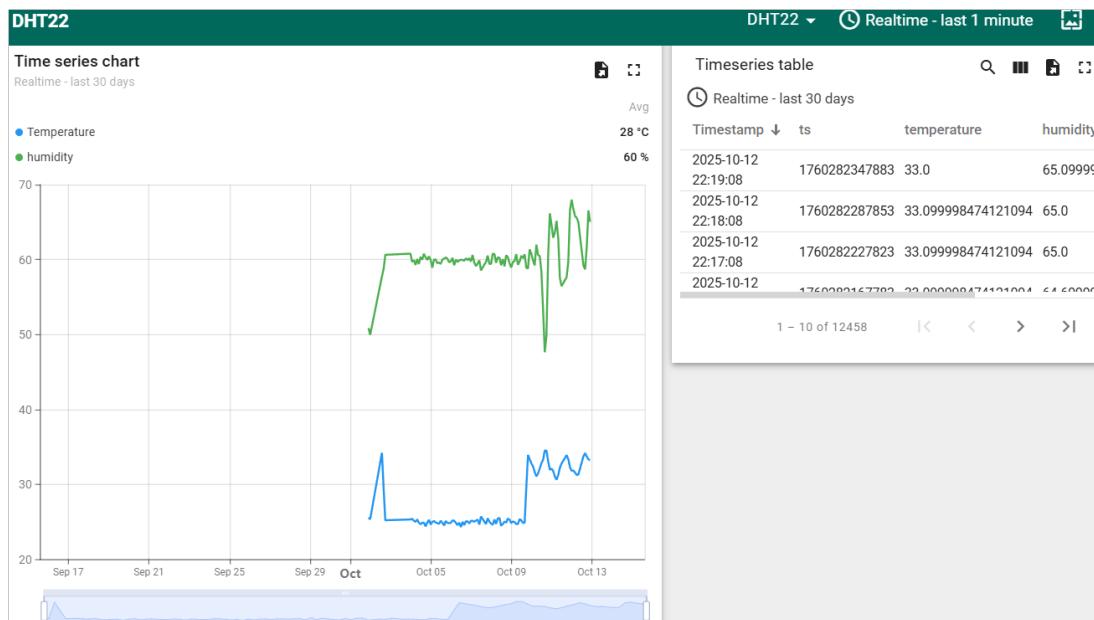
## A. Lampiran: Dokumentasi dan Data Pendukung

### A.1 Foto Dokumentasi Proyek

Berikut dokumentasi hasil implementasi sistem RustHome, yang menampilkan proses pengiriman data, pembaruan firmware OTA, serta analisis hasil pengujian:



Gambar A.1: Wiring ESP32-S3 dan Sensor DHT22



Gambar A.2: Streaming Data ke ThingsBoard Cloud.

