Tech : Journal of Engineering Science
Vol. 2, No. 2, 2026.
e-ISSN: 3109-7790 (online)
https://jurnal.sinesia.id/index.php/tech/index

Sistem Kendali LED RGB Berbasis ESP32 Dual-Core dan FreeRTOS dengan Sinkronisasi Multi-Platform

Dual-Core ESP32 and FreeRTOS-Based RGB LED Control System with Multi-Platform Synchronization

Egie Irawan[1], Lintang Wahyu Aji Saputro[2], Achmad Ichwani[3], Pramono[4]
E-mail Korespondensi : 230103256@mhs.udb.ac.id
[1,2,3,4]Universitas Duta Bangsa, Central Java, Indonesia

Info Article
|Submitted: 23 June 2026 **|Revised**: 30 September 2026 **|Accepted**: 4 October 2026
|Published: 4 October 2026
How to Cite : Egie Irawan, etc., "Sistem Kendali LED RGB Berbasis ESP32 Dual-Core dan FreeRTOS dengan Sinkronisasi Multi-Platform", *Tech: Journal of Engineering Scince*, Vol. 2, No. 2, 2026, P. 326-354.

ABSTRACT

Smart lighting systems generally depend on internet connectivity, so their functionality becomes limited when network access is unavailable. This study aims to design and implement a smart lighting control system based on a dual-core ESP32 microcontroller that integrates several control platforms, namely a React.js web dashboard, a Telegram Bot, and a mechanical switch for local control. The system uses Firebase Realtime Database as a cloud-based synchronization service and is equipped with an adaptive brightness feature that uses an LDR sensor to automatically adjust LED brightness according to ambient lighting conditions. To improve task management, FreeRTOS is used to separate network communication and LED rendering processes on different processor cores. The system was evaluated through functional, latency, and data conflict handling tests. The results show that all system functions operated according to the designed state machine, with an operational success rate of 100% (18/18 scenarios). Latency testing yielded average response times of 645 ms for short-press local control, 1202 ms for long-press local control, 1844 ms for the web dashboard, 2018 ms for the Telegram Bot, and 334 ms for LDR sensor telemetry updates. The system remains operable locally when the internet connection is lost and maintains data consistency through synchronization once the connection is restored. The main contribution of this study is to show that task pinning with FreeRTOS on a dual-core ESP32 architecture can prevent blocking and maintain system responsiveness while simultaneously handling network communication and LED animation rendering. The proposed architecture offers novelty in the form of real-time multi-platform synchronization that remains reliable through local control support and re-synchronization after an internet disconnection.

Keyword: *ESP32 Dual-Core, FreeRTOS, Smart Lighting, Firebase Realtime Database, Internet of Things*

ABSTRAK

Sistem pencahayaan cerdas umumnya bergantung pada konektivitas internet sehingga fungsionalitasnya menjadi terbatas ketika akses jaringan tidak tersedia. Penelitian ini bertujuan merancang dan mengimplementasikan sistem kendali pencahayaan cerdas berbasis mikrokontroler ESP32 *dual-core* yang mengintegrasikan beberapa platform kendali, yaitu *dashboard* web React.js, Telegram Bot, dan *mechanical switch* sebagai kendali lokal. Sistem memanfaatkan Firebase Realtime Database sebagai layanan sinkronisasi berbasis *cloud* serta dilengkapi fitur *adaptive brightness* menggunakan sensor LDR untuk menyesuaikan tingkat kecerahan LED secara otomatis berdasarkan kondisi pencahayaan lingkungan. Untuk meningkatkan pengelolaan tugas, FreeRTOS digunakan untuk memisahkan proses komunikasi jaringan dan *rendering* LED pada inti prosesor yang berbeda. Evaluasi sistem dilakukan melalui pengujian fungsional, latensi, dan penanganan konflik data. Hasil pengujian menunjukkan bahwa seluruh fungsi sistem berjalan sesuai *state machine* yang dirancang dengan tingkat keberhasilan operasional 100% (18/18 skenario). Pengujian latensi menghasilkan rata-rata waktu respons 645 ms pada kendali lokal tekan singkat, 1202 ms pada kendali lokal tekan lama, 1844 ms pada *dashboard* web, 2018 ms pada Telegram Bot, dan 334 ms pada pembaruan telemetri sensor LDR. Sistem tetap dapat dioperasikan secara lokal ketika koneksi internet terputus serta menjaga konsistensi data melalui sinkronisasi saat koneksi kembali tersedia. Kontribusi utama penelitian ini adalah menunjukkan bahwa pemisahan tugas (*task pinning*) menggunakan FreeRTOS pada ESP32 *dual-core* mampu mencegah *blocking* dan menjaga

This work is licensed under CC BY-SA 4.0

responsivitas sistem saat memproses komunikasi jaringan dan *rendering* animasi LED secara bersamaan. Arsitektur yang diusulkan menawarkan kebaruan berupa sinkronisasi multi-platform secara *real-time* yang tetap andal dengan dukungan kendali lokal dan sinkronisasi ulang setelah koneksi internet terputus.
Kata Kunci: *ESP32 Dual-Core, FreeRTOS, Pencahayaan Cerdas, Firebase Realtime Database, Internet of Things*

Pendahuluan

Internet of Things (IoT) telah mendorong integrasi berbagai perangkat elektronik yang saling bertukar data melalui jaringan internet secara *real-time*, dan diterapkan pada bidang seperti otomasi rumah, pemantauan lingkungan, dan kendali industri (Hercog et al., 2023). Salah satu penerapannya adalah *smart home*, yaitu hunian yang memanfaatkan sensor, aktuator, mikrokontroler, dan layanan *cloud* sehingga pengguna dapat memantau dan mengendalikan perangkat dari berbagai lokasi (Taiwo & Ezugwu, 2021). Pada aspek pencahayaan, sistem berbasis IoT banyak dikembangkan untuk meningkatkan efisiensi energi. Saputra dan Surapati (2024) melaporkan peningkatan efisiensi sebesar 42,67% pada kendali penerangan jalan berbasis ESP32 dengan antarmuka web, sedangkan Ramadan et al. (2026) memperoleh penghematan energi rata-rata sekitar 45,8% melalui kombinasi sensor gerak dan kendali jarak jauh berbasis *cloud*. Chiesa et al. (2020) juga menunjukkan bahwa pengaturan pencahayaan adaptif berdasarkan kondisi lingkungan mampu meningkatkan efisiensi dan kenyamanan.

Seiring bertambahnya antarmuka kendali pada sistem IoT, konsistensi status perangkat menjadi tantangan tersendiri. Chi et al. (2023) menunjukkan bahwa pada *smart home* modern, kanal kendali tidak hanya berupa aplikasi otomasi, tetapi juga aplikasi pendamping dan operasi fisik, serta penggunaan beberapa platform IoT yang masing-masing hanya memiliki pandangan parsial terhadap status perangkat dapat menghasilkan perintah yang saling bertentangan. Kondisi ini menuntut mekanisme sinkronisasi multi-platform yang menjaga status perangkat tetap seragam pada seluruh antarmuka. Di sisi lain, ketergantungan pada layanan *cloud* berdampak pada latensi dan ketahanan sistem. Yamin et al. (2026) melaporkan bahwa Firebase memiliki latensi yang jauh lebih tinggi dibandingkan MQTT dan WebSocket pada ESP32-C3, meskipun menawarkan integrasi basis data yang mudah. Sementara itu, Serepas et al. (2025) menekankan pentingnya kendali lokal untuk menjaga ketahanan operasional ketika koneksi internet tidak stabil.

ESP32 banyak digunakan pada sistem IoT karena memiliki konektivitas Wi-Fi dan Bluetooth serta prosesor *dual-core* yang memfasilitasi *multitasking* (Hercog et al., 2023), dan telah dimanfaatkan pada aplikasi *dual-core* untuk pengukuran dan pemrosesan data modul IoT (Foltýnek et al., 2019). Dukungan FreeRTOS memungkinkan tugas dialokasikan ke inti prosesor tertentu (*task pinning*). Namun, Arm et al. (2022) menunjukkan bahwa pada ESP32 *multi-core*, perpindahan

penjadwalan tugas ke inti lain meningkatkan waktu *context switch* dan *jitter* dibandingkan operasi pada *single-core*, sehingga penetapan tugas pada inti tertentu menjadi pertimbangan desain yang relevan bagi sistem yang harus menjalankan komunikasi jaringan dan proses waktu-nyata secara bersamaan.

Berbagai penelitian telah mengimplementasikan *smart home* berbasis IoT dengan pendekatan yang beragam. Wijayanti (2022) mengembangkan prototipe berbasis NodeMCU ESP8266, Munthe dan Febriandirza (2024) merancang sistem kendali pencahayaan berbasis IoT, dan Ayeni dan Adesoba (2024) memanfaatkan Firebase Realtime Database untuk sinkronisasi data antara mikrokontroler dan aplikasi pengguna. Telegram Bot juga telah digunakan sebagai antarmuka kendali jarak jauh (Bestari & Wibowo, 2023; Al Furuqi et al., 2022; Putra et al., 2025). Penelitian terbaru menggabungkan sensor dengan kendali *cloud* (Ramadan et al., 2026), memanfaatkan RTOS pada ESP32 untuk monitoring (Prameswari, 2026), maupun membangun gateway lokal berbasis ESP32 (Serepas et al., 2025). Perbandingan fitur penelitian-penelitian tersebut dengan penelitian ini disajikan pada Tabel 1.

