BAB 2
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Peneliti sudah menganalisis beberapa topik terkait *Middleware*, protokol *AMQP*, dan sistem pemantauan lingkungan berbasis IoT sebelum merancang sistem dalam penelitian desain ini. Hal ini diperlukan untuk memahami sejauh mana penelitian serupa telah dilakukan dan juga di mana kesenjangan yang masih tersisa sehingga studi ini dapat mengisinya, memastikan adanya kontribusi yang baru.

Rafique et al. (2025) dalam *JOIV: International Journal on Informatics Visualization* melakukan kajian literatur mendalam mengenai protokol *AMQP* dari sisi karakteristik teknis dan tantangan keamanannya. Penelitian tersebut mengidentifikasi beberapa kelemahan pada mekanisme autentikasi *AMQP* dan mengusulkan sejumlah peningkatan untuk memperkuat keamanan transmisi data pada ekosistem IoT. Relevansi penelitian ini terhadap topik yang diangkat terletak pada pemahaman mendalam mengenai cara kerja dan keunggulan *AMQP* sebagai protokol komunikasi berbasis antrian pesan.

Gemirter et al. (2021) dalam prosiding IEEE UBMK melakukan evaluasi komparatif antara protokol *AMQP*, MQTT, dan *HTTP* menggunakan data kota pintar secara langsung dan *real-time*. Pengujian dilakukan pada platform Azure IoT dengan beban trafik yang bervariasi. Hasil penelitian menunjukkan bahwa *AMQP* memiliki keunggulan dalam hal kemampuan distribusi pesan ke banyak penerima sekaligus, dengan rata-rata latensi lebih rendah 1,5 ms dibandingkan protokol lain, serta penggunaan CPU yang lebih efisien dibandingkan HTTP. Studi ini menjadi landasan pemilihan *AMQP* sebagai protokol *Middleware* pada penelitian ini sekaligus pembanding performa sistem yang dibangun. Berikut ini perbandingan penelitian terdahulu dengan penelitian yang dilakukan, yang dapat dilihat pada Tabel 2.1.

Akanbi dan Masinde (2020) dalam jurnal *Sensors* (MDPI) mengembangkan *framework Middleware* terdistribusi untuk pemrosesan data *real-time* dari sistem *monitoring* lingkungan yang heterogen. Penelitian itu memperlihatkan bahwa *Middleware* yang dirancang dengan baik mampu menjembatani perbedaan format data dari berbagai sensor sehingga data bisa dianalisis secara seragam di sisi *server*. Kontribusi utama penelitian tersebut bagi penelitian ini adalah konsep integrasi data sensor melalui lapisan *Middleware* yang terstruktur.

Alghazzawi et al. (2021) dalam *PeerJ Computer Science* mengembangkan *Middleware* ScaleUp yang dirancang agar bisa bekerja dengan berbagai jenis perangkat IoT tanpa modifikasi kode yang berarti. Penelitian ini memberikan gambaran tentang pentingnya fleksibilitas dan skalabilitas dalam desain *Middleware*, dua aspek yang juga menjadi perhatian dalam penelitian ini.

Idrees et al. (2018) dalam *Sensors* (MDPI) membangun sistem *monitoring* kualitas udara berbasis IoT dengan pendekatan *edge computing* menggunakan mikrokontroler berbiaya rendah. Hasilnya menunjukkan bahwa pemrosesan data di sisi perangkat sebelum dikirim ke *server* mampu mengurangi beban jaringan secara signifikan. Penelitian ini menjadi referensi dalam pemilihan perangkat keras sensor dan arsitektur sistem yang sesuai.

Nkosi et al. (2025) dalam *IAR Journal of Engineering and Technology* mengembangkan sistem *monitoring* lingkungan dalam ruangan berbasis IoT yang menggunakan MQTT sebagai protokol komunikasinya. Sistem yang dibangun mencatat keandalan pengiriman pesan sebesar 99,8% dan mampu mendeteksi anomali kondisi ruangan 20–40 menit sebelum kondisi kritis. Penelitian ini menjadi pembanding langsung bagi sistem yang akan dibangun, sekaligus memperkuat argumen bahwa protokol berbasis antrian pesan termasuk *AMQP* sangat layak digunakan untuk kebutuhan *monitoring* ruangan secara *real-time*.

