

IMPLEMENTASI OTOMASI AC RUANG KELAS BERBASIS JADWAL KULIAH MENGGUNAKAN *INTERNET OF THINGS* DAN ALGORITMA *INTERVAL MERGING*

Afqilla Regina Putri¹, Sahirul Alam²

^{1,2}Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada, Yogyakarta, Daerah Istimewa Yogyakarta, Indonesia

¹afqillareginaputri@mail.ugm.ac.id

²sahirul.alam@mail.ugm.ac.id

Abstract— *Air conditioner (AC) usage in campus classrooms is often inefficient because units may remain operating when rooms are unoccupied. This study designs and implements an IoT-based classroom AC control and monitoring system that integrates schedule-driven automation, a web dashboard, a PHP RESTful API served by Nginx, an ESP8266-based AC control node, and an independent ESP32-based wattmeter node. The scheduling mechanism applies interval merging to consolidate overlapping class schedules into unified operating windows while retaining manual override capability. The system was evaluated through functional testing, backend performance testing across eight RESTful API endpoints, and a preliminary one-day energy consumption scenario consisting of four class sessions. All functional scenarios were successfully completed, while the average Time to First Byte (TTFB) ranged from 8.38 to 27.21 ms and the average throughput ranged from 7.47 to 274.59 kbps. In the preliminary energy test, cumulative AC energy consumption decreased from 39.6 kWh under the baseline scenario to 27.0 kWh with schedule-based automation, corresponding to a reduction of 12.6 kWh (31.8%). These results indicate the potential of schedule-driven IoT automation to reduce unnecessary AC operation, although further multi-day testing and measurement validation are required to confirm the energy-saving performance under varying operating conditions.*

Keywords—*AC Energy Efficiency, Cron Job Automation, Internet of Things, RESTful API, Schedule Merging.*

Intisari— Penggunaan *air conditioner* (AC) pada ruang kelas kampus sering kali tidak efisien karena unit AC dapat tetap beroperasi ketika ruangan tidak digunakan. Penelitian ini merancang dan mengimplementasikan sistem kontrol dan *monitoring* AC ruang kelas berbasis IoT yang mengintegrasikan otomasi berbasis jadwal, *web dashboard*, RESTful API berbasis PHP yang dilayani Nginx, *node* kontrol AC berbasis ESP8266, dan *node wattmeter* independen berbasis ESP32. Mekanisme penjadwalan menerapkan *interval merging* untuk menggabungkan jadwal kuliah yang tumpang tindih menjadi jendela operasi terpadu dengan tetap menyediakan fungsi *manual override*. Sistem dievaluasi melalui pengujian fungsional, pengujian performa *backend* pada delapan *endpoint* RESTful API, serta skenario awal pengujian konsumsi energi selama satu hari yang terdiri atas empat sesi perkuliahan. Seluruh skenario fungsional berhasil dijalankan, dengan rata-rata *Time to First Byte* (TTFB) sebesar 8,38–27,21 ms dan *throughput* rata-rata sebesar 7,47–274,59 kbps. Pada pengujian energi awal, konsumsi energi kumulatif AC menurun dari 39,6 kWh pada kondisi baseline menjadi 27,0 kWh dengan otomasi berbasis jadwal, atau berkurang sebesar 12,6 kWh (31,8%). Hasil ini

menunjukkan potensi otomasi IoT berbasis jadwal dalam mengurangi operasi AC yang tidak diperlukan, meskipun pengujian multi-hari dan validasi pengukuran lebih lanjut diperlukan untuk mengonfirmasi kinerja penghematan energi pada berbagai kondisi operasional.

Kata Kunci—Efisiensi Energi AC, *Internet of Things*, Otomasi Cron Job, Penggabungan Jadwal, RESTful API.

I. PENDAHULUAN

Konsumsi energi listrik pada bangunan pendidikan terus meningkat seiring meluasnya penggunaan perangkat pendingin ruangan. *Air conditioner* (AC) merupakan salah satu kontributor terbesar konsumsi energi listrik pada gedung-gedung di negara beriklim tropis, termasuk Indonesia. Sebuah penelitian mengenai konsumsi energi gedung universitas beriklim tropis menemukan bahwa penggunaan AC di Escola Politécnica Universidade Federal da Bahia mencapai 36% dari total konsumsi energi listrik gedung [1]. Kondisi serupa juga terjadi di gedung perkantoran Malaysia, di mana AC menyumbang hingga 53% dari total konsumsi energi bangunan institusi [2].

Permasalahan yang umum dijumpai di lingkungan kampus adalah unit AC yang tetap beroperasi ketika ruangan tidak digunakan atau tidak dinyalakan sesuai kebutuhan karena pengoperasiannya masih bergantung pada intervensi pengguna. Sebuah studi pada Gedung Utama BPISDKP mencatat bahwa unit AC pada 13 ruangan dapat beroperasi lebih dari 24 jam tanpa pengendalian yang efisien [4]. Kondisi tersebut menunjukkan perlunya mekanisme kontrol yang dapat menyesuaikan waktu operasi AC dengan penggunaan ruangan secara lebih sistematis.

Perkembangan teknologi *Internet of Things* (IoT) membuka peluang untuk mengotomatisasi dan memantau perangkat listrik dari jarak jauh secara *real-time* [11]. RESTful API juga banyak digunakan dalam komunikasi perangkat IoT karena sederhana, skalabel, dan kompatibel dengan teknologi *web* [5], [12]. Sejumlah penelitian telah mengembangkan sistem kontrol dan *monitoring* AC berbasis IoT, antara lain *monitoring* suhu dan konsumsi energi menggunakan NodeMCU ESP8266 dan platform Blynk [3], *monitoring* dan kontrol AC berbasis *web* menggunakan ZigBee dan ESP8266 [6], pengelolaan suhu AC untuk infrastruktur *eco-smart* [7], serta *monitoring* AC berbasis IoT untuk mendukung pemeliharaan fasilitas [2].

Pendekatan berbasis jadwal akademik juga telah digunakan untuk menyesuaikan operasi AC dengan penggunaan ruang kelas, dengan penghematan energi yang dilaporkan pada beberapa penelitian berkisar antara 20% hingga 45,75% [4], [9], [10].

Meskipun pendekatan berbasis jadwal dapat mengurangi ketergantungan pada pengoperasian manual, pengolahan jadwal secara independen dapat menimbulkan masalah ketika terdapat beberapa jadwal yang tumpang tindih pada ruangan dan hari yang sama. Setiap entri jadwal dapat menghasilkan pasangan perintah ON/OFF tersendiri sehingga muncul *trigger* yang redundan atau konflik waktu operasi. Permasalahan ini menjadi relevan pada pengelolaan ruang kelas yang digunakan oleh beberapa kegiatan perkuliahan dengan rentang waktu yang saling beririsan. Oleh karena itu, data jadwal perlu diproses terlebih dahulu menjadi jendela operasi AC yang konsisten sebelum digunakan oleh sistem otomasi.