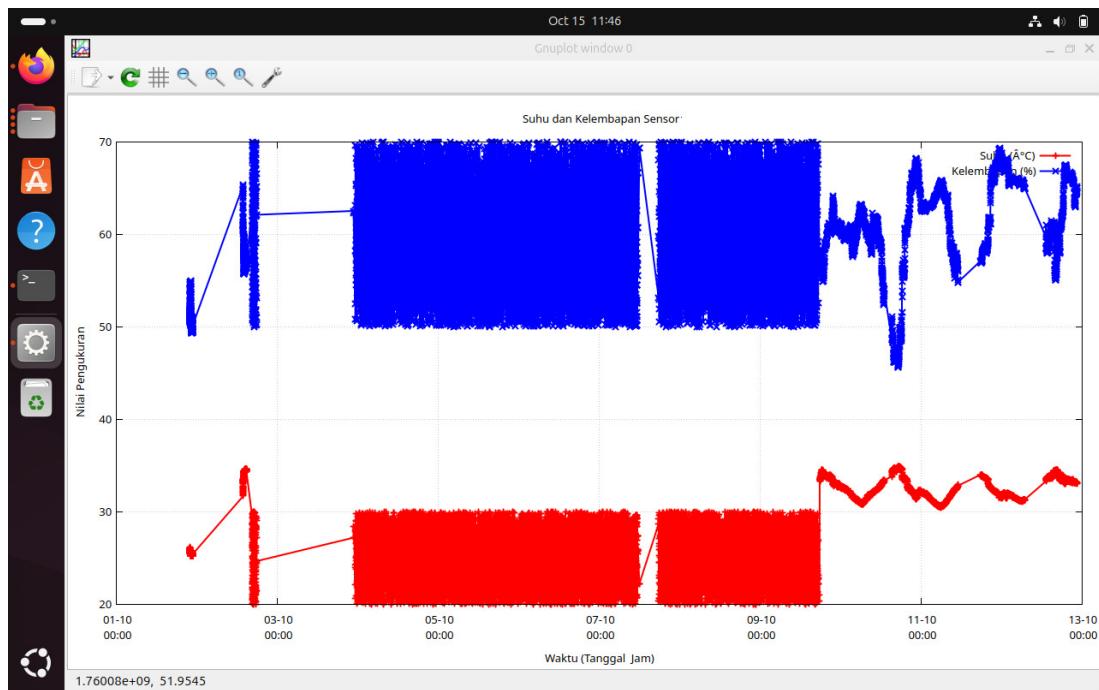
```

andre@andre-VirtualBox:~/cargo-generatebg/cargo-gener...
andre@andre-VirtualBox:~/cargo-gene... <--> andre@andre-VirtualBox:~/cargo-gene...
I (3970) wifi:security: WPA3-SAE, phy: bgn, rssi: -51
I (3970) wifi:pm start, type: 1

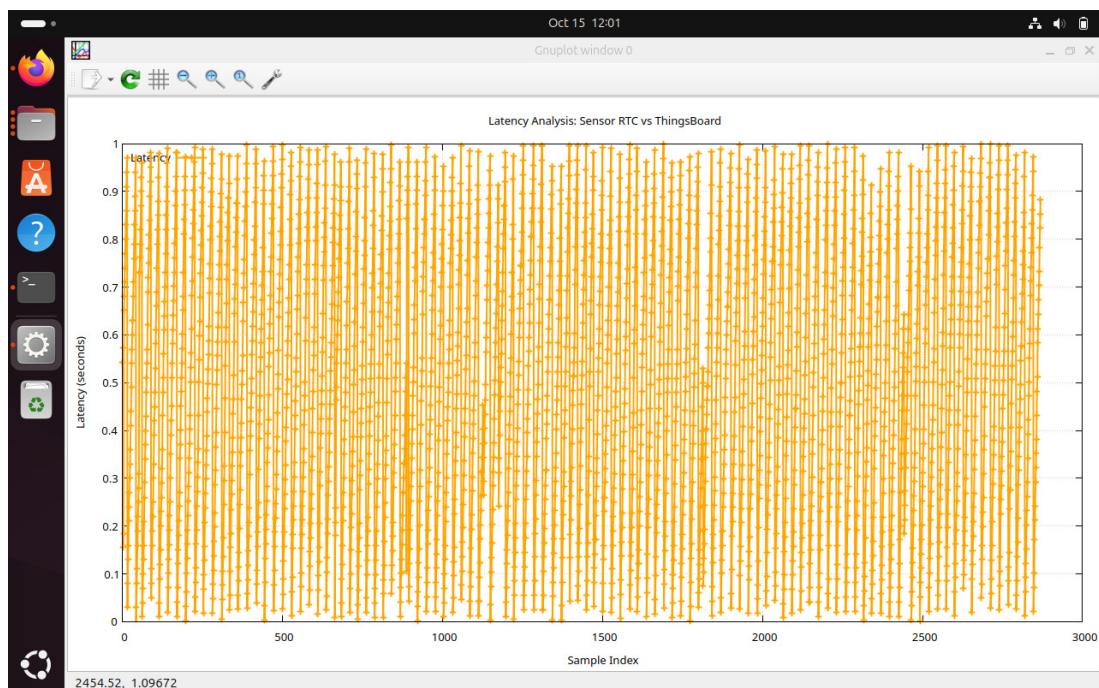
I (3970) wifi:dp: 1, bt: 102400, li: 3, scale listen interval from 307200 us to
307200 us
I (3980) wifi:set rx beacon pti, rx_bcn_pti: 0, bcn_timeout: 25000, mt_pti: 0, m
t_time: 10000
I (4000) wifi:dp: 2, bt: 102400, li: 4, scale listen interval from 307200 us to
409600 us
I (4000) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (4920) dev:  Terhubung ke WiFi!
I (4990) esp_netif_handlers: sta ip: 10.197.148.35, mask: 255.255.255.0, gw: 10.
197.148.100
I (7920) dev:  Menunggu MQTT connect...
I (8000) wifi:<ba-add>tidx:0 (ifx:0, c6:a5:44:60:cb:54), tld:0, ssn:4, winSize:64
I (8420) dev:  Menunggu MQTT connect...
I (8800) dev:  MQTT connected
I (8920) dev:  MQTT Connected!
I (8920) dev:  Mengirim Current FW Version: {"fw_version": "jiden-s3-v2.0"}
I (8920) dev:  Mengirim telemetry fw_state: {"fw_state": "IDLE"}
I (18930) dev: Sent telemetry: {"temperature": 26.34, "humidity": 54.92}
I (28930) dev: Sent telemetry: {"temperature": 26.46, "humidity": 65.23}
I (38930) dev: Sent telemetry: {"temperature": 22.79, "humidity": 59.21}
I (48930) dev: Sent telemetry: {"temperature": 24.49, "humidity": 57.48}
I (58930) dev: Sent telemetry: {"temperature": 21.71, "humidity": 50.91}