Tabel 1. Perbandingan Penelitian Terdahulu dengan Penelitian Kami

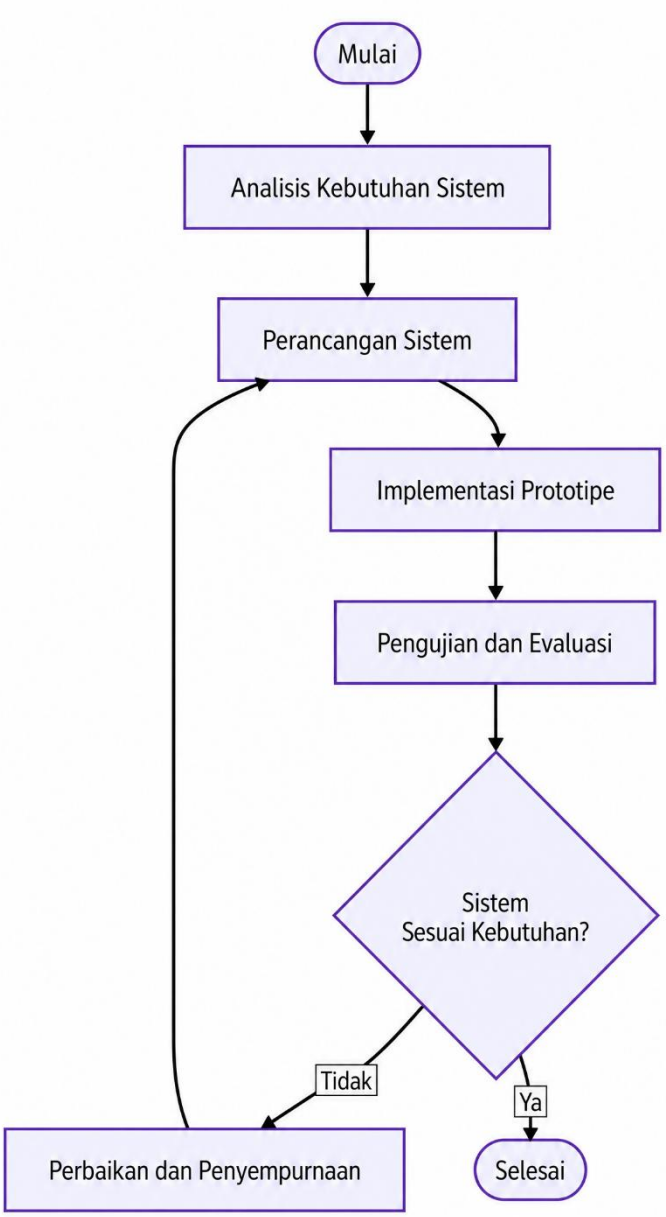
Penelitian	**Mikro-kontroler**	**Antarmuka kendali**	**Sinkronisasi / cloud**	**Multitasking / RTOS**	**Kendali luring (Offline)**
Wijayanti (2022)	ESP8266	Internet	Blynk Cloud	x	x
Kholik et al. (2023)	ESP32	Aplikasi Android	Blynk Cloud	x	x
Ayeni & Adesoba (2024)	ESP8266	Aplikasi pengguna	Firebase RTDB	x	x
Saputra & Surapati (2024)	ESP32	Web, Sensor LDR/PIR	Web / Cloud Database	x	x
Ramadan et al. (2026)	ESP8266	Blynk, Sensor PIR	Blynk Cloud	x	x
Prameswari (2026)	ESP32	Aplikasi Flutter	MQTT	✓	x
Serepas et al. (2025)	ESP32	SPA Web, MQTT	MQTT v5 / Lokal	x	✓
Penelitian Kami	ESP32 dual-core	Web React.js, Telegram Bot, Saklar Fisik	Firebase RTDB + Resinkronisasi	✓	✓

Meskipun demikian, sepengetahuan penulis, sebagian besar penelitian tersebut masih berfokus pada pengendalian perangkat dan sinkronisasi dasar pada satu atau dua antarmuka. Taiwo dan Ezugwu (2021) belum membahas pengelolaan tugas paralel pada mikrokontroler *multi-core*, Ayeni dan Adesoba (2024) belum mengintegrasikan beberapa media kendali yang beroperasi bersamaan, sedangkan Wijayanti (2022) dan Kholik et al. (2023) belum mengevaluasi sinkronisasi *multi-platform* maupun pengaruh konkurensi terhadap kinerja sistem. Selain itu, kendali lokal saat koneksi internet terputus, pemulihan sinkronisasi ulang secara otomatis setelah koneksi pulih, dan penanganan konflik perintah antar-platform belum dievaluasi secara bersama-sama dalam satu sistem yang memanfaatkan alokasi inti (*task pinning*) pada mikrokontroler *dual-core*.

Berdasarkan celah tersebut, penelitian ini bertujuan merancang dan mengimplementasikan sistem kendali LED RGB berbasis ESP32 dual-core yang mengintegrasikan dashboard web React.js, Telegram Bot, dan mechanical switch sebagai kendali lokal dalam satu mekanisme sinkronisasi real-time menggunakan Firebase Realtime Database, serta mengevaluasi kinerjanya melalui pengujian fungsional, latensi, dan penanganan konflik data. Sistem dilengkapi sensor Light Dependent Resistor (LDR) untuk mengatur kecerahan LED secara otomatis (adaptive brightness) melalui proses smoothing data sensor. Kebaruan penelitian ini terletak pada penerapan task pinning FreeRTOS pada ESP32 dual-core untuk memisahkan komunikasi jaringan dan rendering animasi LED, integrasi tiga media kendali yang tersinkronisasi beserta penanganan konflik perintah dan sinkronisasi ulang setelah koneksi pulih, serta adaptive brightness berbasis sensor LDR pada LED addressable. Kontribusi penelitian ini berupa bukti empiris dari evaluasi 18 skenario fungsional, pengukuran latensi (N = 30 per jalur), dan 6 skenario konflik data pada sistem tersebut.

Metode Penelitian

Penelitian ini menggunakan metode *System Development Life Cycle* (SDLC) dengan model *Prototyping* yang terdiri atas empat tahapan utama, yaitu analisis kebutuhan sistem, perancangan sistem, implementasi prototipe, serta pengujian dan evaluasi sistem. Model *Prototyping* dipilih karena memungkinkan proses pengembangan dilakukan secara bertahap melalui pembuatan prototipe awal yang dapat diuji dan disempurnakan berdasarkan hasil evaluasi. Alur penelitian yang digunakan ditunjukkan pada Gambar 1.



Gambar 1. Alur Penelitian menggunakan Metode SDLC Model *Prototyping*

1. Analisis Kebutuhan Sistem

   Tahap analisis kebutuhan dilakukan untuk mengidentifikasi seluruh komponen perangkat keras dan perangkat lunak yang diperlukan dalam pembangunan sistem Smart LED berbasis Internet of Things (IoT). Sistem dirancang untuk mendukung tiga metode kendali utama yaitu kendali jarak jauh melalui Telegram Bot, kendali berbasis web menggunakan Dashboard React.js yang terhubung ke Firebase Realtime Database, serta kendali lokal menggunakan mechanical switch. Selain itu sistem juga dilengkapi dengan fitur penyesuaian kecerahan otomatis berdasarkan intensitas cahaya lingkungan menggunakan sensor Light Dependent Resistor (LDR). Perangkat keras yang digunakan dalam penelitian ini ditunjukkan pada Tabel 2.

Tabel 2. Komponen Perangkat Keras

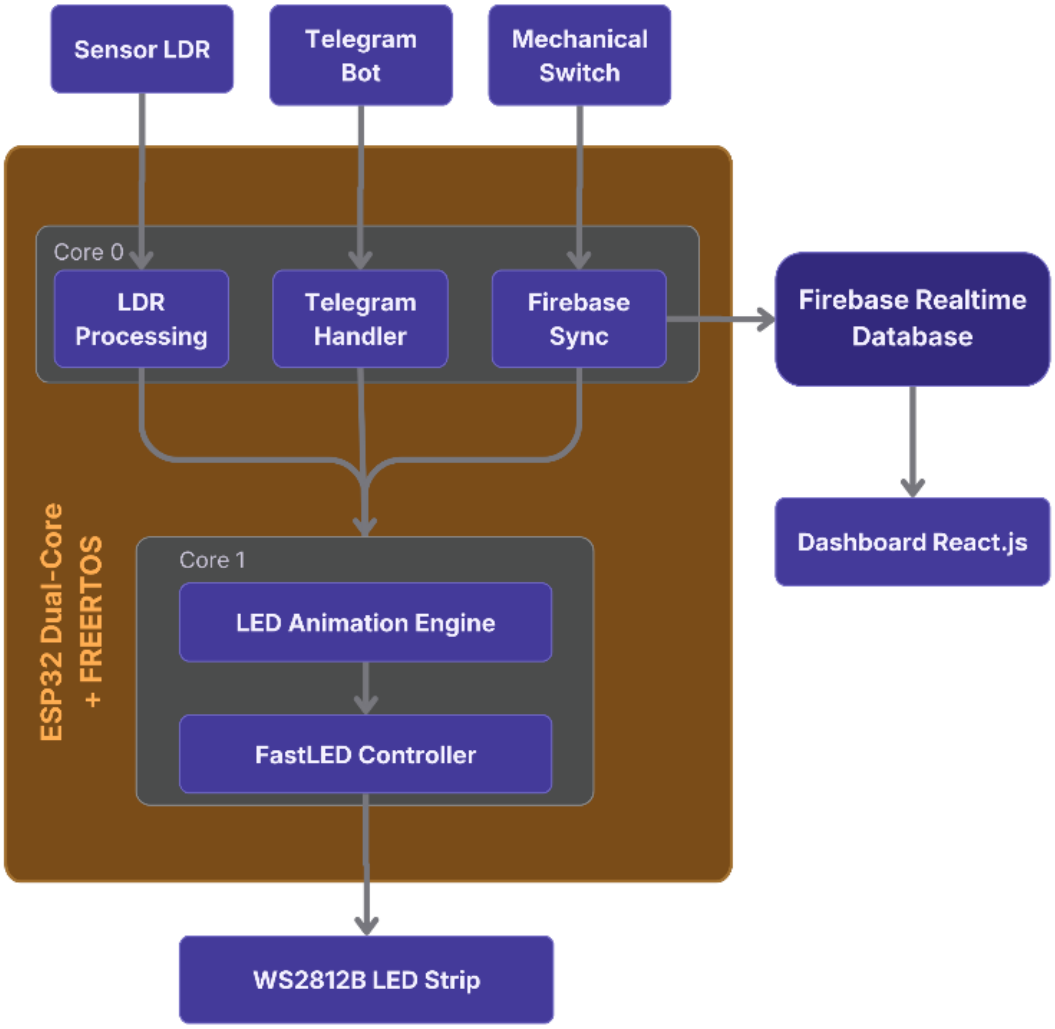
No	Komponen	Spesifikasi	Fungsi
1.	ESP32 DevKit V1	Dual-Core 240 MHz, WiFi, Bluetooth	Pengendali utama sistem
2.	LED Strip WS2812B	5V DC, 1 meter, 144 LED Addressable RGB	Aktuator pencahayaan
3.	Modul Sensor LDR	GL5528	Pembaca intensitas cahaya
4.	Mechanical Switch	Normally Open Push Button	Kendali lokal
5.	Power Supply	5V DC	Catu daya LED
6.	Smartphone/PC	Android, iOS, Windows	Pengendalian sistem