Interval merging merupakan pendekatan komputasional untuk mengonsolidasikan sejumlah interval waktu yang saling tumpang tindih menjadi interval yang tidak redundan. Secara umum, interval terlebih dahulu diurutkan berdasarkan waktu mulai, kemudian dipindai secara linear untuk menentukan apakah suatu interval perlu digabungkan dengan interval sebelumnya. Dengan proses pengurutan berkompleksitas $O(n \log n)$ dan pemindaian linear $O(n)$, kompleksitas keseluruhan pendekatan ini adalah $O(n \log n)$ [16]. Karakteristik tersebut memungkinkan kumpulan jadwal yang berubah secara dinamis diproses secara efisien sebelum digunakan sebagai dasar pembentukan *trigger* operasi [17]. Dalam konteks penelitian ini, *interval merging* digunakan untuk mengonsolidasikan jadwal perkuliahan yang saling tumpang tindih pada ruangan dan hari yang sama sehingga tidak menghasilkan pasangan perintah ON/OFF yang redundan.

Kontribusi penelitian ini bukan pada pengembangan algoritma *interval merging* baru, melainkan pada penerapannya sebagai bagian dari arsitektur otomasi AC berbasis IoT yang terintegrasi dengan data jadwal akademik. Sistem menggabungkan rekalkulasi *trigger* secara otomatis ketika jadwal ditambah, diubah, atau dihapus, *Automation Engine* berbasis *cron job*, mekanisme *manual override*, RESTful API, serta monitoring konsumsi daya melalui *node* IoT independen. Integrasi tersebut dirancang untuk menghasilkan jendela operasi AC yang konsisten berdasarkan jadwal sekaligus tetap memungkinkan intervensi pengguna ketika diperlukan.

Berdasarkan pendekatan tersebut, penelitian ini bertujuan merancang dan mengimplementasikan sistem IoT untuk kontrol dan *monitoring* AC ruang kelas berdasarkan data jadwal perkuliahan serta mengevaluasi fungsionalitas, performa *backend*, dan dampak awalnya terhadap konsumsi energi. Sistem mengintegrasikan *web dashboard*, RESTful API, perangkat ESP8266 untuk kontrol AC melalui inframerah, *node* ESP32 untuk *monitoring* daya, serta *Automation Engine* berbasis *cron job* dan *interval merging* untuk mengelola jendela operasi AC.

II. METODE PENELITIAN

A. Arsitektur Sistem

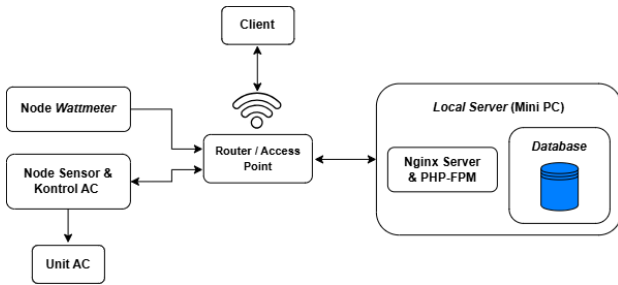
Arsitektur sistem dirancang untuk mengintegrasikan empat komponen utama yang saling berkomunikasi, yaitu *Node* IoT, Infrastruktur Jaringan Lokal, Mini PC (*Local Server*), dan Sisi *Client*, sebagaimana ditunjukkan pada Gambar 1. Rincian alat dan bahan perangkat keras dan perangkat lunak yang digunakan pada penelitian ini disajikan pada Tabel I dan Tabel II.

Tabel I
Perangkat Keras Sistem

Komponen	Spesifikasi	Fungsi
<i>Server local</i> (Mini PC)	Intel NUC 11 Pro, Celeron N5105, RAM 8 GB, NVMe 512 GB	Menjalankan Nginx, PHP-FPM, MariaDB
Router Wi-Fi	TP-Link TL-WR820N, IEEE 802.11b/g/n, 2,4 GHz	Jaringan lokal antar-komponen
<i>Node sensor & kontrol AC</i>	ESP8266 + DHT11 + IR transmitter	Untuk membaca suhu dan kelembaban, eksekusi ON/OFF AC
<i>Node wattmeter</i>	ESP32 + SCT-013 + ZMPT101B	Ukur arus dan tegangan AC secara simultan
Unit AC	Panasonic CS/CU-YN18WKJ (2 PK)	Objek kendali dan monitoring

Tabel II
Perangkat Lunak Sistem

Komponen	Spesifikasi/Versi	Fungsi
OS server	Ubuntu 22	Menjalankan Nginx dan MariaDB
<i>Backend</i>	PHP (Vanilla, OOP)	Membangun RESTful API
<i>Frontend</i>	HTML5, CSS3, JS (ES Modules)	Antarmuka <i>dashboard</i>
<i>Framework CSS</i>	Bootstrap 5	Layout responsif
<i>Library visualisasi</i>	Chart.js	Grafik konsumsi daya
DBMS	MariaDB	Data operasional dan log sensor
<i>Web server</i>	Nginx	Reverse proxy HTTP
<i>Tools pengujian</i>	Postman, Chrome DevTools, Wireshark, Mockoon	Uji fungsional, TTFB, <i>throughput</i>



Gambar 1. Arsitektur Sistem Kontrol dan Monitoring AC

Node IoT terdiri atas dua mikrokontroler independen: *Node Sensor & Kontrol AC* (ESP8266) yang membaca suhu/kelembaban dan mengeksekusi perintah ON/OFF via *IR transmitter*, serta *Node Wattmeter* (ESP32) yang membaca arus dan tegangan AC secara simultan. Kedua *node* terhubung ke Mini PC melalui Infrastruktur Jaringan Lokal (router Wi-Fi). Pada Mini PC dijalankan dua lapisan layanan: *Application Layer*, berupa RESTful API yang dilayani Nginx sebagai *web server* [13] dengan PHP-FPM sebagai prosesor *backend* melalui *front controller* yang mendelegasikan permintaan ke empat *controller* (Room, Schedule, Sensor, Ac), dan *Automation Engine* (cron_automation.php) yang dipicu *scheduler* sistem operasi setiap satu menit; serta *Data Layer*, berupa MariaDB [15] dengan dua skema terpisah untuk data operasional dan data log sensor. Sisi *Client* berupa *web dashboard* yang diakses melalui browser.

Kontrol AC pada arsitektur ini berjalan melalui dua jalur yang berdampingan. Pada jalur otomatis, *Automation Engine* membaca jadwal dan status *override* langsung dari *Data Layer*, kemudian mengirimkan perintah ON/OFF langsung ke *Node IoT* tanpa melalui *Application Layer*. Pada jalur manual, perintah dari Sisi *Client* dikirimkan langsung ke *Node IoT* agar eksekusi fisik instan, dengan status hasilnya kemudian disinkronkan ke *Data Layer* melalui *Application Layer* agar tetap konsisten dan dapat dibaca kembali oleh *Automation Engine* maupun sesi *client* lain.