Berikut ini merupakan perbandingan penelitian terdahulu dengan penelitian yang dilakukan, yang dapat dilihat pada Tabel 2.1.

Tabel 2. 1 Perbandingan penelitian terdahulu

Kategori	**Peneliti 1 Gemirter et al. (2021)**	**Peneliti 2 Alghazzawi et al. (2021)**	**Peneliti 3 Nkosi et al. (2025)**	**Penelitian Saat Ini**
Fokus Penelitian	Perbandingan performa protokol *AMQP*, MQTT, dan HTTP	*Middleware* IoT yang scalable dan interoperable	*Monitoring* lingkungan indoor berbasis IoT	Implementasi *Middleware AMQP* pada sistem *monitoring* lingkungan dalam ruangan
Metode	Eksperimen komparatif	*Middleware ScaleUp*	Implementasi IoT berbasis MQTT	Implementasi *RabbitMQ* berbasis *AMQP* dengan *Flask Server*
Protokol	*AMQP*, MQTT, HTTP	*Middleware* IoT	MQTT	*AMQP*
Perangkat	*Azure IoT Platform*	Multi-device IoT	Sensor IoT	*ESP32*, Sensor *DHT11*, *MQ-2, RabbitMQ, Flask, Dashboard Web*
Parameter Pengujian	*Latency* dan *CPU Usage*	Skalabilitas dan interoperabilitas	Pengujian pengiriman pesan	*Latency, Throughput, Delay*, Akurasi Sensor.
Hasil Penelitian	*AMQP* memiliki *latency* lebih rendah dan CPU lebih efisien	*Middleware* mampu bekerja pada berbagai perangkat IoT	Tingkat keberhasilan pengiriman pesan mencapai 99,8%	Sistem berhasil melakukan *monitoring* suhu dan kelembapan secara *real-time* menggunakan *Middleware AMQP*
Perbedaan dengan Penelitian Saat Ini	Hanya membandingkan protokol, belum	Tidak mengimplementasikan *monitoring* suhu	Menggunakan MQTT sebagai protokol	Mengimplementasikan *RabbitMQ* berbasis *AMQP* pada sistem

Kategori	Peneliti 1 Gemirter et al. (2021)	Peneliti 2 Alghazzawi et al. (2021)	Peneliti 3 Nkosi et al. (2025)	Penelitian Saat Ini
	membangun sistem *monitoring* secara nyata	dan kelembapan serta pengujian QoS	komunikasi, bukan *AMQP*	*monitoring* lingkungan menggunakan *ESP32*, *DHT11*, *Flask Server*, dan *Web Dashboard* serta melakukan pengujian QoS (*latency, throughput, delay*), akurasi sensor, dan pengujian fungsional sistem.

Berdasarkan hasil kajian dari beberapa penelitian terdahulu pada Tabel 2.1, dapat diketahui bahwa penelitian mengenai sistem *monitoring* berbasis *Internet of Things* (IoT), *Middleware RabbitMQ,* serta penggunaan protokol *AMQP* dan MQTT telah banyak dilakukan. Penelitian-penelitian tersebut menunjukkan bahwa penggunaan *Middleware* dan *message queue* mampu meningkatkan pengujian pengiriman data dan mendukung proses *monitoring* secara *real-time.*

Pada penelitian ini, sistem yang dirancang menggunakan *ESP32* dan sensor *DHT11* untuk melakukan *monitoring* suhu dan kelembapan lingkungan secara *real-time* dengan bantuan *Middleware RabbitMQ* berbasis protokol *AMQP*. Data yang diperoleh kemudian ditampilkan melalui *Dashboard Web* sehingga proses *monitoring* dapat dilakukan dengan lebih mudah. Selain itu, penelitian ini juga melakukan pengujian performa jaringan menggunakan parameter *Quality of Service* (QoS) seperti *delay, latency*, dan *throughput* untuk mengetahui kualitas komunikasi data pada sistem yang dibangun.

Dengan adanya penelitian ini, diharapkan dapat memberikan pengembangan terhadap penelitian sebelumnya serta menghasilkan sistem *monitoring* lingkungan yang lebih stabil, *real-time*, dan mudah digunakan.