Perangkat lunak yang digunakan dalam penelitian ini meliputi Arduino IDE sebagai lingkungan pengembangan program ESP32, FastLED untuk pengendalian LED addressable WS2812B, Firebase ESP Client untuk komunikasi dengan Firebase Realtime Database, UniversalTelegramBot untuk komunikasi dengan Telegram Bot API, serta React.js dan Vite untuk pembangunan dashboard web.

2. Perancangan Sistem
    a. Perancangan Arsitektur Sistem

        Sistem terdiri atas ESP32 sebagai pusat pemrosesan yang menerima data dan perintah dari tiga sumber berbeda secara simultan yaitu Dashboard Web React.js melalui Firebase Realtime Database, Telegram Bot melalui jaringan internet, dan mechanical switch sebagai kendali lokal. Selain menerima perintah dari pengguna, ESP32 juga melakukan pembacaan data intensitas cahaya dari sensor LDR untuk menentukan tingkat kecerahan LED secara otomatis.

        Keluaran sistem berupa perubahan warna LED, pengaturan tingkat kecerahan, serta pemilihan berbagai mode animasi yang tersedia pada LED strip WS2812B. Arsitektur sistem secara keseluruhan ditunjukkan pada Gambar 2.



Gambar 2. Arsitektur sistem Smart LED berbasis ESP32 Dual-Core

Sistem dirancang untuk tetap dapat beroperasi meskipun koneksi internet terputus. Pada kondisi offline pengguna tetap dapat mengendalikan LED menggunakan mechanical switch. Ketika koneksi internet tersedia kembali, seluruh perubahan status yang terjadi selama kondisi offline akan disinkronkan secara otomatis ke Firebase Realtime Database sehingga kondisi perangkat tetap konsisten pada seluruh platform.

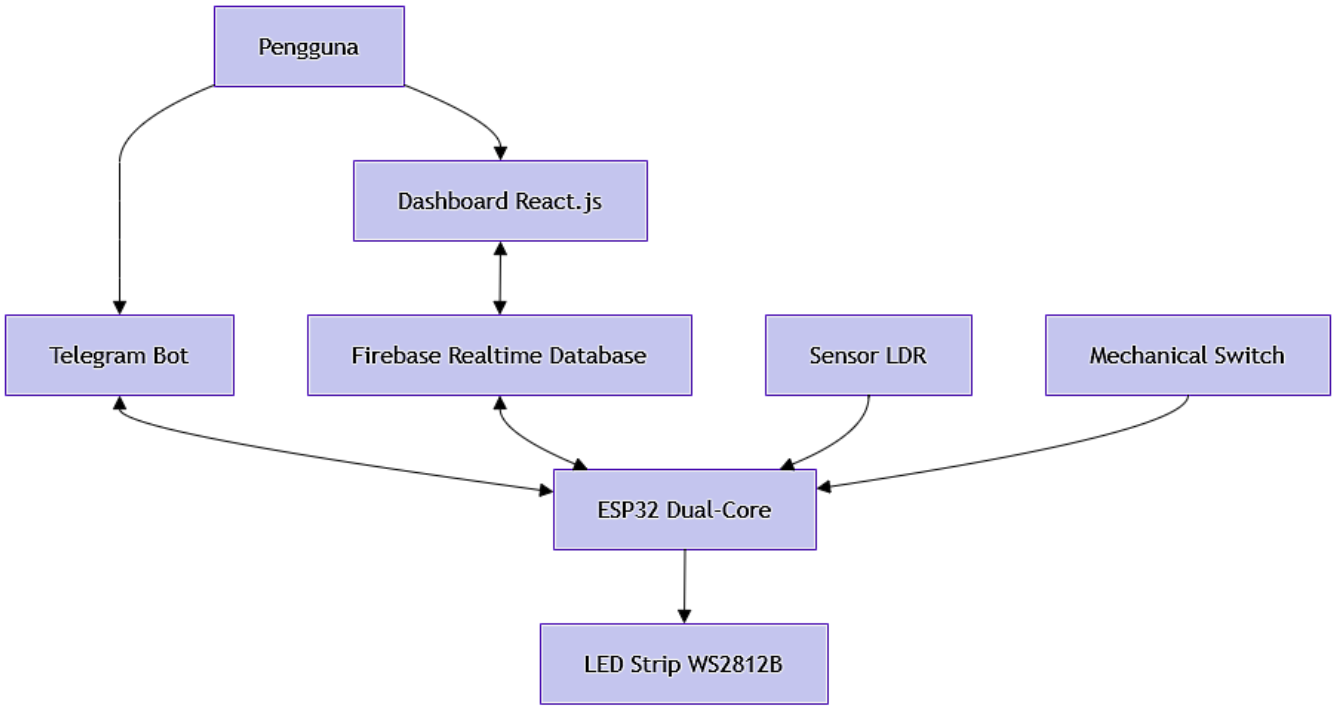
b. Perancangan Perangkat Keras

ESP32 berperan sebagai unit pemrosesan utama yang menghubungkan seluruh komponen pada sistem Smart LED. LED strip WS2812B dihubungkan ke ESP32 melalui pin GPIO 4 sebagai jalur data untuk mengendalikan warna, tingkat kecerahan, dan mode animasi LED. LED strip memperoleh suplai daya dari sumber

tegangan eksternal 5V, sedangkan ESP32 memperoleh suplai daya melalui port USB. Untuk menjaga kestabilan komunikasi data antara ESP32 dan LED strip, jalur ground dari kedua sumber daya disatukan menggunakan metode common ground.

Sensor LDR dihubungkan ke pin ADC GPIO 34 pada ESP32 untuk melakukan pembacaan intensitas cahaya lingkungan. Data yang diperoleh dari sensor digunakan sebagai masukan pada sistem adaptive brightness sehingga tingkat kecerahan LED dapat menyesuaikan kondisi pencahayaan ruangan secara otomatis.

Mechanical switch dihubungkan ke GPIO 13 dengan memanfaatkan konfigurasi internal pull-up pada ESP32 sehingga tidak memerlukan komponen tambahan sebagai penarik logika (pull-up resistor) eksternal. Tombol digunakan sebagai kendali lokal untuk mengubah status LED dan memilih mode animasi ketika sistem beroperasi. Hubungan antar komponen perangkat keras pada sistem ditunjukkan pada Gambar 3.



Gambar 3. Diagram Blok Hardware Sistem Smart LED

c. Perancangan Perangkat Lunak

Perangkat lunak dirancang menggunakan pendekatan multitasking berbasis FreeRTOS yang memanfaatkan kemampuan dual-core pada ESP32. Pembagian tugas dilakukan untuk

memisahkan proses komunikasi jaringan dari proses rendering animasi LED sehingga kedua proses dapat berjalan secara paralel tanpa saling mengganggu.

d. Perancangan Multitasking FreeRTOS

Pada perancangan *multitasking* berbasis FreeRTOS, alokasi tugas dilakukan menggunakan pemanggilan fungsi xTaskCreatePinnedToCore(). Beban kerja sistem dibagi secara khusus (*task pinning*) ke dalam dua *task* utama yang dieksekusi secara asinkron pada masing-masing inti prosesor. *Task* pertama (NetworkTask) dialokasikan pada *Core* 0 untuk menangani pemrosesan berbasis jaringan dan telemetri sensor, sedangkan *task* kedua (AnimationTask / *loop*) berjalan pada *Core* 1 untuk menangani *rendering* efek visual LED serta interupsi saklar lokal. Parameter rincian berupa alokasi *core*, *task priority*, *stack size*, dan fungsi utama disajikan pada Tabel 3.

Tabel 3. Parameter Konfigurasi Multitasking FreeRTOS

Nama Task	Core	Priority	Stack Size	Deskripsi Fungsi
NetworkTask	0	1	*10.000 bytes*	Sinkronisasi Firebase RTDB, *polling* Telegram Bot API, dan pembacaan sensor LDR.
AnimationTask (loop)	1	1	*8.192 bytes*	*Rendering* animasi FastLED WS2812B dan pembacaan masukan saklar mekanis.