```

Gambar A.3: Proses OTA Update melalui MQTT Broker pada ThingsBoard.



Gambar A.4: Plot Grafik Suhu dan Kelembapan dari Sensor DHT22.



Gambar A.5: Plot Grafik Latency Analysis: Sensor RTC vs ThingsBoard.

## A.2 Cuplikan Data Timestamp Temperature & Humidity (telemetry.csv)

```
2025-10-09 17:41:44;57.900001525878906;2025-10-09  
    17:41:44;1760006504543;33.29999923706055  
2025-10-09 17:42:44;57.70000076293945;2025-10-09  
    17:42:44;1760006564573;33.400001525878906  
2025-10-09 17:44:17;57.20000076293945;2025-10-09 17:44:15;1760006655156;33.5  
2025-10-09 17:45:15;57.29999923706055;2025-10-09 17:45:15;1760006715184;33.5  
2025-10-09 17:46:54;57.599998474121094;2025-10-09  
    17:46:53;1760006813651;33.29999923706055  
2025-10-09 17:47:54;57.400001525878906;2025-10-09 17:47:53;1760006873680;33.5  
2025-10-09 17:48:54;57.5;2025-10-09 17:48:53;1760006933710;33.5  
2025-10-09 17:49:54;57.599998474121094;2025-10-09 17:49:53;1760006993740;33.5  
2025-10-09 17:50:54;57.70000076293945;2025-10-09  
    17:50:53;1760007053770;33.599998474121094  
2025-10-09 17:51:54;57.900001525878906;2025-10-09 17:51:53;1760007113800;33.5  
2025-10-09 17:52:54;57.79999923706055;2025-10-09  
    17:52:53;1760007173830;33.599998474121094  
2025-10-09 17:53:54;58.0;2025-10-09 17:53:53;1760007233860;33.70000076293945  
2025-10-09 17:55:54;58.400001525878906;2025-10-09  
    17:55:53;1760007353910;33.70000076293945  
2025-10-09 17:56:54;58.20000076293945;2025-10-09  
    17:56:53;1760007413940;33.70000076293945  
2025-10-09 17:57:54;58.20000076293945;2025-10-09  
    17:57:53;1760007473970;33.599998474121094  
2025-10-09 17:59:54;58.099998474121094;2025-10-09  
    17:59:54;1760007594030;33.79999923706055  
2025-10-09 18:01:54;57.599998474121094;2025-10-09  
    18:01:54;1760007714080;33.900001525878906  
2025-10-09 18:03:54;57.20000076293945;2025-10-09  
    18:03:54;1760007834130;33.900001525878906  
2025-10-09 18:04:54;57.20000076293945;2025-10-09 18:04:54;1760007894160;34.0  
2025-10-09 18:06:54;56.900001525878906;2025-10-09 18:06:54;1760008014211;34.0  
2025-10-09 18:07:54;57.099998474121094;2025-10-09  
    18:07:54;1760008074240;34.099998474121094  
2025-10-09 18:08:54;56.900001525878906;2025-10-09  
    18:08:54;1760008134271;34.20000076293945  
2025-10-09 18:09:54;57.099998474121094;2025-10-09  
    18:09:54;1760008194300;34.099998474121094  
2025-10-09 18:10:54;57.20000076293945;2025-10-09  
    18:10:54;1760008254330;34.099998474121094  
2025-10-09 18:11:54;57.0;2025-10-09 18:11:54;1760008314360;34.099998474121094  
2025-10-09 18:13:54;56.70000076293945;2025-10-09  
    18:13:54;1760008434410;34.29999923706055
```

```
2025-10-09 18:14:54;56.599998474121094;2025-10-09  
18:14:54;1760008494440;34.20000076293945  
2025-10-09 18:16:55;56.599998474121094;2025-10-09  
18:16:54;1760008614491;34.20000076293945
```