2.2 Landasan Teori

2.2.1 Monitoring Lingkungan

Monitoring lingkungan merupakan kegiatan pengamatan terhadap kondisi lingkungan untuk mengetahui perubahan parameter tertentu, seperti suhu dan kelembapan udara. Perkembangan teknologi saat ini membuat proses *monitoring* dapat dilakukan secara otomatis dengan memanfaatkan *Internet of Things* (IoT). Menurut penelitian Nkosi et al. (2025), sistem *monitoring* lingkungan berbasis IoT mampu melakukan pengiriman data secara *real-time* dengan tingkat keberhasilan mencapai 99,8%. Selain itu, penelitian tersebut menunjukkan bahwa sistem *monitoring* dapat membantu mendeteksi perubahan kondisi lingkungan lebih cepat sehingga proses pengawasan menjadi lebih efektif. Yang dapat dilihat pada Gambar 2.1.

Gambar 2. 1 *Environtmental Monitoring*
(Sumber: https://www.istockphoto.com)

Dari Gambar 2.1, sistem *monitoring* lingkungan dirancang menggunakan *ESP32* dan sensor *DHT11* untuk membaca data suhu dan kelembapan secara *real-time*. Data hasil pembacaan sensor kemudian dikirim menggunakan *Middleware RabbitMQ* berbasis protokol *AMQP* dan ditampilkan pada *Dashboard Web monitoring*. Penggunaan *Middleware* pada sistem *monitoring* didukung oleh penelitian Rahman dan Sumardi (2021) yang menyatakan bahwa penggunaan *RabbitMQ* mampu meningkatkan reliabilitas pengiriman data pada sistem IoT. Dengan adanya *Dashboard Web*, pengguna dapat memantau kondisi lingkungan secara lebih mudah, menarik, dan dapat dilakukan dari jarak jauh melalui jaringan internet.

2.2.2 Internet of Things (IoT)

Internet of Things (IoT) secara sederhana bisa dipahami sebagai konsep yang memungkinkan berbagai perangkat elektronik saling terhubung dan bertukar data

melalui internet secara otomatis. Perangkat yang dimaksud bisa bermacam-macam, mulai dari sensor kecil yang dipasang di dinding ruangan, hingga perangkat yang lebih kompleks seperti kamera atau aktuator. Yang membuat IoT menarik adalah kemampuannya mengumpulkan data dari lingkungan fisik secara terus-menerus tanpa perlu ada orang yang mengeceknya secara langsung. Idrees et al. (2018) dalam jurnal Sensors (MDPI) menyebutkan bahwa perkembangan IoT telah membuka peluang besar untuk membangun sistem *monitoring* lingkungan yang bisa berjalan secara otomatis dengan biaya yang jauh lebih terjangkau dibandingkan metode konvensional.

Teknologi IoT sendiri sudah banyak dibuktikan keandalannya dalam berbagai penelitian di bidang *monitoring*. Nkosi et al. (2025) dalam IAR *Journal of Engineering and Technology* misalnya, berhasil membangun sistem *monitoring* lingkungan berbasis IoT yang mampu bekerja stabil selama 30 hari penuh dengan tingkat keberhasilan pengiriman data mencapai 99,8%. Menariknya, sistem tersebut juga bisa mendeteksi perubahan kondisi lingkungan 20 hingga 40 menit sebelum kondisi benar-benar kritis sesuatu yang tidak mungkin dilakukan dengan pemantauan manual biasa. Hasil ini menunjukkan bahwa IoT bukan hanya soal menghubungkan perangkat ke internet, tapi juga soal bagaimana data yang masuk bisa diolah dengan cepat dan tepat untuk menghasilkan informasi yang benar-benar berguna.