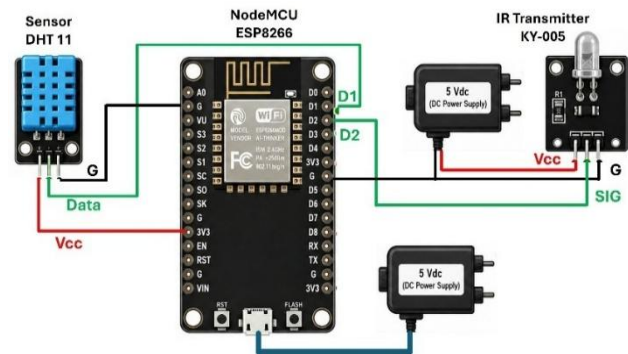
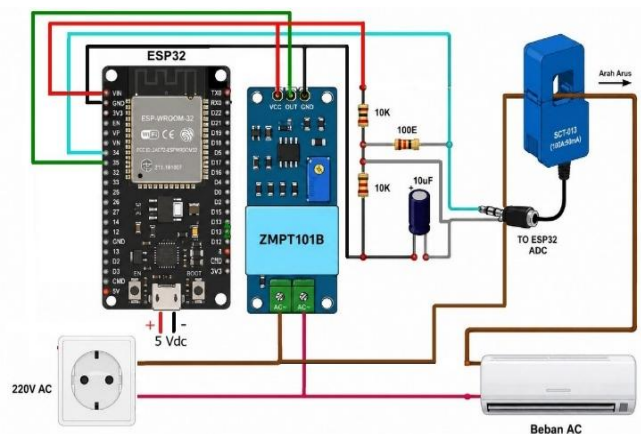
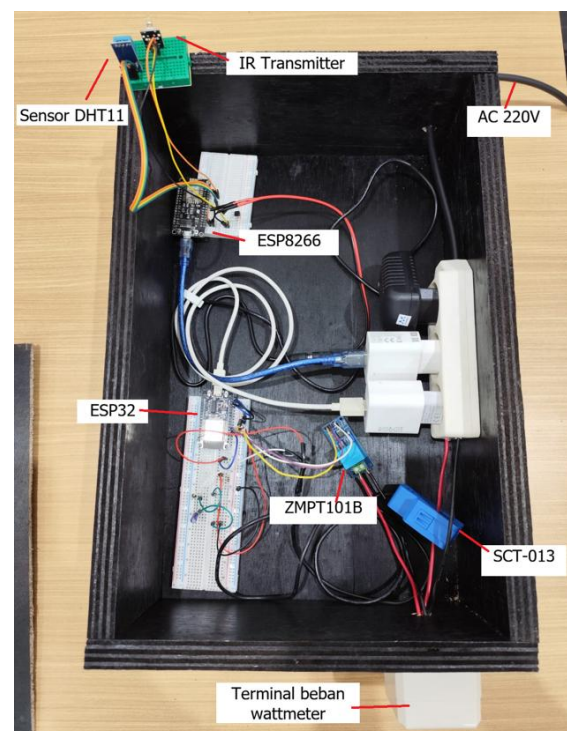
B. Perancangan dan Implementasi Perangkat Keras

Perangkat keras IoT dirancang terintegrasi langsung dengan sistem informasi jadwal kuliah agar beroperasi secara kontekstual tanpa bergantung pada sensor kehadiran (PIR) yang rawan bias deteksi. Rancangan skematik memisahkan fungsi *Node Sensor & Kontrol AC* dari *Node Wattmeter*, sehingga kedua fungsi berjalan independen dan tidak saling mengganggu.

Rancangan skematik *Node Sensor & Kontrol AC* ditunjukkan pada Gambar 2. *NodeMCU ESP8266* dihubungkan dengan sensor DHT11 melalui satu jalur data (GPIO) untuk membaca suhu dan kelembaban ruangan secara periodik, serta dengan modul *IR transmitter* melalui pin D5 sebagai jalur keluaran sinyal inframerah menuju unit AC.

Berdasarkan Gambar 2, ketika perintah menyala diterima dari server, mikrokontroler memancarkan satu kode sinyal inframerah tetap hasil *capture-and-replay*, dengan konfigurasi *mode Cool*, suhu target 25°C, serta kecepatan kipas dan *swing* otomatis; *firmware* tidak mendukung pengubahan parameter tersebut secara dinamis karena fungsi kendali pada penelitian ini hanya mencakup

power ON/OFF. Selain itu, *firmware Node Sensor & Kontrol AC* juga menjalankan *web server* lokal dengan dua *endpoint*, yaitu GET /status untuk melaporkan status daya AC terkini dan POST /control untuk menerima perintah ON/OFF dari server.

Gambar 2. Skematik *Node Sensor & Kontrol AC*Gambar 3. Skematik *Node Wattmeter*

Gambar 4. Purwarupa Perangkat IoT

Rancangan skematik *Node Wattmeter* ditunjukkan pada Gambar 3. ESP32 dihubungkan dengan sensor arus SCT-013 yang bertipe *split-core current transformer*, dipasang non-invasif (*clamp-on*) pada kabel fasa AC tanpa perlu memutus rangkaian, serta dengan sensor tegangan ZMPT101B yang mengukur tegangan AC hingga 250V melalui isolasi galvanis yang memisahkan sirkuit tegangan tinggi dari sirkuit mikrokontroler. Kedua sensor terhubung ke dua kanal masukan analog terpisah pada ESP32. Nilai bacaan mentah dari kedua kanal analog dikalikan dengan faktor kalibrasi agar nilainya mendekati alat ukur pembanding yaitu *wattmeter* digital fisik. Selanjutnya, hasil pengukuran daya dikirimkan ke server setiap lima detik melalui permintaan HTTP GET. Hasil implementasi perangkat keras IoT dalam bentuk purwarupa awal ditunjukkan pada Gambar 4. Perangkat ini dipasang di setiap unit AC untuk otomasi pengontrolan dan monitoring konsumsi daya.

C. Perancangan dan Implementasi Perangkat Lunak

Basis data dirancang dengan dua skema terpisah pada MariaDB: skema operasional yang bersifat relasional (*rooms*, *ac_status*, *ac_overrides*, *schedules*, *ac_triggers*) dan skema climasense_sensor_log yang bersifat *insert-only* berfrekuensi tinggi (*sensor_logs*), sehingga performa *query* operasional tidak terbebani volume penulisan data sensor. Ringkasan relasi setiap tabel terhadap tabel *rooms* disajikan pada Tabel III. *Backend* RESTful API dibangun dengan pola *front controller*: seluruh permintaan HTTP diterima oleh *index.php*, diurai menjadi *resource*, *id*, dan *action*, lalu didelegasikan ke salah satu dari empat *controller* (Room, Schedule, Sensor, Ac) sesuai Tabel IV. Koneksi basis data menerapkan pola Singleton (*getDB()* dan *getSensorDB()*) memanfaatkan ekstensi PDO PHP [14] agar setiap *controller* menggunakan satu koneksi yang sama tanpa membuka handle baru, sementara konfigurasi alamat *Node IoT* per ruangan dipusatkan pada berkas *iot.php* lengkap dengan mekanisme *fallback* dan batas *timeout* agar kegagalan komunikasi ke satu *Node IoT* tidak menghambat proses otomasi ruangan lain. Sebelum *routing*, *index.php* menetapkan *header* CORS dan JSON, sementara kombinasi *resource/id/action/method* yang tidak cocok dikembalikan sebagai HTTP 404.