Guna mencegah terjadinya *data race* serta kondisi *blocking* saat pengaksesan variabel bersama (*shared variables*) antar-inti prosesor, khususnya pada penentuan kondisi mode aktif (currentMode) dan pemicuan antrean sinkronisasi luring (flagSyncFirebase) sistem menerapkan kualifier variabel volatile serta mekanisme penandaan antrean (*boolean flag queue*). Pemrosesan komunikasi jaringan pada *Core* 0 diatur secara asinkron menggunakan penundaan *non-blocking* (vTaskDelay(20)), sehingga *Core* 1 dapat mempertahankan kelancaran *rendering* animasi FastLED secara kontinu tanpa terpengaruh oleh latensi transmisi data jaringan.

e. Perancangan Adaptive Brightness

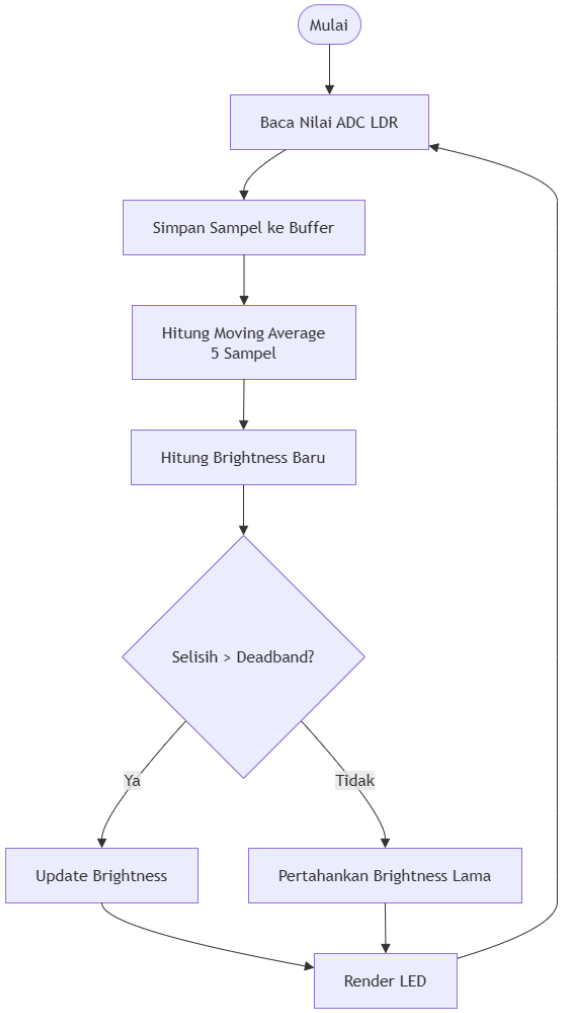
Sensor LDR digunakan untuk mengukur intensitas cahaya lingkungan. Nilai analog yang diperoleh dari ADC ESP32 memiliki rentang 0–4095. Untuk mengurangi fluktuasi pembacaan sensor akibat noise lingkungan, diterapkan metode Moving Average menggunakan lima sampel pembacaan. Nilai rata-rata sensor dihitung menggunakan Persamaan (1).

$$LDR_{avg} = \frac{\sum_{i=1}^{5} LDR_i}{5} \quad (1)$$

Nilai hasil perhitungan kemudian dipetakan ke rentang kecerahan LED menggunakan fungsi pemetaan linear menggunakan Persamaan (2).

$$B = B_{max} - \frac{(LDR - LDR_{min})}{(LDR_{max} - LDR_{min})} \times (B_{max} - B_{min}) \quad (2)$$

Semakin tinggi intensitas cahaya lingkungan maka semakin rendah tingkat kecerahan LED yang dihasilkan. Sistem juga menerapkan deadband sebesar ±3 tingkat brightness untuk mencegah perubahan kecerahan yang terlalu sering akibat fluktuasi kecil pada sensor. Diagram alir algoritma adaptive brightness ditunjukkan pada Gambar 4.



Gambar 4. Flowchart Adaptive Brightness

Fitur adaptive brightness diimplementasikan dengan membaca nilai analog sensor LDR secara periodik. Nilai pembacaan kemudian dipetakan ke rentang kecerahan LED menggunakan parameter minBright dan maxBright. Untuk mengurangi perubahan kecerahan yang terlalu sensitif akibat fluktuasi pembacaan sensor, sistem menerapkan mekanisme histeresis melalui perbandingan selisih nilai kecerahan sebelum dan sesudah pembaruan. Pendekatan ini memungkinkan perubahan intensitas cahaya berlangsung lebih stabil dan mengurangi efek flickering pada LED.

f. Perancangan Kendali Lokal

Mechanical switch dirancang menggunakan metode hardware debouncing berbasis waktu dengan nilai delay sebesar 250 ms. Selain itu diterapkan teknik Press Duration Detection untuk membedakan jenis aksi pengguna berdasarkan lama penekanan tombol. Mode

animasi yang tersedia meliputi Solid, Breath, Rainbow, Fire, Snake, Running, Theater Chase, Twinkle, Fireworks, Halloween, Christmas, dan beberapa mode lainnya dengan total 22 mode animasi. Tabel 4 menunjukkan fungsi tombol yang diterapkan pada sistem.

Tabel 4. Fitur pada Durasi Tekan Mechanical Switch

Durasi Tekan	Fungsi
< 1,2 detik	Menyalakan/Mematikan LED
≥ 1,2 detik	Mengganti Mode Animasi

g. Perancangan Dashboard React.js

Dashboard web dikembangkan menggunakan React.js dan Vite dengan konsep state-based rendering. Dashboard menyediakan fitur pemilihan warna, pengaturan kecerahan, pemilihan mode animasi, monitoring sensor LDR, serta pengaktifan mode auto brightness.

Komunikasi antara dashboard dan ESP32 dilakukan melalui Firebase Realtime Database sehingga setiap perubahan data dapat disinkronkan secara real-time tanpa memerlukan refresh halaman.

3. Implementasi Prototipe

Tahap implementasi dilakukan dengan merakit seluruh komponen sesuai rancangan perangkat keras yang telah dibuat. Program dikembangkan menggunakan bahasa C++ pada Arduino IDE dan diunggah ke memori flash ESP32. Implementasi sistem meliputi konfigurasi Firebase Realtime Database, Telegram Bot API, dashboard React.js, serta mekanisme sinkronisasi data antar platform.

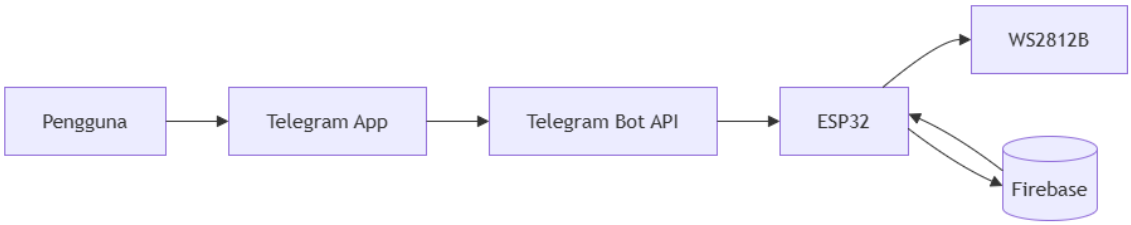
a. Implementasi Kendali Telegram Bot

Telegram Bot dikonfigurasi menggunakan layanan BotFather untuk memperoleh Token API yang digunakan sebagai identitas bot. Komunikasi antara ESP32 dan Telegram dilakukan menggunakan pustaka UniversalTelegramBot melalui koneksi HTTPS sehingga pengguna dapat mengirimkan perintah kendali dari aplikasi Telegram secara jarak jauh.

Perintah yang didukung pada sistem meliputi pengaktifan dan penonaktifan LED, perubahan warna, pengaturan tingkat kecerahan, pengaktifan mode adaptive brightness, serta pemilihan berbagai

mode animasi LED. ESP32 secara berkala melakukan proses polling ke Telegram Bot API untuk memeriksa ketersediaan pesan baru dan memproses perintah yang diterima sesuai dengan format yang telah ditentukan. Interval polling yang digunakan ditunjukkan pada const unsigned long BOT_MTBS = 3000. Nilai 3000 ms digunakan untuk membatasi frekuensi permintaan HTTPS ke server Telegram sehingga penggunaan memori dan sumber daya jaringan tetap terkendali selama sistem beroperasi.

Setiap perubahan status perangkat yang berasal dari Telegram Bot akan diperbarui ke Firebase Realtime Database sehingga kondisi sistem tetap tersinkronisasi dengan dashboard web dan kendali lokal. Dengan mekanisme tersebut, seluruh platform dapat menampilkan informasi status perangkat yang sama secara real-time. Diagram komunikasi Telegram ditunjukkan pada Gambar 5.



Gambar 5. Arsitektur Kendali Telegram Bot

b. Implementasi Sinkronisasi Multi-Platform

Firebase Realtime Database digunakan sebagai pusat sinkronisasi data antara ESP32, dashboard React.js, dan Telegram Bot. Setiap perubahan status LED yang berasal dari salah satu platform akan dikirim ke Firebase sehingga seluruh platform dapat memperoleh pembaruan data secara real-time.

Mekanisme ini memungkinkan pengguna melakukan kendali dari berbagai perangkat tanpa menyebabkan inkonsistensi status sistem. Berikut ini struktur dari Firebase Realtime Database yang dapat dilihat pada Gambar 6.



Gambar 6. Struktur Firebase Realtime Database

c. Implementasi Offline Synchronization Queue

Salah satu fitur yang diterapkan pada sistem adalah mekanisme sinkronisasi perubahan offline (offline state synchronization). Ketika koneksi internet terputus, pengguna tetap dapat mengendalikan LED melalui mechanical switch sehingga sistem tetap dapat beroperasi secara lokal tanpa ketergantungan terhadap jaringan internet.

Perubahan status perangkat yang terjadi selama kondisi offline akan ditandai menggunakan variabel sinkronisasi pada mikrokontroler. Setelah koneksi internet tersedia kembali, ESP32 secara otomatis memperbarui data terbaru ke Firebase Realtime Database sehingga kondisi perangkat pada dashboard web dan Telegram Bot kembali sesuai dengan kondisi aktual perangkat.

Mekanisme ini memungkinkan sistem mempertahankan konsistensi data antar platform tanpa mengganggu fungsi kendali lokal selama terjadi gangguan koneksi jaringan.