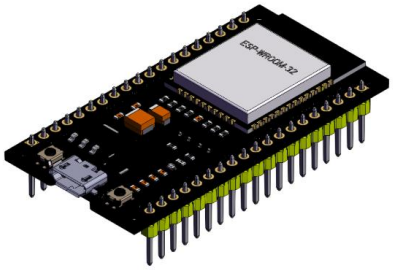
Pada penelitian ini, konsep IoT diterapkan untuk memantau kondisi lingkungan di dalam ruangan secara *real-time* menggunakan sensor *DHT11* dan mikrokontroler *ESP32*. Sensor membaca suhu dan kelembapan ruangan secara berkala, lalu *ESP32* mengirimkan data tersebut ke *server* melalui jaringan *WiFi*. Namun seperti yang disampaikan Akanbi dan Masinde (2020) dalam jurnal Sensors (MDPI), mengelola data dari banyak sensor IoT secara bersamaan bukan perkara mudah dibutuhkan infrastruktur komunikasi yang dirancang dengan baik agar data bisa mengalir lancar tanpa ada yang hilang di tengah jalan.

2.2.3 ESP32

ESP32 adalah mikrokontroler yang cukup populer di kalangan pengembang sistem IoT karena kemampuannya yang terbilang lengkap untuk sebuah chip berukuran kecil. Berbeda dengan mikrokontroler biasa, *ESP32* sudah dilengkapi

modul *WiFi* dan *Bluetooth* secara bawaan sehingga tidak perlu menambahkan modul komunikasi eksternal lagi. Hal ini tentu menghemat biaya dan memperkecil ukuran perangkat secara keseluruhan. Selain itu, jumlah pin GPIO yang dimilikinya cukup banyak, sehingga bisa dihubungkan ke berbagai sensor maupun perangkat tambahan sesuai kebutuhan sistem.

Dari sisi performa komunikasi, *ESP32* juga menunjukkan hasil yang memuaskan ketika digunakan pada sistem *monitoring* berbasis IoT. Ismail dan Rahman (2022) dalam penelitiannya melaporkan bahwa *ESP32* mampu menangani transmisi data IoT secara *real-time* dengan tingkat keberhasilan pengiriman mencapai 99,2% dalam kondisi jaringan *WiFi* yang stabil. Angka ini cukup meyakinkan untuk kebutuhan sistem *monitoring* yang mengharuskan data sampai ke *server* tanpa banyak kegagalan. Yang dapat dilihat pada Gambar 2.2 dibawah ini.



Gambar 2. 2 *ESP32*
(Sumber: https://embeddednesia.com)

Dari Gambar 2.2 *ESP32* difungsikan sebagai pusat kendali sistem *monitoring* lingkungan dalam ruangan. Tugasnya adalah membaca data suhu dan kelembapan dari sensor *DHT11* secara berkala, lalu mengirimkan data tersebut ke *Flask Server* melalui jaringan *WiFi* menggunakan metode *HTTP POST.* Data yang sudah sampai di *server* kemudian akan diteruskan ke *message broker RabbitMQ* sebelum akhirnya ditampilkan di *Dashboard Web* secara *real-time*. *ESP32* dipilih karena ukurannya yang ringkas, konsumsi dayanya yang rendah, dan sudah terbukti andal untuk keperluan sistem *monitoring* berbasis IoT.

2.2.4 Sensor DHT11

Sensor *DHT11* adalah sensor digital yang paling banyak ditemui dalam proyek-proyek *monitoring* suhu dan kelembapan, terutama di kalangan pengembang sistem IoT pemula maupun profesional. Alasannya cukup sederhana:

sensor ini harganya murah, mudah digunakan, dan langsung menghasilkan data dalam format digital sehingga tidak perlu rangkaian pengkondisi sinyal tambahan. Cukup hubungkan ke pin GPIO mikrokontroler, dan sensor sudah siap membaca kondisi udara di sekitarnya.

Cara kerja sensor *DHT11* sebenarnya tidak terlalu rumit. Sensor ini mendeteksi perubahan suhu dan kelembapan di lingkungan sekitarnya menggunakan elemen resistif untuk kelembapan dan termistor untuk suhu, lalu mengubah hasil deteksi tersebut menjadi sinyal digital yang dikirimkan ke mikrokontroler melalui antarmuka satu kabel. Rahmawati dan Sujono (2026) dalam jurnal *Jurnal Manajemen Sistem Informasi* (JMASIF) membuktikan bahwa sensor *DHT11* yang diintegrasikan dengan *ESP32* mampu membaca data suhu dan kelembapan secara stabil dalam kondisi lingkungan ruangan dengan tingkat keberhasilan transmisi data mencapai 98%, meskipun sensor ini memiliki keterbatasan akurasi pada kondisi kelembapan yang sangat tinggi. Yang dapat dilihat pada Gambar 2.3.