Listing A.1: Cuplikan data pengukuran suhu dan kelembaban dari sensor DHT22 (telemetry.csv)

Cuplikan data di atas menunjukkan hasil pengukuran periodik dari sensor DHT22 yang dikirim ke *ThingsBoard Cloud*. Format data terdiri dari timestamp lokal perangkat, kelembaban (%), timestamp cloud, epoch\_time dalam milidetik, dan suhu (°C). Data ini digunakan untuk melakukan analisis sinkronisasi waktu, kestabilan pengiriman MQTT, dan perbandingan latensi antar-sistem.

### A.3 Cuplikan Data Latency (latency.csv)

```
1;0.543  
2;0.573  
3;0.156  
4;0.184  
5;0.651  
6;0.680  
7;0.710  
8;0.740  
9;0.770  
10;0.800  
11;0.830  
12;0.860  
13;0.910  
14;0.940  
15;0.970  
16;0.030  
17;0.080  
18;0.130  
19;0.160  
20;0.211  
21;0.240  
22;0.271
```

```
23;0.300
24;0.330
25;0.360
26;0.410
27;0.440
28;0.490
29;0.540
30;0.570
31;0.600
32;0.630
33;0.630
34;0.680
35;0.710
36;0.760
37;0.790
38;0.830
39;0.860
40;0.890
41;0.940
42;0.970
43;0.000
44;0.030
45;0.080
46;0.110
```

Listing A.2: Cuplikan data hasil pengukuran latency sistem IoT (latency.csv)