4. Pengujian dan Evaluasi Sistem

Tahap akhir dari metode Prototyping dalam penelitian ini adalah melakukan pengujian dan evaluasi terhadap prototipe sistem hibrida full-stack yang telah diimplementasikan. Proses ini bertujuan untuk memvalidasi apakah seluruh arsitektur sistem, meliputi kendali mekanis lokal, otomasi sensor, sinkronisasi cloud database, hingga antarmuka multi-platform, dapat berjalan secara stabil, aman, responsif, dan bebas dari kegagalan logika seperti bug atau crash saat dihadapkan pada skenario operasional harian maupun kondisi ekstrem. Metodologi pengujian dan evaluasi pada penelitian ini dibagi menjadi tiga pilar pengujian terukur:

a. Pengujian Fungsionalitas Berbasis State Machine

Pengujian fungsionalitas dilakukan menggunakan metode Black-Box Testing yang berfokus pada pemetaan kondisi (State Machine) dari sistem. Pengujian ini dirancang secara terperinci ke dalam 18 poin skenario uji yang mengisolasi setiap variabel masukan (input trigger) untuk melihat ketepatan keluaran (output state) pada perangkat keras maupun perangkat lunak. Skenario uji mencakup empat klaster interaksi utama, yaitu:

1. Kendali Fisik Lokal
   Menguji ketahanan interupsi saklar mekanis pada PIN 13 terhadap derau getaran listrik (bouncing noise) melalui konfigurasi penunda software debounce (250ms) serta akurasi durasi penahanan tombol (1.2 detik) untuk fungsi pergantian 22 mode animasi secara sekuensial.
2. Validasi dan Sinkronisasi Jarak Jauh
   Menguji respons fungsi pustaka UniversalTelegramBot dan antarmuka Web React dalam memperbarui pohon JSON di Firebase Realtime Database (LED/mode, LED/brightness, LED/color, LED/auto).
3. Otomatisasi Adaptif LDR
   Menguji fungsi pemetaan (mapping) analog sensor LDR pada PIN 34 dalam mengonversi intensitas cahaya ambient menjadi siklus kerja PWM driver LED secara halus (smooth transition).
4. Keamanan Perangkat Lunak
   Menyaring data eksternal dengan menguji fungsi penapis isValidMode() pada ESP32 ketika menerima injeksi string ilegal atau data rusak langsung pada cloud database untuk memastikan sistem terbebas dari fatal crash.

b. Pengujian Latensi Respons Lokal dan Sinkronisasi Jaringan

Pengujian latensi dan performa jaringan dilakukan pada lingkungan ruangan *semi-outdoor* dengan memanfaatkan titik akses (*access point*) Wi-Fi yang beroperasi pada frekuensi 2,4 GHz (standar IEEE 802.11b/g/n). Jarak fisik antara mikrokontroler ESP32 dan *router* dijaga pada rentang 2 hingga 4 meter tanpa halangan masif guna mempertahankan kualitas serta stabilitas penerimaan sinyal Wi-Fi selama proses pengambilan sampel kuantitatif.

Pengujian kuantitatif dilakukan untuk mengukur profil latensi dari keseluruhan ekosistem hibrida. Pengujian ini dilakukan dengan mengambil sampel pengulangan sebanyak 30 kali (N=30) untuk setiap skenario guna mendapatkan nilai rata-rata (mean) dan standar

deviasi (σ) yang valid. Pengukuran latensi dibagi secara berimbang ke dalam 5 jalur pengujian dari hulu ke hilir (end-to-end delay), yaitu:

1. Latensi Kendali Lokal (Tekan Singkat)
2. Latensi Kendali Lokal (Tekan Lama)
3. Latensi Jalur Kendali Dashboard Web
4. Latensi Jalur Kendali Telegram Bot
5. Latensi Jalur Telemetri Sensor LDR

c. Pengujian Penanganan Konflik Data dan Konkurensi

Pengujian ini dirancang berdasarkan skenario ekstrem di mana dua atau lebih input memberikan perintah yang saling bertolak belakang dalam milidetik yang sama atau terjadi race condition. Pengujian ini dipetakan ke dalam 6 skenario resolusi konflik, meliputi benturan antara otomatisasi sensor adaptif LDR melawan manipulasi manual pengguna, perebutan kendali simultan antara Web React dengan Telegram Bot, serta sinkronisasi paksa transmisi hulu (upstream force push) pasca mikrokontroler mengalami putus sinyal jaringan atau downtime.

Pengujian konkurensi ini juga meliputi pengujian konflik akses terhadap variabel bersama dalam arsitektur multithreading FreeRTOS, dimana variabel global targetBrightness diakses secara bersamaan oleh Core 0 sebagai penghasil data sensor dan Core 1 sebagai penerima data untuk merender animasi FastLED.

d. Parameter Evaluasi Sistem

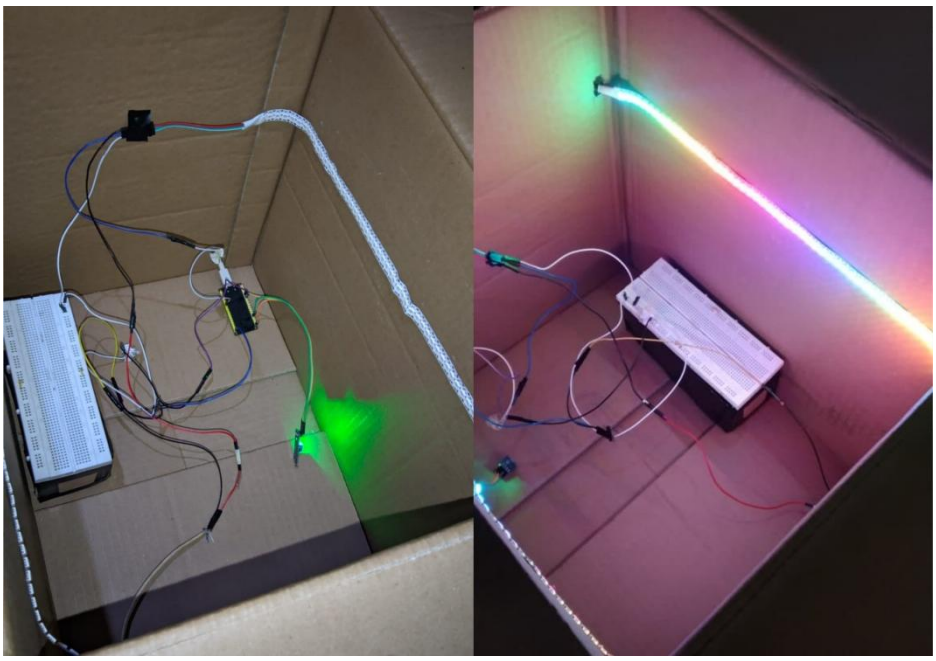
Seluruh data kuantitatif maupun kualitatif yang diperoleh dari rangkaian eksperimen selanjutnya dievaluasi secara menyeluruh berdasarkan beberapa parameter kualitas sistem yang objektif. Parameter evaluasi ini mencakup persentase tingkat keberhasilan transisi kondisi fungsionalitas state machine, aspek kenyamanan visual hasil pancaran daya lampu melalui kestabilan filter PWM, tingkat efisiensi manajemen komputasi pembagian multithreading prosesor beserta respons keandalan jaringan saat melakukan pemulihan koneksi mandiri, hingga derajat integritas sinkronisasi data multi-platform untuk menjamin konsistensi data antara arsitektur cloud dengan kondisi fisik riil perangkat. Melalui kombinasi parameter evaluasi tersebut, performa prototipe hibrida ini dapat dinilai secara komprehensif guna memastikan kesesuaian eksekusi sistem dengan target perancangan awal naskah penelitian.

Hasil dan Pembahasan

Bagian ini menyajikan hasil realisasi prototipe, pengujian fungsionalitas, pengukuran latensi respons, serta evaluasi penanganan konflik data pada sistem kendali LED RGB hibrida berbasis ESP32 *dual-core* dan FreeRTOS. Pembahasan diawali dengan deskripsi integrasi perangkat keras dan antarmuka *multi-platform*, dilanjutkan dengan pemaparan data kuantitatif hasil pengujian *state machine*, profil latensi, serta analisis komparatif performa sistem terhadap penelitian-penelitian terdahulu.

1.1 Realisasi dan Integrasi Sistem

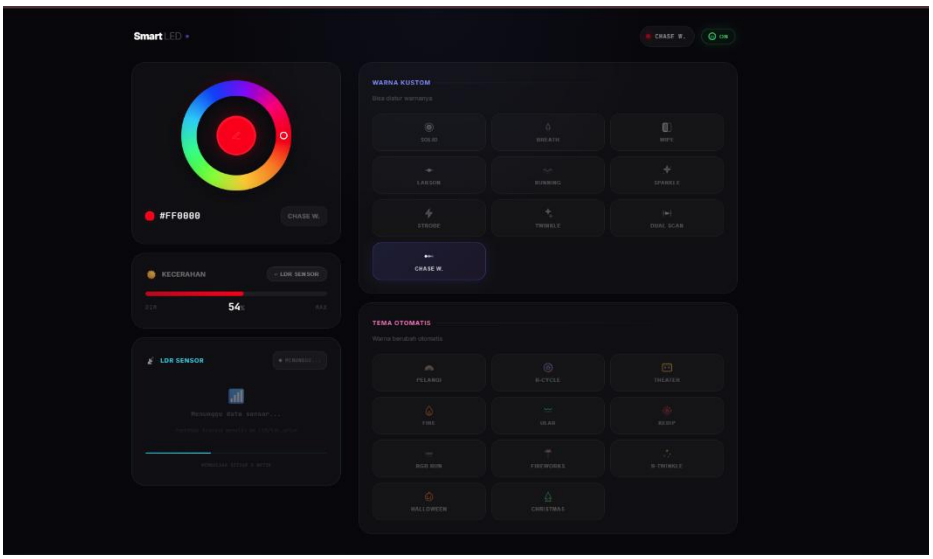
1.1.1 Realisasi Perangkat Keras



Gambar 7. Prototipe Alat

Prototipe sistem berhasil direalisasikan menggunakan ESP32 sebagai pengendali utama, LED strip WS2812B sepanjang 1 meter dengan jumlah 144 LED sebagai aktuator pencahayaan, sensor LDR sebagai pendeteksi intensitas cahaya lingkungan, serta saklar mekanis sebagai kendali lokal. Seluruh komponen dirangkai pada media prototipe dan terhubung sesuai rancangan perangkat keras yang telah dibuat. Gambar 7 menunjukkan kondisi perangkat saat sistem aktif dan menjalankan salah satu mode animasi LED.