Gambar 2. 3 *DHT11*
(Sumber: https://jagorobotik.com)

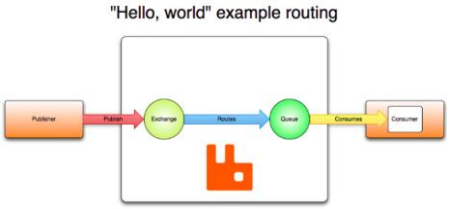
Pada gambar 2.3 sensor *DHT11* digunakan untuk membaca kondisi suhu dan kelembapan ruangan yang kemudian diproses oleh *ESP32.* Hasil pembacaan sensor dikirim ke *server* dan ditampilkan pada *Dashboard monitoring* berbasis *Web*. Pemilihan sensor *DHT11* didasarkan pada beberapa pertimbangan praktis: konsumsi dayanya rendah, harganya terjangkau, dan dihubungkan dengan *ESP32* sangat mudah dilakukan.

2.2.5 RabbitMQ

RabbitMQ adalah perangkat lunak message broker yang cukup banyak digunakan dalam arsitektur sistem terdistribusi, termasuk pada sistem IoT. Secara

sederhana, *RabbitMQ* berperan sebagai perantara yang menerima pesan dari pengirim, menyimpannya sementara dalam antrian, lalu meneruskannya ke penerima sesuai aturan yang telah dikonfigurasi. Mekanisme antrian ini membuat komunikasi data menjadi lebih teratur dan tidak bergantung pada ketersediaan penerima secara bersamaan dengan pengirim. Salah satu kelebihan *RabbitMQ* yang cukup menonjol dalam konteks IoT adalah kemampuannya menyimpan pesan secara sementara ketika penerima sedang tidak aktif atau koneksi internet sedang terganggu.

Penelitian dalam Jurnal Informatika dan Sains (JISA) Universitas Trilogi menjelaskan bahwa implementasi *RabbitMQ* sebagai message broker pada sistem IoT terbukti mampu menjaga kestabilan proses pengiriman data meskipun koneksi internet sesekali tidak stabil, karena semua pesan yang belum tersampaikan akan tetap tersimpan di antrian dan dikirim ulang secara otomatis ketika koneksi pulih kembali. Yang dapat dilihat pada Gambar 2.4 dibawah ini.



Gambar 2. 4 *RabbitMQ*

(Sumber: *https://www.RabbitMQ.com*)

Pada Gambar 2.4 *RabbitMQ* digunakan sebagai *Middleware* yang menghubungkan *Flask Server* dengan *Dashboard Web monitoring*. Data suhu dan kelembapan yang diterima dari *ESP32* oleh *Flask Server* akan dipublikasikan ke antrian *RabbitMQ* sebelum diteruskan ke *consumer* yang bertugas menyimpan data ke database dan menampilkannya di *Dashboard. RabbitMQ* dipilih karena kemampuannya mendistribusikan satu pesan ke beberapa tujuan sekaligus, mendukung konfigurasi antrian yang persisten, dan memiliki antarmuka manajemen berbasis *Web* yang memudahkan pemantauan kondisi sistem secara langsung.

2.2.6 *Advanced Message Queuing Protocol* (*AMQP*)

Advanced Message Queuing Protocol (*AMQP*) adalah protokol komunikasi standar terbuka yang dirancang khusus untuk keperluan *message-oriented Middleware*. Protokol ini bekerja di atas lapisan TCP dan mendefinisikan cara pesan dibuat, dikirim, diantrekan, dan diterima antar komponen dalam sebuah sistem terdistribusi. Yang membedakan *AMQP* dari protokol komunikasi lain adalah jaminan pengirimannya yang bisa dikonfigurasi sesuai kebutuhan, mulai dari memaksimalkan, meminimalkan, hingga sesuai dengan kebutuhan, dengan demikian kerahasian data bisa dijaga sesuai tingkat keandalan yang diinginkan.antar perangkat agar data yang dikirim dapat diterima dengan baik tanpa mengalami kehilangan informasi.