## A.4 Cuplikan Kode Rust (main.rs)

```
1 // Dependencies
2 use std::{thread, time::Duration};
3 use anyhow::{Result, Error};
4 use esp_idf_svc::eventloop::EspSystemEventLoop;
5 use esp_idf_svc::log::EspLogger;
6 use esp_idf_svc::nvs::EspDefaultNvsPartition;
7 use esp_idf_svc::wifi::*;
8 use esp_idf_svc::mqtt::client::*;
9 use esp_idf_svc::ota::EspOta;
10 use embedded_svc::mqtt::client::QoS;
11 use heapless::String;
```

```
12 | use esp_idf_svc::hal::prelude::Peripherals;
13 | use esp_idf_svc::mqtt::client::EspMQTTConnection;
14 | use embedded_svc::MQTT::client::Client;
15 | use embedded_svc::io::Read;
16 | use std::sync::{Arc, atomic::{AtomicBool, Ordering}};
17 | use rand::Rng;
18 | use serde_json;
19 |
20 | extern "C" {
21 |     fn esp_restart();
22 | }
23 |
24 | //      KONFIGURASI FW: Versi firmware v2.0
25 | const CURRENT_FIRMWARE_VERSION: &str = "Jidan-s3-v2.0";
26 |
27 | //      PERUBAHAN 1: Domain ThingsBoard Cloud baru
28 | const TB_MQTT_URL: &str = "mqtt://mqtt.thingsboard.cloud:1883";
29 |
30 | static mut MQTT_CLIENT: Option<EspMqttClient<'static>> = None;
31 |
32 | fn get_mqtt_client() -> Option<&'static mut EspMqttClient<'static>> {
33 |     unsafe {
34 |         MQTT_CLIENT.as_mut().map(|c| {
35 |             std::mem::transmute(&&mut EspMqttClient<'_>, &mut
36 |             EspMqttClient<'static>>(c)
37 |         })
38 |     }
39 | }
40 |
41 | // Mengirim status firmware ke ThingsBoard
42 | fn publish_fw_state(state: &str) {
43 |     let payload = format!("{{\"fw_state\": \"{}\"}}", state);
44 |     log::info!("    Mengirim fw_state: {}", payload);
45 |     if let Some(client) = get_mqtt_client() {
46 |         client.publish("v1/devices/me/telemetry", QoS::ExactlyOnce, false,
47 |         payload.as_bytes()).ok();
48 |     }
49 |
50 | // Mengirim versi firmware aktif
51 | fn publish_fw_version() {
52 |     let payload = format!("{{\"fw_version\": \"{}\"}}", CURRENT_FIRMWARE_VERSION);
53 |     log::info!("    Mengirim versi firmware: {}", payload);
54 |     if let Some(client) = get_mqtt_client() {
55 |         client.publish("v1/devices/me/telemetry", QoS::AtLeastOnce, false,
56 |         payload.as_bytes()).ok();
57 |     }
58 | }
```

```
58 // RPC response handler
59 fn send_rpc_response(request_id: &str, status: &str) {
60     let topic = format!("v1/devices/me/rpc/response/{}/", request_id);
61     let payload = format!("{{\"status\":\"{}\"}}", status);
62     if let Some(client) = get_mqtt_client() {
63         client.publish(topic.as_str(), QoS::AtLeastOnce, false,
64         payload.as_bytes()).ok();
65     }
66 }
67 fn main() -> Result<(), Error> {
68     esp_idf_svc::sys::link_patches();
69     EspLogger::initialize_default();
70     log::info!("      RustHome v2.0 dimulai...");
71
72     // --- WiFi Setup ---
73     let peripherals = Peripherals::take().unwrap();
74     let sysloop = EspSystemEventLoop::take()?;
75     let nvs = EspDefaultNvsPartition::take().unwrap();
76     let mut wifi = EspWifi::new(peripherals.modem, sysloop.clone(),
77     Some(nvs.clone()))?;
78     let mut ssid: String<32> = String::new();
79     ssid.push_str("No Internet").unwrap();
80     let mut pass: String<64> = String::new();
81     pass.push_str("tertolong123").unwrap();
82     wifi.set_configuration(&Configuration::Client(ClientConfiguration { ssid,
83     password: pass, ..Default::default() }))?;
84     wifi.start()?;
85     wifi.connect()?;
86
87     while !wifi.is_connected().unwrap() {
88         log::info!("      Menunggu koneksi WiFi...");
89         thread::sleep(Duration::from_secs(1));
90     }
91     log::info!("      Terhubung ke WiFi!");
92
93     // --- MQTT Setup ---
94     let mqtt_config = MqttClientConfiguration {
95         client_id: Some("esp32-rust"),
96         username: Some("5UfJUCUW1M5ihKMBGua0"),