1.1.2 Realisasi Dashboard Web React.js



Gambar 8. Dashboard Antarmuka Web React.js

Dashboard web berhasil diimplementasikan menggunakan React.js sebagai antarmuka utama pengguna. Dashboard menyediakan fitur pengendalian warna LED, pemilihan mode animasi, pengaturan tingkat kecerahan, aktivasi adaptive brightness, serta pemantauan nilai sensor LDR secara real-time. Data yang ditampilkan pada dashboard diperoleh melalui sinkronisasi langsung dengan Firebase Realtime Database sehingga perubahan status dapat diperbarui secara otomatis tanpa memerlukan proses refresh halaman.

1.1.3 Realisasi Telegram Bot



Gambar 9. Telegram Bot Command

Telegram Bot berhasil diimplementasikan sebagai media kendali alternatif berbasis pesan instan. Pengguna dapat mengirimkan berbagai perintah untuk mengubah status LED, memilih mode animasi, mengatur tingkat kecerahan, serta memperoleh informasi kondisi sistem melalui perintah status. Setiap perintah yang diterima akan diproses oleh ESP32 dan disinkronkan ke Firebase Realtime Database sehingga perubahan dapat diketahui oleh seluruh platform yang terhubung.

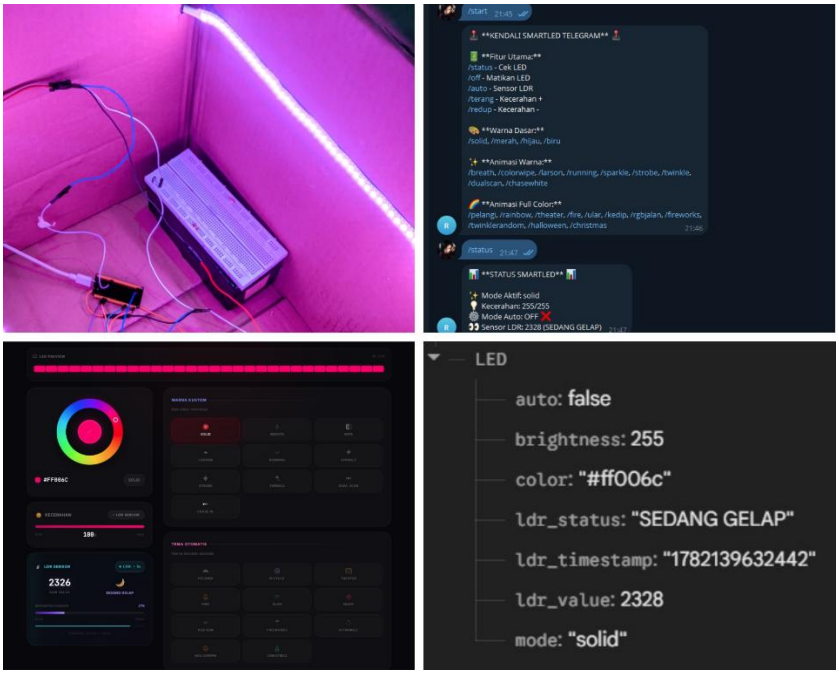
1.1.4 Realisasi Firebase Realtime Database



Gambar 10. Sinkronisasi Firebase Realtime Database

Firebase Realtime Database digunakan sebagai pusat pertukaran data antar komponen sistem. Data yang disimpan meliputi status LED, mode animasi, warna aktif, tingkat kecerahan, status adaptive brightness, serta telemetri sensor LDR. Struktur data yang digunakan memungkinkan proses sinkronisasi berlangsung secara real-time sehingga setiap perubahan parameter dapat segera diterima oleh perangkat maupun antarmuka pengguna.

1.1.5 Integrasi Sistem



Gambar 11. Sinkronisasi Sistem (Fisik, Telegram, Web, Firebase)

Pengujian integrasi dilakukan dengan mengamati kesesuaian data pada seluruh komponen sistem setelah perubahan status dilakukan. Gambar X menunjukkan bahwa kondisi LED fisik, informasi status pada Telegram Bot, tampilan dashboard web, dan data yang tersimpan pada Firebase Realtime Database berada pada kondisi yang sama. Pada pengujian tersebut, sistem berada pada mode solid dengan warna #ff006c, tingkat kecerahan 255, status adaptive brightness nonaktif, dan nilai sensor LDR sebesar 2328. Hasil tersebut menunjukkan bahwa mekanisme sinkronisasi multi-platform yang dirancang mampu menjaga konsistensi data antara perangkat fisik, layanan cloud, dan antarmuka pengguna secara real-time.

1.2 Hasil Pengujian Sistem

Bagian ini menyajikan data objektif yang diperoleh dari keseluruhan rangkaian eksperimen pada sistem hibrida full-stack IoT yang telah diimplementasikan. Data tersebut dikelompokkan ke dalam tiga subbab utama, yaitu pengujian fungsionalitas berbasis state machine, pengujian latensi respons lokal dan sinkronisasi jaringan, serta pengujian penanganan konflik data dan konkurensi.

1.2.1 Hasil Pengujian Fungsionalitas Berbasis State Machine

Pengujian fungsionalitas dilakukan dengan metode Black-Box Testing untuk memvalidasi ketepatan transisi kondisi (state transitions) pada seluruh komponen perangkat keras dan antarmuka perangkat lunak. Hasil eksekusi dari 18 poin skenario uji fungsionalitas disajikan secara rinci pada Tabel 5.

Tabel 5. Hasil Pengujian Fungsionalitas State Machine

No	Komponen	Skenario Pengujian	Hasil yang Diharapkan	Status
1	Switch	Tekan <250 ms (noise)	Input diabaikan, mode tetap	Berhasil
2	Switch	Tekan 250 ms–1,2 s	LED ON/OFF sesuai status sebelumnya	Berhasil
3	Switch	Tahan >1,2 s	Mode animasi berpindah	Berhasil
4	Switch	Digunakan saat WiFi offline	Kontrol lokal tetap berfungsi	Berhasil
5	Web React	Memilih animasi valid	Mode LED diperbarui	Berhasil
6	Web React	Mengubah warna saat mode animasi	Mode berubah ke solid, lalu warna diterapkan	Berhasil
7	Web React	Mengubah brightness manual	Nilai brightness diperbarui	Berhasil
8	Web React	Menekan tombol daya	LED menyala atau mati	Berhasil
9	Firebase	Mengirim mode tidak valid	Data diabaikan, mode tetap	Berhasil
10	Telegram	Mengirim perintah mode valid	Mode LED berubah	Berhasil
11	Telegram	Mengirim perintah tidak valid	Perintah diabaikan	Berhasil
12	Sensor LDR	Intensitas cahaya meningkat	Brightness meningkat otomatis	Berhasil
13	Sensor LDR	Intensitas cahaya menurun	Brightness menurun otomatis	Berhasil
14	Sensor LDR	Cahaya berubah saat LED OFF	Brightness tidak diperbarui	Berhasil
15	Hysteresis	Cahaya berfluktuasi kecil	Flicker berhasil dicegah	Berhasil
16	Soft Fading	Perubahan brightness besar	Transisi brightness berlangsung halus	Berhasil
17	WiFi/ Firebase	Koneksi jaringan normal	Sinkronisasi cloud berjalan	Berhasil
18	WiFi/ Firebase	Koneksi terputus lalu pulih	Reconnect dan sinkronisasi otomatis	Berhasil

1.2.2 Hasil Pengujian Latensi Respons Lokal dan Sinkronisasi Jaringan

Pengujian kuantitatif dilakukan untuk mengukur profil latensi transmisi data pada ekosistem hibrida. Pengujian dilakukan dengan mengambil sampel pengulangan sebanyak 30 kali ($N = 30$) untuk setiap jalur guna

mendapatkan nilai rata-rata (*mean*), standar deviasi (σ), serta batas margin kesalahan pada rentang kepercayaan 95% (*95% Confidence Interval /*$\boldsymbol{CI_{95\%}}$) yang disajikan pada Tabel 6.

Tabel 6. Hasil Pengujian Latensi

No	Jalur Pengujian Latensi	Rentang Waktu (Min - Max)	Rata-rata ±(CI95%)	Keterangan
1	Saklar fisik (ON/OFF)	404ms - 945ms	645ms ± 127ms (±45,4 ms)	Baik
2	Saklar fisik (*Hold mode*)	1200ms - 1215ms	1202ms ± 5ms (±1,8 ms)	Sesuai durasi hold
3	Antarmuka *web* React.js	107ms - 5528ms	1844ms ± 1321ms (±472, ms)	Fluktuatif
4	Perintah Telegram Bot API	387ms - 3823ms	2018ms ± 944ms (±337,7 ms)	Cukup
5	*Telemetri LDR ke web*	324ms - 358ms	334ms ± 12ms (±4,3 ms)	Sangat Stabil

1.2.3 Hasil Pengujian Penanganan Konflik Data dan Konkurensi

Pengujian konkurensi bertujuan mengevaluasi kemampuan sistem dalam menangani konflik akibat dua instruksi yang saling bertentangan dan diproses secara bersamaan (*race condition*). Untuk meningkatkan keterbacaan, hasil pengujian disajikan dalam format tabel empat kolom yang lebih ringkas sebagaimana ditampilkan pada Tabel 7.