Perbandingan antara *AMQP* dengan protokol komunikasi IoT lainnya sudah pernah dilakukan secara langsung oleh Gemirter et al. (2021) dalam konferensi IEEE UBMK. Pengujian dilakukan menggunakan data kota pintar secara *real-time* dan hasilnya cukup menarik: *AMQP* memiliki rata-rata latensi 1,5 ms lebih rendah dibandingkan *HTTP* dan terbukti jauh lebih efisien dalam hal penggunaan CPU, dengan konsumsi hanya 2,13% dibanding *HTTP* yang mencapai 6,36%. Lebih dari itu, *AMQP* juga unggul dalam kemampuan mendistribusikan pesan ke banyak penerima sekaligus berkat mekanisme *exchange* dan *routing* yang fleksibel—fitur yang tidak dimiliki secara native oleh *HTTP* maupun MQTT. Yanga dapat dilihat pada Gambar 2.5.

Gambar 2. 5 *AMQP*
(Sumber: https://www.cleo.com)

Pada Gambar 2.5 Protokol *AMQP* digunakan sebagai protokol komunikasi antara *Flask Server* dan *RabbitMQ*. Setiap data suhu dan kelembapan yang diterima dari *ESP32* akan dipublikasikan ke *exchange RabbitMQ* melalui koneksi *AMQP* yang diamankan dengan enkripsi TLS pada port 5671, kemudian didistribusikan ke antrian-antrian yang telah dikonfigurasi.

Pemilihan *AMQP* didasarkan pada keunggulannya dalam menjamin keutuhan pesan, kemampuan distribusi multi-penerima, serta dukungannya terhadap sistem *monitoring real-time* yang membutuhkan latensi rendah.

2.2.7 Erlang

Erlang adalah bahasa pemrograman yang awalnya dikembangkan oleh perusahaan telekomunikasi Ericsson pada tahun 1980-an untuk menangani sistem komunikasi yang harus beroperasi tanpa henti. Tidak seperti bahasa pemrograman pada umumnya, Erlang dirancang dari awal untuk mendukung *concurrent processing*, yaitu kemampuan menjalankan ribuan proses secara bersamaan dalam satu mesin tanpa saling mengganggu. Selain itu, Erlang juga memiliki mekanisme *fault tolerance* bawaan yang memungkinkan sistem terus berjalan meskipun salah satu prosesnya mengalami kegagalan sistem akan secara otomatis memulihkan proses yang bermasalah tanpa harus mematikan keseluruhan layanan.

Kaitan Erlang dalam penelitian ini terletak pada penggunaannya sebagai fondasi *RabbitMQ*. Rafique et al. (2025) dalam *JOIV: International Journal on Informatics Visualization* menjelaskan bahwa *RabbitMQ* dibangun menggunakan Erlang justru karena karakteristik bahasa ini sangat cocok untuk kebutuhan *message broker* yang harus mampu menangani banyak koneksi dan pesan secara bersamaan dengan keandalan tinggi. Kemampuan *concurrent processing* Erlang memungkinkan *RabbitMQ* melayani banyak *publisher* dan *consumer* sekaligus tanpa penurunan performa yang penting.

Erlang tidak digunakan secara langsung dalam penulisan kode program. Namun kehadirannya sangat terasa melalui *RabbitMQ* yang menjadi tulang punggung komunikasi data sistem *monitoring* ini. Berkat fondasi Erlang, *RabbitMQ* pada penelitian ini mampu menangani distribusi pesan dari *Flask Server* ke beberapa *consumer* sekaligus dengan stabil, bahkan ketika salah satu *consumer* sesaat tidak aktif, sistem tetap berjalan *NORMAL* tanpa kehilangan data.

2.2.8 Web Dashboard

Web Dashboard adalah antarmuka berbasis *Web* yang menampilkan data dari sistem *monitoring* dalam bentuk visual yang mudah dipahami pengguna.