97         keep_alive_interval: Some(Duration::from_secs(30)),
98         ..Default::default()
99     };
100    let mqtt_connected = Arc::new(AtomicBool::new(false));
101    let mqtt_callback = {
102        let mqtt_connected = mqtt_connected.clone();
103        move |event: EspMqttEvent| match event.payload() {
104            EventPayload::Connected(_) => {
105                log::info!("      MQTT connected");
106            }
107        }
108    };
109 }
```

```
104         mqtt_connected.store(true, Ordering::SeqCst);
105     }
106     EventPayload::Received { topic, data, .. } => {
107         if let Ok(payload_str) = std::str::from_utf8(data) {
108             if let Some(t) = topic {
109                 if t.starts_with("v1/devices/me/rpc/request/") {
110                     let id = t.split('/').last().unwrap_or("0");
111                     if let Ok(json) =
112                         serde_json::from_str::<serde_json::Value>(payload_str) {
113                         if let Some(params) = json.get("params") {
114                             if let Some(url) =
115                                 params.get("ota_url").and_then(|u| u.as_str()) {
116                                 send_rpc_response(id, "success");
117                                 ota_process(url);
118                             }
119                         }
120                     }
121                 }
122             }
123             _ => {}
124         }
125     };
126
127 // MQTT connect loop
128 let client = loop {
129     match unsafe { EspMqttClient::new_nonstatic_cb(TB_MQTT_URL, &mqtt_config,
130     mqtt_callback.clone()) } {
131         Ok(c) => {
132             unsafe { MQTT_CLIENT = Some(c) };
133             if let Some(c_ref) = get_mqtt_client() {
134                 while !mqtt_connected.load(Ordering::SeqCst) {
135                     thread::sleep(Duration::from_millis(500));
136                 }
137                 c_ref.subscribe("v1/devices/me/rpc/request/+",
138 QoS::AtLeastOnce).unwrap();
139                 publish_fw_version();
140                 publish_fw_state("IDLE");
141                 break c_ref;
142             }
143         }
144     }
145 }
146
147 }
148 // --- Telemetry Loop ---
```

```

149     let mut rng = rand::thread_rng();
150     let mut counter = 0;
151     loop {
152         if counter >= 60 {
153             let temp = rng.gen_range(20.0..30.0);
154             let hum = rng.gen_range(50.0..70.0);
155             let payload = format!(r#"{{"temperature":{:.2}, "humidity":{:.2}}}"#, temp, hum);
156             client.publish("v1/devices/me/telemetry", QoS::AtLeastOnce, false,
157             payload.as_bytes()).ok();
158             counter = 0;
159         }
160         thread::sleep(Duration::from_secs(1));
161         counter += 1;
162     }
163
164 // --- OTA process ---
165 fn ota_process(url: &str) {
166     log::info!("OTA dari URL: {}", url);
167     publish_fw_state("DOWNLOADING");
168     match EspOta::new() {
169         Ok(mut ota) => {
170             let conn = EspMQTTConnection::new(&Default::default()).unwrap();
171             let mut client = Client::wrap(conn);
172             let mut response = client.get(url).unwrap().submit().unwrap();
173             let mut buf = [0u8; 1024];
174             let mut update = ota.initiate_update().unwrap();
175             loop {
176                 match response.read(&mut buf) {
177                     Ok(0) => break,
178                     Ok(size) => { update.write(&buf[..size]).unwrap(); }
179                     Err(_) => break,
180                 }
181             }
182             update.complete().unwrap();
183             publish_fw_state("SUCCESS");
184             unsafe { esp_restart(); }
185         }
186         Err(_) => publish_fw_state("FAILED"),
187     }
188 }