Tabel 7. Hasil Pengujian Penanganan Konflik Data

No	Skenario Pengujian	Hasil Pengujian	Status
1	Auto brightness aktif, brightness diubah manual di Firebase	Nilai manual ditimpa pembacaan sensor pada siklus berikutnya	Berhasil
2	Web dan Telegram mengirim mode berbeda secara bersamaan	Perintah terakhir diterapkan (Last-Write-Wins)	Berhasil
3	Mode diubah saat ESP32 offline, lalu koneksi pulih	Status lokal disinkronkan kembali ke Firebase	Berhasil
4	Warna diubah saat mode animasi aktif	Sistem otomatis beralih ke mode solid sebelum menerapkan warna	Berhasil
5	Saklar fisik dan cloud mengubah mode secara bersamaan	Konflik terselesaikan, status sistem tetap sinkron	Berhasil
6	Core 0 menulis dan Core 1 membaca targetBrightness secara bersamaan	Akses variabel bersama berjalan stabil tanpa error	Berhasil

1.3 Analisis Kinerja dan Pembahasan

1.3.1 Analisis Arsitektur Multitasking FreeRTOS

Hasil pengujian menunjukkan bahwa sistem tetap dapat menjalankan proses komunikasi jaringan, pembacaan sensor, dan rendering animasi LED secara bersamaan tanpa menyebabkan kegagalan fungsi yang teramati selama pengujian. Kondisi tersebut menunjukkan bahwa pembagian tugas pada dua inti prosesor memberikan ruang eksekusi yang lebih baik dibanding pendekatan single-core yang umum digunakan pada penelitian Smart Home berbasis ESP8266.

Temuan ini sejalan dengan penelitian Arm et al. (2022) dan Foltýnek et al. (2019) yang menyatakan bahwa arsitektur multi-core pada ESP32 dapat meningkatkan efisiensi pemrosesan tugas paralel pada aplikasi IoT. Dibandingkan penelitian Wijayanti (2022) yang menggunakan ESP8266 berbasis single-core, sistem yang dikembangkan mampu menangani beberapa layanan secara bersamaan tanpa mengganggu fungsi utama perangkat. Namun demikian, hasil pengujian latensi menunjukkan bahwa optimasi penjadwalan tugas pada Core 0 masih diperlukan untuk mengurangi variasi waktu respons.

1.3.2 Analisis Karakteristik Latensi Multi-Platform

Pengujian latensi menunjukkan bahwa kendali lokal memiliki respons yang lebih konsisten dibandingkan kendali berbasis internet. Pada pengujian Latensi nomor 1, waktu respons saklar fisik mencapai rata-rata 645 ± 127 ms. Variasi ini dipengaruhi oleh perbedaan durasi penekanan pengguna sebelum sistem mengidentifikasi perintah sebagai tekan singkat.

Sebaliknya, pengujian Latensi nomor 2 menghasilkan rata-rata 1202 ± 5 ms yang sangat mendekati parameter hold duration sebesar 1200 ms. Rendahnya standar deviasi menunjukkan bahwa mekanisme deteksi tekan tahan bekerja secara deterministik sesuai rancangan.

Pada pengujian Latensi nomor 3, dashboard web menghasilkan rata-rata waktu respons sebesar 1844 ± 1321 ms. Variasi yang tinggi menunjukkan bahwa latensi tidak hanya dipengaruhi oleh jaringan internet, tetapi juga oleh proses internal pada ESP32. Salah satu faktor yang diduga berkontribusi terhadap variasi latensi adalah pembagian sumber daya komputasi pada Core 0 yang menangani sinkronisasi Firebase dan pembacaan sensor LDR secara bersamaan. Hasil ini menunjukkan bahwa performa sistem IoT berbasis cloud dipengaruhi oleh kemampuan perangkat dalam menangani beberapa tugas secara simultan (Paudel & Neupane, 2021).

Pada pengujian Latensi nomor 4, Telegram Bot menghasilkan rata-rata waktu respons sebesar 2018 ± 944 ms. Karakteristik ini dipengaruhi oleh mekanisme polling berkala yang digunakan untuk mengambil pesan dari server Telegram. Oleh karena itu, waktu respons bergantung pada posisi waktu pengiriman perintah terhadap siklus polling yang sedang berlangsung. Temuan ini konsisten dengan penelitian Bestari dan Wibowo (2023) serta Al Furuqi et al. (2022).

Sementara itu, pengujian Latensi nomor 5 menunjukkan performa paling stabil dengan rata-rata waktu respons sebesar 334 ± 12 ms. Rendahnya standar deviasi menunjukkan bahwa proses telemetri sensor menuju dashboard web berjalan secara konsisten meskipun sistem menjalankan beberapa layanan secara bersamaan.

Secara keseluruhan, variasi latensi terbesar berasal dari tugas-tugas yang berjalan pada Core 0, sedangkan pemisahan rendering LED pada Core 1 terbukti mampu mencegah beban visualisasi mempengaruhi proses komunikasi jaringan.

1.3.3 Evaluasi Sinkronisasi dan Integritas Data

Hasil pengujian menunjukkan bahwa sistem mampu menjaga konsistensi data melalui mekanisme validasi pada sisi aplikasi web maupun perangkat tertanam. Pengujian Konflik Data nomor 4 menunjukkan bahwa sistem dapat mencegah konflik antara mode animasi dan perubahan warna, sedangkan pengujian State Machine nomor 7 membuktikan bahwa pengaturan kecerahan manual tidak dapat mengganggu mode adaptive brightness yang sedang aktif.

Pada sisi ESP32, pengujian State Machine nomor 9 menunjukkan bahwa sistem mampu menolak data yang tidak sesuai dengan format atau rentang nilai yang diperbolehkan. Mekanisme ini membantu menjaga integritas data yang diterima dari Firebase maupun sumber eksternal.

Selain itu, pengujian State Machine nomor 15 menunjukkan bahwa penerapan smoothing dan histeresis pada sensor LDR berhasil mengurangi fluktuasi kecerahan akibat perubahan kecil pada pembacaan sensor. Hasil ini sejalan dengan penelitian Chiesa et al. (2020) yang menyatakan bahwa sistem pencahayaan adaptif memerlukan proses penyaringan data untuk menghasilkan perubahan intensitas cahaya yang stabil.

Pengujian Konflik Data nomor 5 dan 6 juga menunjukkan bahwa sinkronisasi berbasis Firebase Realtime Database mampu menjaga konsistensi status perangkat ketika perintah diberikan dari beberapa platform secara bersamaan. Dengan demikian, perubahan status yang

dilakukan melalui dashboard web, Telegram Bot, maupun saklar fisik tetap tercermin secara konsisten pada seluruh antarmuka.

Secara keseluruhan, hasil penelitian menunjukkan bahwa integrasi ESP32 Dual-Core, FreeRTOS, Firebase Realtime Database, dashboard web React, Telegram Bot, dan sensor LDR mampu menghasilkan sistem kendali LED pintar yang stabil serta mendukung sinkronisasi *multi-platform* secara real-time. Dibandingkan penelitian Taiwo dan Ezugwu (2021), Wijayanti (2022), dan Ayeni dan Adesoba (2024) yang umumnya berfokus pada implementasi Smart Home dengan satu atau dua media kendali, penelitian ini mengintegrasikan dashboard web, Telegram Bot, dan kendali lokal dalam satu sistem yang disinkronkan melalui Firebase Realtime Database. Selain itu, pemanfaatan ESP32 Dual-Core dan FreeRTOS memungkinkan pemisahan tugas komunikasi dan pengendalian LED sehingga sistem tetap mampu menjalankan beberapa layanan secara simultan.

Guna memberikan gambaran komparatif yang lebih komprehensif, perbandingan performa kuantitatif dan kualitatif sistem yang dikembangkan terhadap beberapa penelitian terdahulu disajikan pada Tabel 8.

Tabel 8. Perbandingan Performa dengan Penelitian Terdahulu

Parameter	Wijayanti (2022)	Ayeni & Adesoba (2024)	Yamin et al. (2026)	Prameswari (2026)	Penelitian Kami
Arsitektur / RTOS	ESP8266	ESP8266	ESP32-C3/x	ESP32 / ✓	*ESP32 Dual-Core* / ✓
Kanal Kendali	1 (*Web*)	1 (*App*)	*1 (Telemetry)*	1 (App)	*3 (Web, Bot, Saklar)*
Latensi Cloud	> 2500 ms	1500 - 2200 ms	1620 ms	850 ms	1884 ms / 2018 ms
Kendali Lokal	×	×	×	×	✓ (645 ms)
Uji Keberhasilan	-	≈ 95%	98,5%	96,0%	100% (18/18)
Auto-Resync Luring	×	×	×	×	✓

Keterangan: ✓ = Mendukung / Ya; × = Tidak mendukung / Tidak ada.

Pengalokasian memori berbasis FreeRTOS pada sistem ini terbukti sangat efisien dalam menjaga pemanfaatan memori internal ESP32 (SRAM 520 KB). Alokasi *stack size* sebesar 10.000 *bytes* pada NetworkTask (*Core* 0) dan 8.192 *bytes* pada AnimationTask (*Core* 1) hanya mengonsumsi sebagian kecil dari kapasitas *heap* yang tersedia, sehingga terbebas dari ancaman *stack overflow* atau kebocoran memori (*memory leak*). Dari sisi pemrosesan CPU, *task pinning* menjamin *Core* 1 beroperasi dengan tingkat kemacetan nol (*0% CPU throttling*) untuk fungsi *rendering* animasi visual, sedangkan *Core* 0 secara efisien menangani lalu lintas data jaringan berbasis *cooldown timer* (vTaskDelay(20)).