Tabel III
Entitas Utama Basis Data Operasional

Tabel	Relasi thd. <i>rooms</i>	Fungsi
<i>ac_status</i>	<i>One-to-one</i>	Status AC terkini (<i>power</i> , <i>mode</i> , suhu, kelembaban)
<i>ac_overrides</i>	<i>One-to-(0 atau 1)</i>	Status <i>override</i> manual yang sedang berlaku
<i>schedules</i>	<i>One-to-many</i>	Data jadwal kegiatan akademik per ruangan
<i>ac_triggers</i>	<i>One-to-many</i>	Jendela waktu ON/OFF hasil <i>schedule merging</i>
<i>sensor_logs</i>	Relasi logis (<i>room_code</i>)	Log historis sensor (skema terpisah)

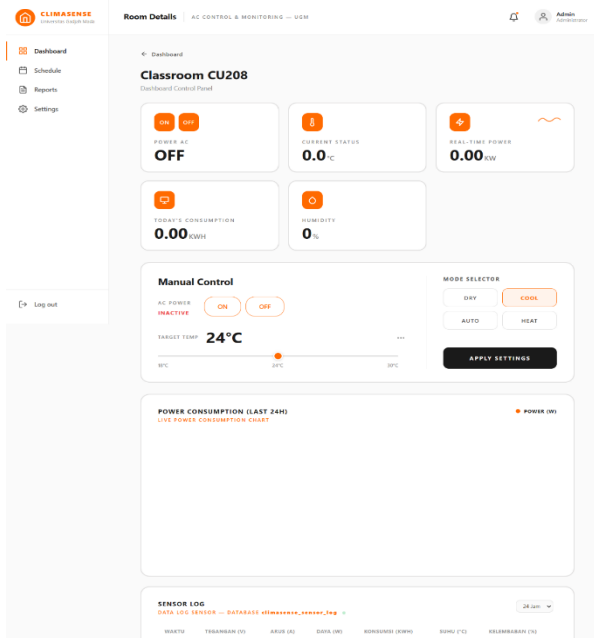
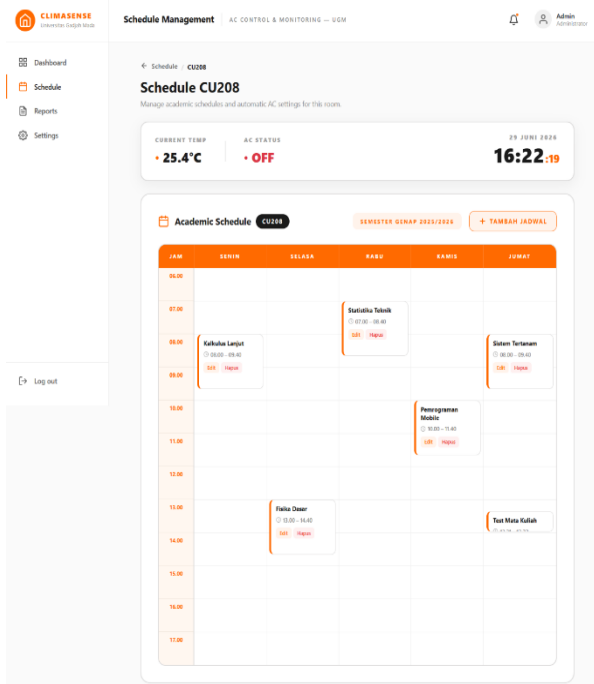
Tabel IV
Endpoint RESTful API Utama

Method	Endpoint	Controller / Fungsi
GET/POST	/rooms	RoomController: CRUD ruangan & status AC
POST/PUT/DELETE	/schedules/{id}	ScheduleController: CRUD jadwal, memicu <i>recalcTriggers()</i>
GET	/ac-triggers	ScheduleController: jendela waktu hasil <i>merging</i>
GET/POST	/rooms/{id}/control	AcController: baca/sinkron status kendali AC
POST	/sensor/{room_id}	SensorController: simpan data sensor dari <i>Node IoT</i>
GET	/sensor/{room_id}/history	SensorController: riwayat sensor untuk grafik

Inti kebaruan penelitian pada sisi *backend* terletak pada fungsi *recalcTriggers()* di dalam *ScheduleController*, yang dipanggil otomatis setiap kali terjadi operasi tambah, ubah, atau hapus jadwal. Fungsi ini menerapkan algoritma *interval merging* melalui empat tahap: (1) menghapus *trigger* lama pada ruangan dan hari terkait; (2) mengambil dan mengurutkan seluruh jadwal berdasarkan waktu mulai; (3) memindai jadwal secara satu-arah untuk mendeteksi tumpang tindih - dua interval $[s_1, e_1]$ dan $[s_2, e_2]$ digabung menjadi $[\min(s_1, s_2), \max(e_1, e_2)]$ apabila $s_2 \leq e_1$; dan (4) menyimpan jendela waktu gabungan ke tabel *ac_triggers*. Tahap pengurutan menggunakan *merge sort* berkompleksitas $O(n \log n)$ [16], dilanjutkan pemindaian linear $O(n)$, sehingga kompleksitas keseluruhan algoritma adalah $O(n \log n)$; pendekatan ini terbukti menghasilkan alokasi sumber daya penjadwalan yang lebih optimal dibandingkan penanganan interval secara independen [17].

Otomasi berbasis jadwal dijalankan oleh *Automation Engine* (*cron_automation.php*) sebagai skrip PHP CLI yang dipicu *Cron Job server* setiap satu menit, independen dari sesi browser. Algoritmanya terdiri atas empat tahap: membaca seluruh ruangan terdaftar, memeriksa apakah waktu saat ini berada dalam jendela *ac_triggers*, memeriksa keberadaan *override* manual pada *ac_overrides*, dan memperbarui *ac_status* beserta mengirim perintah HTTP melalui cURL ke *Node IoT* apabila status target berbeda dari status saat ini.

Frontend dibangun sebagai *Single Page Application* modular menggunakan HTML5, *framework* Bootstrap 5 [18], dan Vanilla JavaScript, dengan prinsip *thin client* seluruh keputusan evaluasi jadwal dilakukan di *backend* sehingga *frontend* tidak memuat logika *timer* apa pun. Kontrol manual direalisasikan pada fungsi *buttonPower()* melalui dua tahap: mengirim perintah langsung ke *Node IoT* agar eksekusi fisik instan, kemudian menyinkronkan status hasilnya ke *backend*. Hasil implementasi antarmuka ditunjukkan pada Gambar 5 dan Gambar 6.