```

Listing A.3: Cuplikan kode utama sistem RustHome (main.rs)

## A.5 Cuplikan Kode Konfigurasi Cargo.toml

```
1 [package]
2 name = "dev"
3 version = "0.1.0"
4 authors = ["jidanrrey"]
5 edition = "2021"
6 resolver = "2"
7 rust-version = "1.77"
8
9 [[bin]]
10 name = "dev"
11 harness = false # Nonaktifkan test harness untuk menghindari
12 # error rust-analyzer
13
14 [profile.release]
15 opt-level = "s"
16
17 [profile.dev]
18 debug = true # Simbol debug tetap diaktifkan
19 opt-level = "z" # Optimasi ukuran
20
21 [features]
22 default = []
23 experimental = ["esp-idf-svc/experimental"]
24
25 [dependencies]
26 log = "0.4"
27 esp-idf-svc = "0.51"
28 embedded-svc = "0.28"
29 embedded-io = "0.6"
30 rand = "0.8"
31 anyhow = "1.0"
32 heapless = "0.8"
33 serde_json = "1.0"
34 dht-sensor = "0.2"
35 [build-dependencies]
```

```
36 | embuild = "0.33"  
37 |  
38 | [package.metadata.esp-idf]  
39 | partition_table = "partition_table.csv"
```

Listing A.4: Konfigurasi proyek pada berkas Cargo.toml

## A.6 Link Github

<https://github.com/Reyleo-create/Andre-Mahesa-Bagaskara—Teknologi-IoT.git>

Github berisikan main.rs, cargo.toml, prosedur project, dll digunakan agar lebih mudah diakses.

## Daftar Pustaka

1. D. Alhalabi, S. Alnakhiani, B. Fteiha, H. Zia (2024). *Enhancing Smart Home Automation: Secure AC Control and Environmental Monitoring with MQTT*.
2. G. Zhu, J. Xu, K. Huang, S. Cui (2021). *Over-the-Air Computing for Wireless Data Aggregation in Massive IoT*.
3. M. Meribout et al. (2022). *State of the Art IoT and Edge Embedded Systems for Real-Time Machine Vision Applications*.
4. J. Wan et al. (2016). *Software-Defined Industrial Internet of Things in the Context of Industry 4.0*.
5. Phala, Kumar, Hancke (2016). *Air Quality Monitoring System Based on ISO/IEC/I-EEE 21451 Standards*.
6. Annas et al. (2024). *Cloud-Based IoT System for Real-Time Harmful Algal Bloom Monitoring (ThingsBoard via MQTT and REST API)*.
7. Pfitzinger & Wöhrle (2023). *Embedded Real-Time Human Activity Recognition on ESP32-S3 Using Ambient Audio Data*.
8. Khalaf et al. (2021). *Controlling Smart Home Activities Using IoT*.
9. Kolahi et al. (2015). *Analysis of UDP DDoS Flood Cyber Attack and Defense Mechanisms on Web Server with Linux Ubuntu 13*.
10. Bello & Zeadally (2016). *Intelligent Device-to-Device Communication in the Internet of Things*.
11. Rahman et al. (2023). *IoT-Based Monitoring System for Indoor Air Quality Using ThingsBoard*.
12. Gao et al. (2023). *Rust for Linux: Understanding the Security Impact of Rust in the Linux Kernel*.

13. Li et al. (2024). *Scalable Smart Home Management with ESP32-S3: A Low-Cost Solution for Accessible Home Automation.*
14. Al-Fuqaha et al. (2023). *Sensor Data Collection and Analytics with ThingsBoard and Spark Streaming.*
15. Postolache et al. (2009). *Smart Sensors Network for Air Quality Monitoring Applications.*
16. Liu, Zang, Li, & Vucetic (2020). *Over-the-Air Computation Systems: Optimization, Analysis and Scaling Laws.*
17. Alabadi, Habbal, & Wei (2022). *Industrial Internet of Things: Requirements, Architecture, Challenges, and Future Research Directions.*
18. Ghosh et al. (2021). *Software-Defined Industrial Internet of Things in the Context of Industry 4.0.*
19. Liu et al. (2020). *Over-the-Air Computation Systems: Optimization, Analysis, and Scaling Laws (I).*
20. Montdher Alabadi et al. (2022). *Industrial Internet of Things: Requirement, Architecture, Challenges, and Future Research Directions.*
21. Zeng et al. (2021). *Federated Learning via Over-the-Air Computation.*