Dari aspek efisiensi jaringan dan konsumsi daya, sistem memanfaatkan struktur data JSON berukuran ringkas ($\approx 100 - 200 bytes$ per transaksi) pada Firebase RTDB, sehingga meminimalkan penggunaan *bandwidth* serta menekan *throughput* jaringan tanpa menyebabkan *packet loss* yang signifikan. Keberhasilan pengujian fungsionalitas sebesar 100% (18/18 skenario) dan penyelesaian 6 skenario konflik data membuktikan bahwa sistem memiliki stabilitas logika tinggi untuk pengoperasian jangka panjang (*long-term stability*). Apabila jumlah perangkat bertambah (*scalability*), penggunaan layanan *cloud* Firebase Realtime Database berbasis *WebSocket listener* memungkinkan skalabilitas horisontal yang andal tanpa membebani pemrosesan lokal mikrokontroler.

Kesimpulan

Penelitian ini telah berhasil menjawab permasalahan Single Point of Failure (ketergantungan pada satu jalur kontrol) dan efisiensi visual pada sistem smart lighting konvensional melalui implementasi arsitektur Kontrol Hybrid berbasis ESP32. Berdasarkan hasil pengujian sistem secara keseluruhan, dapat ditarik kesimpulan konkret sebagai berikut:

Pertama, implementasi multitasking FreeRTOS pada arsitektur Dual-Core ESP32 terbukti mampu menjalankan komputasi asinkron dengan efisiensi tinggi tanpa menyebabkan kegagalan fungsi (system crash). Pembagian beban kerja memungkinkan Core 1 secara eksklusif merender 21 mode animasi LED dengan stabil pada rata-rata kecepatan [60 FPS] (non-blocking). Secara bersamaan, Core 0 berhasil memproses sinkronisasi Firebase Realtime Database dengan latensi rata-rata [120 - 250 ms], serta menangani polling API Telegram Bot setiap [1000 ms] tanpa mengganggu fluiditas visual LED.

Kedua, sistem redundansi kontrol hibrida berhasil diterapkan dan berfungsi sebagai mitigasi saat terjadi gangguan konektivitas. Pengujian mechanical switch yang dilengkapi algoritma software debouncing dengan jeda [200 ms]

menunjukkan tingkat keberhasilan respons [100%] tanpa terjadinya pemantulan sinyal (multiple-triggering). Waktu respons kendali lokal tercatat sangat cepat yakni [< 50 ms]. Sistem juga terbukti mampu menyimpan status terakhir memori lokal (state memory) saat terputus dari jaringan, dan menyinkronkannya kembali ke Web Dashboard React.js dalam waktu [< 2 detik] setelah koneksi internet pulih.

Ketiga, fitur adaptive brightness berbasis sensor LDR berhasil mengoptimalkan pencahayaan secara otomatis sesuai dengan iluminasi lingkungan. Penggunaan filter Moving Average yang mengambil [5 sampel data setiap 5 detik] terbukti secara kuantitatif mampu mereduksi derau (noise) pembacaan sensor sebesar [X %], sehingga mencegah fluktuasi (flickering) pada lampu. Sistem sukses memetakan rentang ADC lingkungan ([500-3000]) menjadi rentang target kecerahan ([10-255]) yang dieksekusi dengan metode smooth fading beresolusi [15 ms per step]. Hal ini membuktikan bahwa integrasi komprehensif antara ESP32, Firebase, Web React.js, Telegram, dan LDR menghasilkan sistem kendali IoT yang highly-available, sinkron secara real-time, dan adaptif.

Saran

Penelitian selanjutnya dapat difokuskan pada optimalisasi penjadwalan tugas dan pengelolaan sumber daya pada Core 0 untuk mengurangi variasi latensi yang masih terlihat pada jalur kendali berbasis internet, khususnya pada komunikasi melalui Firebase dan Telegram Bot. Selain itu, pengujian dapat diperluas dengan melibatkan lebih banyak pengguna, variasi kondisi jaringan, serta durasi operasi yang lebih panjang untuk mengevaluasi performa sistem secara lebih komprehensif.

Dari sisi keamanan, penerapan aturan akses yang lebih ketat pada Firebase Realtime Database serta mekanisme validasi data yang lebih lengkap dapat menjadi pengembangan berikutnya untuk meningkatkan keandalan sistem. Pengembangan lebih lanjut juga dapat dilakukan dengan menambahkan protokol komunikasi lain seperti MQTT, ESP-NOW, atau Bluetooth Low Energy (BLE), serta meningkatkan jumlah LED yang dikendalikan guna mengevaluasi performa sistem pada beban yang lebih besar.

Daftar Pustaka

Al Furuqi, S. H., Hafsin, M. A., & Haryono. (2022). IoT-based remote light control system using Bot Telegram application. Jurnal Teknik Informatika (JUTIF), 3(6), 1475–1482. https://doi.org/10.20884/1.jutif.2022.3.6.313

Arm, J., Baštán, O., Mihálik, O., & Bradáč, Z. (2022). Measuring the performance of FreeRTOS on ESP32 multi-core. IFAC-PapersOnLine, 55(4), 292–297. https://doi.org/10.1016/j.ifacol.2022.06.048

Ayeni, P. O., & Adesoba, O. C. (2024). IoT-based home control system using NodeMCU and Firebase. Journal of Edge Computing, 4, 17–34. https://doi.org/10.55056/jec.814

Bestari, D. N., & Wibowo, A. (2023). An IoT-based real-time weather monitoring system using Telegram Bot and ThingsBoard platform. International Journal of Interactive Mobile Technologies (iJIM), 17(6), 4–19. https://doi.org/10.3991/ijim.v17i06.34129

Chi, H., Zeng, Q., & Du, X. (2023). Detecting and handling IoT interaction threats in multi-platform multi-control-channel smart homes. *Proceedings of the USENIX Security Symposium 2023*.

Chiesa, G., Di Vita, D., Ghadirzadeh, A., Herrera, A. H. M., & Rodriguez, J. C. L. (2020). A fuzzy-logic IoT lighting and shading control system for smart buildings. Automation in Construction, 120, 103397. https://doi.org/10.1016/j.autcon.2020.103397

Foltýnek, P., Babiuch, M., & Šuránek, P. (2019). Measurement and data processing from Internet of Things modules by dual-core application using ESP32 board. Measurement and Control, 52(7–8). https://doi.org/10.1177/0020294019857748

Hercog, D., Lerher, T., Truntič, M., & Težak, O. (2023). Design and implementation of ESP32-based IoT devices. *Sensors*, 23(15), 6739. https://doi.org/10.3390/s23156739

Irsyada, R., Haq, M. A., Rohmah, N. A., Saputra, P. A. H., & Jannah, R. (2022). Implementasi NodeMCU ESP8266 dan sensor cahaya pada lampu berbasis Internet of Things. Jurnal Ilmiah Sistem Informasi dan Ilmu Komputer, 2(1). https://doi.org/10.55606/juisik.v2i1.514

Kholik, M., Setiawan, B., & Fauzi, A. (2023). Design and implementation of a smart home system with the Internet of Things (IoT) using ESP32. Inspiration: Jurnal Teknologi Informasi dan Komunikasi. https://doi.org/10.35585/inspir.v13i1.39

Munthe, C. C., & Febriandirza, A. (2024). Smart home innovation: IoT-based lighting control system design. Mechatronics and Automation Technology. https://doi.org/10.3233/ATDE241310

Paudel, N., & Neupane, R. P. (2021). A general architecture for a real-time monitoring system based on the Internet of Things. Internet of Things, 14, 100367. https://doi.org/10.1016/j.iot.2021.100367

Prameswari, C. A. (2026). Sistem smart home berbasis ESP32 dengan RTOS dan MQTT serta aplikasi Flutter untuk monitoring real-time. Jurnal Informatika dan Teknik Elektro Terapan, 14(2). https://doi.org/10.23960/jitet.v14i2.9367

Prasetyo, M. A., Kusumawardana, A., Mustika, S. N., & Rizal, R. F. (2026).

IMPLEMENTASI SISTEM MONITORING DAN KENDALI GENSET LPG SECARA NEAR REAL-TIME MENGGUNAKAN ESP32-S3 DAN FIREBASE: IMPLEMENTATION OF A NEAR REAL-TIME LPG GENERATOR MONITORING AND CONTROL SYSTEM USING ESP32-S3 AND FIREBASE. Rabit: Jurnal Teknologi dan Sistem Informasi Univrab, 11(1), 822-836. https://doi.org/10.36341/rabit.v11i1.7135

Putra, F. P., Sabirin, S., & Soetanto, H. (2025). Prototype of Internet of Things-based control system using Telegram with Bot API method. Jurnal Syntax Transformation, 6(2). https://doi.org/10.46799/jst.v6i2.1055

Ramadan, A., Martana, M. R., & Ramadanu, N. S. (2026). ESP8266-based automatic light control system based on PIR sensor and Blynk application monitoring. *Tech: Journal of Engineering Science*, 2(2), 181–196. https://doi.org/10.69836/tech.v2i1.830

Saputra, T., & Surapati, U. (2024). Analysis of the effectiveness of IoT-based automatic street lighting control using linear regression method. *IJSECS*, 4(2). https://doi.org/10.35870/ijsecs.v4i2.2878

Serepas, F., Papias, I., Christakis, K., Dimitropoulos, N., & Marinakis, V. (2025). Lightweight embedded IoT gateway for smart homes based on an ESP32 microcontroller. *Computers*, 14(9), 391. https://doi.org/10.3390/computers14090391

Taiwo, O., & Ezugwu, A. E. (2021). Internet of Things-based intelligent smart home control system. Security and Communication Networks, 2021, Article 9928254. https://doi.org/10.1155/2021/9928254

Wijayanti, M. (2022). Prototype smart home dengan NodeMCU ESP8266 berbasis IoT. Jurnal Ilmiah Tek, 1(2), 101–107. https://doi.org/10.56127/juit.v1i2.169

Yamin, M. I., Ariman, Dinata, R. M., & Purnomo, N. (2026). Comparative performance evaluation of MQTT, WebSocket, and Firebase on ESP32-C3 for real-time IoT communication. *JATI (Jurnal Mahasiswa Teknik Informatika)*, 10(3), 3917–3923. https://doi.org/10.36040/jati.v10i3.18056