Gambar 5. Halaman *Room Details*Gambar 6. Halaman *Schedule Management*

Halaman *Room Details* (Gambar 5) menyajikan tiga kelompok informasi utama: kartu kondisi ruangan *real-time* (status daya, suhu, daya terpakai, energi harian, kelembaban), panel *Manual Control* (tombol ON/OFF, slider suhu, pemilihan mode), serta grafik tren konsumsi daya 24 jam menggunakan Chart.js [19] yang diperbarui berkala melalui *polling* data sensor. Halaman *Schedule Management* (Gambar 6) menampilkan kalender mingguan visual (grid akademik Senin-Jumat) dengan setiap kegiatan perkuliahan diplot sebagai blok berwarna menggunakan *absolute positioning* berdasarkan jam mulai dan durasi, dilengkapi tombol Edit dan Hapus pada setiap blok jadwal serta modal Tambah Jadwal merupakan

representasi visual langsung dari data yang diproses oleh algoritma *recalcTriggers()*.

D. Algoritma Penggabungan Jadwal (*Interval Merging*)

Tanpa penggabungan interval, setiap baris jadwal dapat menghasilkan pasangan *trigger* ON/OFF secara independen. Kondisi ini berpotensi menimbulkan *trigger* yang redundan ketika terdapat dua atau lebih jadwal yang saling tumpang tindih pada ruangan dan hari yang sama. Oleh karena itu, algoritma *interval merging* digunakan untuk mengonsolidasikan jadwal yang tumpang tindih menjadi satu jendela waktu operasi AC sebelum dievaluasi oleh *Automation Engine*.

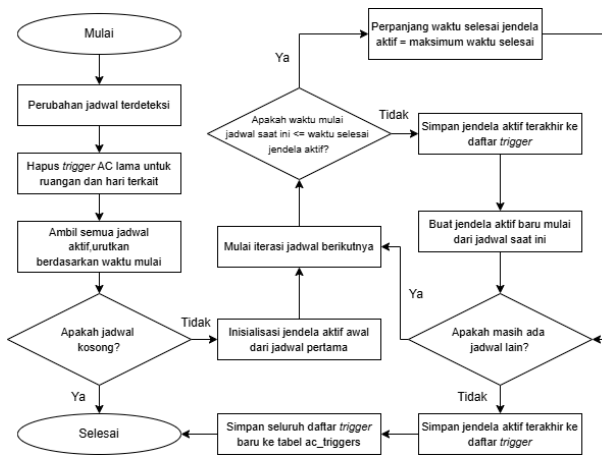
Secara formal, dua interval yang telah diurutkan berdasarkan waktu mulai, $([s_1, e_1])$ dan $([s_2, e_2])$, digabung menjadi $([\min(s_1, s_2), \max(e_1, e_2)])$ apabila $(s_2 \leq e_1)$. Algoritma diwujudkan dalam fungsi *recalcTriggers()* pada *ScheduleController* dan dijalankan setiap kali terjadi operasi tambah, ubah, atau hapus jadwal. Proses pada Gambar 7 terdiri atas empat tahap, yaitu menghapus *trigger* lama pada ruangan dan hari terkait, mengurutkan seluruh jadwal berdasarkan waktu mulai menggunakan *merge sort* $O(n \log n)$ [16], melakukan pemindaian linear $O(n)$ untuk menggabungkan interval yang tumpang tindih, dan menyimpan jendela hasil penggabungan ke tabel *ac_triggers* beserta *schedule ids*.

Sebagai contoh, terdapat tiga jadwal pada hari yang sama, yaitu $[08:00-09:40]$, $[09:30-11:00]$, dan $[13:00-14:40]$. Karena jadwal kedua dimulai sebelum jadwal pertama berakhir, dua interval pertama digabung menjadi $[08:00-11:00]$, sedangkan jadwal ketiga tetap tersimpan sebagai interval terpisah. Dengan demikian, sistem menghasilkan dua jendela operasi pada *ac_triggers* alih-alih tiga pasangan *trigger* independen.

Perubahan jadwal ditangani melalui proses rekalkulasi tersebut. Apabila suatu kelas dibatalkan dan jadwal dihapus dari sistem, *recalcTriggers()* menghapus trigger sebelumnya dan membangun kembali jendela operasi berdasarkan jadwal yang masih aktif. Jika waktu perkuliahan diperpanjang dan perubahan tersebut diperbarui pada data jadwal, *trigger* juga dihitung ulang secara otomatis. Untuk perubahan operasional yang belum tercatat pada jadwal, seperti perkuliahan yang berlangsung lebih lama dari waktu yang direncanakan, pengguna dapat menggunakan mekanisme manual *override* agar status AC tidak langsung diubah oleh evaluasi jadwal otomatis.

Kompleksitas waktu keseluruhan algoritma adalah $O(n \log n)$, yang didominasi oleh proses pengurutan, sedangkan pemindaian interval memiliki kompleksitas $O(n)$. Tabel *ac_triggers* hasil penggabungan selanjutnya menjadi acuan bagi *Automation Engine* (*cron_automation.php*) yang mengevaluasi status target AC setiap satu menit.

Algoritma pada implementasi saat ini hanya menggabungkan interval yang saling tumpang tindih dan belum menerapkan ambang minimum jeda ON/OFF untuk menggabungkan dua jadwal yang terpisah oleh jeda singkat. Oleh karena itu, *interval merging* pada penelitian ini ditujukan terutama untuk menghilangkan *trigger* yang redundan akibat jadwal yang tumpang tindih dan tidak diklaim secara khusus sebagai mekanisme pencegahan *short cycling* pada seluruh kondisi operasi.



Gambar 7. Flowchart Algoritma Penggabungan Jadwal (*recalcTriggers()*)

E. Skenario Pengujian

Pengujian sistem dilakukan melalui tiga tahap. Pertama, pengujian fungsional (*black-box testing*) terhadap sembilan skenario yang mencakup manajemen ruangan, kendali manual AC, pemantauan kondisi ruangan, manajemen jadwal, otomasi penjadwalan, dan responsivitas antarmuka pada tiga kategori *viewport* (mobile, tablet, desktop). Kedua, pengujian performa sistem terhadap delapan *endpoint* RESTful API, meliputi *Time to First Byte* (TTFB) yang diukur melalui panel *Timing* pada Google Chrome DevTools, dan *throughput* yang diukur melalui aplikasi Wireshark dengan menghitung rasio *Bytes* terhadap *Time Span* pada *Capture File Properties*, dikonversi ke kbps. Setiap *endpoint* diuji sepuluh kali pengulangan untuk memperoleh nilai minimum, maksimum, dan rata-rata. Ketiga, pengujian dampak efisiensi energi dilakukan dengan membandingkan konsumsi daya AC pada kondisi *baseline* (tanpa otomasi) dan kondisi dengan sistem kontrol dan otomatisasi AC pada pola jadwal empat sesi perkuliahan dalam satu hari.

III. HASIL DAN PEMBAHASAN

A. Hasil Pengujian Fungsionalitas

Pengujian *black-box* dilakukan terhadap sembilan skenario yang merepresentasikan seluruh fitur utama *dashboard*, sebagaimana dirangkum pada Tabel V. Seluruh skenario dinyatakan berhasil (100% *pass rate*), termasuk pengujian penggabungan jadwal tumpang tindih yang menghasilkan satu baris data baru pada tabel *ac_triggers*, transisi otomatis AC menyala dan mati tepat waktu yang terkonfirmasi melalui log terminal *cron_automation.php* (*STATE CHANGE DETECTED*), serta *manual override* yang terbukti mampu mempertahankan status manual pengguna tanpa ditimpa oleh hasil evaluasi jadwal berikutnya.

Tabel V
Rekapitulasi Hasil Pengujian Fungsionalitas

No	Skenario Pengujian	Status
1	Tambah/hapus ruangan	Berhasil
2	Kendali manual ON/OFF AC	Berhasil

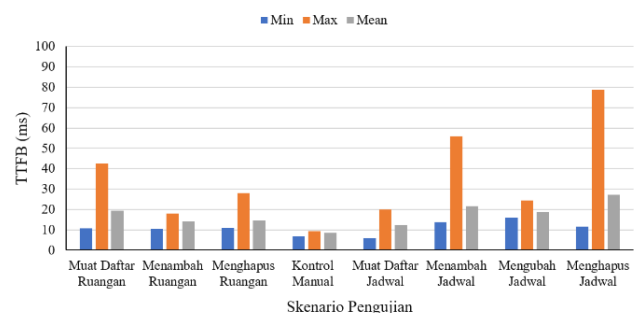
No	Skenario Pengujian	Status
3	Pemantauan suhu, kelembaban, daya, dan energi harian	Berhasil
4	Grafik konsumsi daya 24 jam dan tabel log sensor	Berhasil
5	Validasi tambah, ubah, dan hapus jadwal	Berhasil
6	Penggabungan jadwal tumpang tindih (<i>recalcTriggers</i>)	Berhasil
7	Transisi otomatis ON/OFF AC tepat waktu	Berhasil
8	<i>Manual override</i> di tengah jadwal aktif	Berhasil
9	Responsivitas antarmuka (mobile, tablet, desktop)	Berhasil

B. Hasil Pengujian Performa Sistem

Pengujian TTFB, yaitu metrik performa web yang mengukur durasi sejak permintaan dikirim hingga *byte* pertama respons diterima [20], terhadap delapan *endpoint* RESTful API menunjukkan rata-rata berkisar antara 8,38 ms hingga 27,21 ms, sebagaimana disajikan pada Tabel VI dan Gambar 8. *Endpoint* Kontrol Manual ON/OFF mencatatkan TTFB rata-rata terendah (8,38 ms) karena hanya melakukan pembaruan satu kolom status pada basis data. Sebaliknya, *endpoint* Tambah Jadwal, Ubah Jadwal, dan Hapus Jadwal menunjukkan TTFB yang lebih tinggi (18,84-27,21 ms), konsisten dengan beban komputasi tambahan dari eksekusi fungsi *recalcTriggers()* pada setiap operasi jadwal.

Tabel VI
Hasil Pengujian TTFB (ms)

Skenario	Min	Max	Mean
Muat Daftar Ruangan	10,79	42,60	19,34
Tambah Ruangan	10,39	17,96	14,11
Hapus Ruangan	11,07	28,02	14,58
Kontrol Manual ON/OFF	6,86	9,28	8,38
Muat Daftar Jadwal	6,01	20,04	12,39
Tambah Jadwal	13,92	55,88	21,52
Ubah Jadwal	15,93	24,34	18,84
Hapus Jadwal	11,63	78,72	27,21

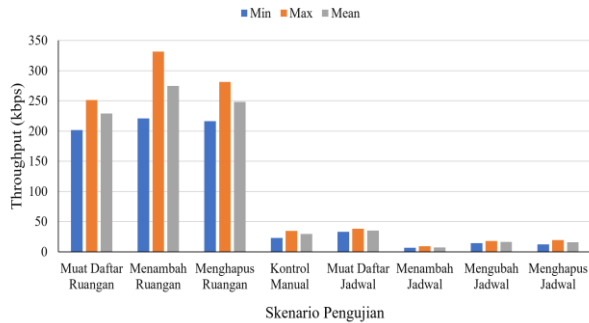


Gambar 8. Perbandingan TTFB Kedelapan *Endpoint*

Tabel VII
Hasil Pengujian *Throughput* (kbps)

Skenario	Min	Max	Mean
Muat Daftar Ruangan	201,58	251,43	228,98
Tambah Ruangan	220,86	331,47	274,59
Hapus Ruangan	216,42	281,64	248,56

Skenario	Min	Max	Mean
Kontrol Manual ON/OFF	23,23	34,56	29,72
Muat Daftar Jadwal	33,02	38,31	35,15
Tambah Jadwal	6,62	9,46	7,47
Ubah Jadwal	14,39	17,76	16,24
Hapus Jadwal	12,65	19,45	15,90



Gambar 9. Perbandingan *Throughput* Kedelapan *Endpoint*

Hasil pengujian *throughput* pada Tabel VII dan Gambar 9 memperkuat pola yang sama: *endpoint* manajemen ruangan mencapai *throughput* tertinggi (228,98-274,59 kbps) karena *time span* transaksi yang singkat, sedangkan *endpoint* manajemen jadwal mencatatkan *throughput* terendah (7,47-16,24 kbps) akibat *time span* yang jauh lebih panjang (5,16-14,96 detik) selama proses rekalkulasi dan penggabungan jadwal berlangsung. Hal ini menegaskan bahwa performa transfer data *backend* lebih dipengaruhi oleh kompleksitas komputasi pada sisi server dibandingkan besar kecilnya volume data yang dipertukarkan.

C. Hasil Pengujian Dampak Efisiensi Energi

Pengujian dampak efisiensi energi mengadaptasi metodologi studi kasus lapangan pada ruang kelas CU208 di lingkungan kampus Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada, dengan membandingkan pola konsumsi daya AC pada kondisi *baseline* (tanpa otomasi) dan kondisi dengan sistem kontrol dan otomatisasi AC, menggunakan pola jadwal kuliah harian dengan empat sesi perkuliahan sebagaimana dirangkum pada Tabel VIII.

Pengujian dilakukan selama 1 hari untuk tiap skenario (*baseline* dan otomasi). Namun pengujian tidak dilakukan pada saat perkuliahan riil berlangsung, melainkan menggunakan skenario seperti pada Tabel VIII. Ruang kelas memiliki 4 unit AC masing-masing berkapasitas 2 PK dengan jenis standard. Pengujian dilakukan pada cuaca cerah dengan suhu luar ruangan berkisar antara 24°C hingga 31°C. Kontrol AC hanya bersifat on-off dengan suhu setpoint 24°C. Selain itu, pada saat jeda perkuliahan, pintu ruangan dibuka agar skenario mendekati kondisi biasanya.

Jeda antar sesi umumnya berlangsung sekitar 20 menit, kecuali jeda antara Sesi 2 dan Sesi 3 yang mencapai sekitar 80 menit rentang jeda panjang inilah yang menjadi fokus utama pengujian, karena pada praktiknya AC sering kali tetap menyala tanpa termatikan selama periode tersebut. Pengujian juga mencakup periode pasca-kuliah (± 60 menit setelah Sesi 4 berakhir) sebagai potensi hemat

tambahan sebelum dilakukan pengecekan manual terhadap ruangan. Pada kondisi *baseline*, AC dibiarkan menyala terus-menerus sejak Sesi 1 dimulai (07:15) hingga periode pasca-kuliah berakhir (16:55), termasuk pada seluruh periode jeda, untuk merepresentasikan kebiasaan pengguna yang tidak mematikan AC secara manual saat ruangan tidak digunakan. Pada kondisi dengan sistem kontrol dan otomatisasi AC, AC dikendalikan sepenuhnya oleh mekanisme *schedule merging* dan *cron job automation*, sehingga hanya menyala pada jendela waktu hasil penggabungan jadwal (*ac_triggers*) dan mati otomatis pada seluruh periode jeda, termasuk jeda panjang antara Sesi 2 dan Sesi 3 serta periode pasca-kuliah.

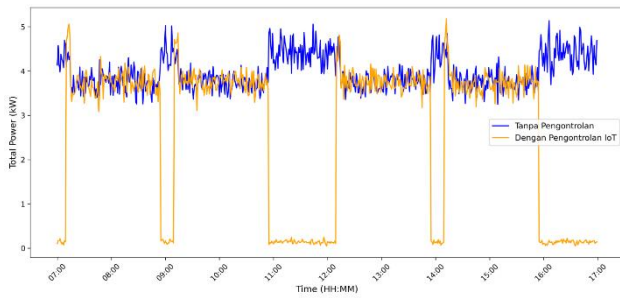
Tabel VIII
Skenario Pengujian Dampak Efisiensi Energi

Sesi/Jeda	Waktu	Durasi	Kondisi Baseline	Kondisi Otomasi AC
Sesi 1	07:15–08:55	100 mnt	AC menyala	AC menyala
Jeda 1	08:55–09:15	20 mnt	AC tetap menyala	AC mati otomatis
Sesi 2	09:15–10:55	100 mnt	AC menyala	AC menyala
Jeda Panjang	10:55–12:15	80 mnt	AC tetap menyala	AC mati otomatis
Sesi 3	12:15–13:55	100 mnt	AC menyala	AC menyala
Jeda 2	13:55–14:15	20 mnt	AC tetap menyala	AC mati otomatis
Sesi 4	14:15–15:55	100 mnt	AC menyala	AC menyala
Pasca Kuliah	15:55–16:55	60 mnt	AC tetap menyala	AC mati otomatis

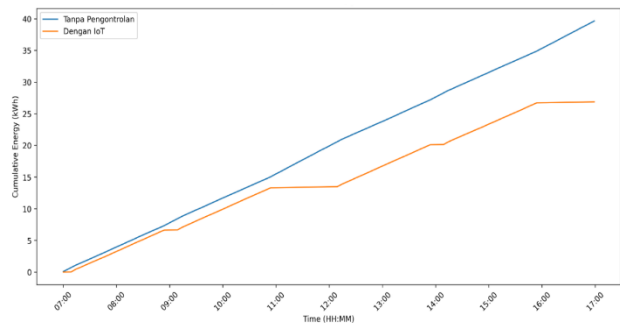
Hasil pengujian konsumsi daya pada sistem disajikan pada Gambar 10 sebagai perbandingan antara kondisi tanpa pengontrolan (*baseline*) dan kondisi ketika mekanisme otomasi berbasis IoT diterapkan. Pada skenario *baseline*, unit AC tetap beroperasi selama periode jeda antar sesi meskipun ruangan tidak sedang digunakan, sehingga konsumsi energi tetap berlangsung. Kondisi tersebut juga dipengaruhi oleh pintu ruangan yang dibiarkan terbuka selama jeda, sehingga beban pendinginan tetap terjadi atau bahkan meningkat. Sebaliknya, pada skenario dengan otomasi, sistem mengendalikan operasi AC berdasarkan jadwal yang telah ditentukan dan secara otomatis mematikan unit ketika tidak terdapat aktivitas terjadwal. Penerapan mekanisme ini menurunkan konsumsi daya selama periode jeda. Perbedaan kedua skenario terlihat pada pola kurva konsumsi daya, di mana kondisi otomasi menunjukkan area di bawah kurva yang lebih kecil dibandingkan kondisi *baseline*, yang mengindikasikan berkurangnya penggunaan energi selama periode pengujian.

Gambar 11 menunjukkan kurva energi kumulatif pada kondisi *baseline* yang terus meningkat sepanjang periode jeda, sedangkan pada kondisi dengan sistem yang dikembangkan, kurva melandai selama jeda, terutama pada rentang jeda panjang antara Sesi 2 dan Sesi 3 membuktikan bahwa *schedule merging* dan *cron job automation* efektif

mengeliminasi konsumsi energi yang tidak produktif tanpa intervensi manual.



Gambar 10. Perbandingan konsumsi daya AC (*baseline* vs otomasi IoT)



Gambar 11. Perbandingan *Cumulative Energy Consumption*

Secara kuantitatif, total konsumsi energi kumulatif menurun dari 39,6 kWh (*baseline*) menjadi 27,0 kWh (dengan otomasi), setara efisiensi energi sebesar 32,2%. Besarnya efisiensi ini bersifat spesifik terhadap kepadatan jadwal ruangan yang diuji, dan penggabungan jadwal yang berdekatan turut mencegah *short cycling* yang justru dapat menurunkan efisiensi energi akibat lonjakan daya saat AC pertama kali dinyalakan.

IV. KESIMPULAN

Sistem IoT untuk pengontrolan dan *monitoring* AC ruang kelas berbasis jadwal kuliah berhasil dikembangkan dengan mengintegrasikan RESTful API, *web dashboard*, perangkat ESP8266 untuk kontrol AC, node ESP32 untuk *monitoring* daya, serta *Automation Engine* berbasis *cron job*. Algoritma *interval merging* digunakan untuk menggabungkan jadwal yang saling tumpang tindih menjadi jendela operasi yang konsisten, sedangkan mekanisme *manual override* memungkinkan pengguna mempertahankan kendali ketika diperlukan. Seluruh sembilan skenario pengujian fungsional berhasil dijalankan (100%), sementara pengujian terhadap delapan *endpoint* API menghasilkan TTFB rata-rata 8,38–27,21 ms dan *throughput* rata-rata 7,47–274,59 kbps.

Pada pengujian awal selama satu hari untuk masing-masing skenario, konsumsi energi kumulatif menurun dari 39,6 kWh pada kondisi *baseline* menjadi 27,0 kWh pada kondisi otomasi, atau berkurang sebesar 12,6 kWh (31,8%). Hasil tersebut menunjukkan potensi otomasi berbasis jadwal dalam mengurangi operasi AC yang tidak diperlukan, terutama selama periode jeda dan setelah kegiatan berakhir. Namun, nilai penghematan tersebut

masih merupakan hasil awal dari skenario pengujian yang terbatas sehingga belum dapat digeneralisasi untuk berbagai pola jadwal dan kondisi operasional.

Penelitian selanjutnya dapat memperluas pengujian selama beberapa hari dengan variasi pola jadwal dan kondisi operasional serta menggunakan analisis statistik untuk mengonfirmasi konsistensi penghematan energi. Evaluasi keandalan sistem pada gangguan jaringan dan skenario multi-gedung, penguatan keamanan RESTful API, serta pengembangan fitur *pre-cooling* juga dapat dilakukan pada pengembangan berikutnya.

REFERENSI

- [1] M. L. da Silva Costa, M. R. Freire, K. F. da Silva, and F. G. M. Freires, "Analysis of Energy Consumption Related to the Use of Air Conditioning in University Buildings in a Tropical Climate," *Revista de Gestão Social e Ambiental*, vol. 16, no. 2, 2022, doi: 10.24857/rgsa.v16n2-019.
- [2] Z. Y. Yap, M. H. Md Saad, S. Sahrani, K. Habib, and A. Hussain, "Internet of Things (IoT) Based Air Conditioner Monitoring System for Intelligent Facility Maintenance," *Jurnal Kejuruteraan*, vol. 35, no. 6, pp. 1487–1500, 2023.
- [3] N. Sugiarta, I. D. M. G. S. Okgiarta, I. M. Sugina, and I. D. G. A. T. Putra, "Monitoring Suhu dan Konsumsi Energi AC Split Berbasis Blynk IoT," *J. Energy, Mater., & Manuf. Technol. (JEMMTEC)*, vol. 4, no. 1, pp. 10–14, Jan. 2025.
- [4] I. M. S. Yasa, P. E. Aridona, K. Parti, K. A. Yasa, and I. P. Sutawinaya, "Sistem Kontrol dan Monitoring Air Conditioner Berbasis IoT di Gedung Utama Balai Pengelolaan Informasi Sumber Daya Kelautan dan Perikanan," *VASTUWIDYA*, vol. 8, no. 2, pp. 90–94, Aug. 2025.
- [5] E. Mabotha, N. E. Mabunda, and A. Ali, "Dynamic RESTful API Framework and Deployment using Docker: Enhancing IoT Deployment Strategies," in *Proc. 2024 IEEE PES/IAS PowerAfrica Conference*, Johannesburg, South Africa, 2024.
- [6] T. Ghazali, G. Ilena, M. Mulyadi, and L. Wijayanti, "Web-Based Air Conditioner Monitoring and Control," *Saudi J. Eng. Technol.*, vol. 7, no. 8, pp. 461–471, Sep. 2022, doi: 10.36348/sjet.2022.v07i08.006.
- [7] S. Sharma and P. Randhawa, "IoT-Powered AC Temperature Management for Eco-Smart Infrastructures," in *Proc. 2nd Int. Conf. Intell. Data Commun. Technol. Internet of Things (IDCIoT)*, 2024, doi: 10.1109/IDCIoT59759.2024.10467797.
- [8] A. A. Pratama, R. P. Pradana, A. A. Kurniasari, A. F. Rosyady, and D. P. S. Setyohadi, "Smart Campus: Design and Implementation of Lights and Air Conditioners Monitoring and Control System," *bit-Tech*, vol. 8, no. 3, 2026, doi: 10.32877/bt.v8i3.3406.
- [9] T.-T. Huynh, T.-B. Nguyen, and H. Ho-Dac, "A Practical Approach for Air-Conditioner Management System toward Power Saving for Universities," in *Proc. 2024 Int. Conf. Adv. Technol. Commun. (ATC)*, 2024.
- [10] Waluyo, A. Widura, F. Hadiatna, and R. S. Fikri, "IoT-Based Air Conditioning Control System for Energy Saving," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 13, no. 1, pp. 8–16, Jan. 2023, doi: 10.18517/ijaseit.13.1.17152.
- [11] J. S. Yalli, M. H. Hasan, and A. A. Badawi, "Internet of Things (IoT): Origins, Embedded Technologies, Smart Applications, and Its Growth in the Last Decade," *IEEE Access*, vol. 12, pp. 91357–91382, 2024, doi: 10.1109/ACCESS.2024.3418995.
- [12] E. Mabotha, N. E. Mabunda, A. Ali, and B. Khan, "Exploring Dynamic RESTful API Implementation in IoT

- Environments Using Docker,” *Sci. Rep.*, vol. 15, art. no. 34267, 2025, doi: 10.1038/s41598-025-16460-0.
- [13] W. Wicoksono, H. A. Mustaqhim, P. P. Anwas, and L. N. L. Badratul, “Performance Comparison of NGINX, Apache, and Lighttpd Using WRK on a Debian,” *bit-Tech*, vol. 8, no. 1, pp. 670-680, Aug. 2025, doi: 10.32877/bt.v8i1.2661.
- [14] PHP.net, “PHP: PDO - Manual,” 2024. [Online]. Available: <https://www.php.net/manual/en/book.pdo.php>
- [15] S. Akinola, “Trends in Open Source RDBMS: Performance, Scalability and Security Insights,” *J. Res. Sci. Eng.*, vol. 6, no. 7, pp. 22–28, 2024, doi: 10.53469/jrse.2024.06(07).05.
- [16] K. Alhajri, W. Alsinan, S. Almuhaishi, F. Alhmood, N. AlJumaia, and A. A. A. Ali, “Analysis and Comparison of Sorting Algorithms (Insertion, Merge, and Heap) Using Java,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 22, no. 12, pp. 197-204, Dec. 2022, doi: 10.22937/IJCSNS.2022.22.12.25.
- [17] J. Mayank and A. Mondal, “A Unified Job Scheduler for Optimization of Different System Performance Metrics,” *Concurrency Computat. Pract. Exper.*, vol. 37, art. no. e70111, 2025, doi: 10.1002/cpe.70111.
- [18] Bootstrap, “Bootstrap Official Documentation, v5.3”, 2024. [Online]. Available: <https://getbootstrap.com/docs/5.3/>
- [19] Chart.js, “Chart.js Documentation, v4.x”, 2024. [Online]. Available: <https://www.chartjs.org/docs/latest/>
- [20] B. Pollard and J. Wagner, “Time to First Byte (TTFB),” *web.dev*, Google, 2024.