Dibandingkan melihat data mentah berupa teks atau angka dari terminal, *Dashboard* menyajikan informasi dalam bentuk grafik, *gauge*, dan indikator status yang bisa langsung dimengerti dalam hitungan detik. Dalam sistem *monitoring* berbasis IoT, keberadaan *Dashboard* sangat penting karena memungkinkan pengguna memantau kondisi perangkat atau lingkungan dari mana saja hanya melalui *browser* tanpa perlu berada di lokasi fisik sensor.

Penggunaan *Dashboard* pada sistem *monitoring* IoT sudah terbukti meningkatkan efektivitas pengawasan secara signifikan. Ismail dan Rahman (2022) dalam penelitiannya menunjukkan bahwa visualisasi data *monitoring* dalam bentuk grafik historis pada *Dashboard Web* memberikan gambaran tren kondisi lingkungan yang memudahkan pengguna dalam mengidentifikasi pola perubahan dan mengambil keputusan lebih cepat dibandingkan harus membaca log data secara manual.

Web Dashboard dikembangkan menggunakan *framework Flask* dan menampilkan data suhu serta kelembapan yang diperoleh dari sensor *DHT11* melalui *ESP32* secara *real-time*. Selain menampilkan nilai sensor terkini dalam bentuk *gauge*, *Dashboard* juga menampilkan grafik tren historis dan parameter *Quality of Service* (QoS) seperti *latency,delay* dan *throughput* untuk memantau performa komunikasi sistem secara langsung. Dengan adanya *Dashboard* ini, pengguna bisa memantau kondisi lingkungan ruangan kapan saja dan dari mana saja tanpa perlu mengecek perangkat secara langsung.

2.2.9 Flask Framework

Flask framework berbasis bahasa pemrograman *Python* yang digunakan untuk membangun aplikasi *Web*. Dibandingkan *framework* lainnya, *Flask* tergolong ringan dan minimalis karena hanya menyediakan komponen-komponen dasar yang dibutuhkan, sementara komponen tambahan bisa dipasang sesuai kebutuhan pengembang. Walingkas dan Saian (2023) dalam *Jurnal JTIK* menjelaskan bahwa *Flask* menggunakan konsep *routing* berbasis *decorator* yang memudahkan pengembang dalam menentukan alur logika aplikasi dan pengelolaan *endpoint* URL secara terstruktur, sehingga sesuai digunakan untuk membangun aplikasi *Web* berskala kecil hingga menengah secara efisien. Yang dapat dilihat pada Gambar 2.6.

Gambar 2. 6 *Flask Framework*
(Sumber: https://levelup.gitconnected.com)

Pada Gambar 2.6, *Flask* sering digunakan sebagai *Web server* yang menerima data dari perangkat keras melalui *endpoint* API, lalu memprosesnya dan menampilkannya ke pengguna melalui antarmuka *Web*. Kemampuan *Flask* dalam menangani permintaan *HTTP* secara *real-time* dan kompatibilitasnya dengan berbagai pustaka *Python* menjadikannya pilihan yang tepat untuk sistem *monitoring* berbasis IoT.*Flask* digunakan sebagai *Web server* sekaligus lapisan *Middleware* yang menerima data sensor dari *ESP32*, memvalidasinya, lalu mempublikasikannya ke *RabbitMQ* melalui protokol *AMQP. Flask* juga bertugas menyajikan *Dashboard monitoring* kepada pengguna melalui antarmuka *HTML*, CSS, dan *JavaScript* yang menampilkan nilai suhu, kelembapan, status ruangan, dan parameter QoS secara *real-time*.

2.2.10 Arduino IDE

Arduino IDE merupakan perangkat lunak yang digunakan untuk menulis, mengompilasi, dan mengunggah program ke berbagai jenis mikrokontroler, termasuk *ESP32.* Tampilannya yang sederhana dengan editor kode, panel *output*, dan *serial monitor* dalam satu jendela membuat *Arduino IDE* mudah digunakan bahkan oleh pemula sekalipun. Pramuditya et al. (2023) dalam jurnal *Sensors* (MDPI) menjelaskan bahwa *Arduino IDE* memiliki komunitas pengguna yang besar dan aktif dengan dokumentasi daring yang lengkap, serta tersedia ribuan pustaka (*library*) pendukung yang bisa dipasang langsung melalui *Library Manager* bawaan untuk mempercepat pengembangan proyek IoT berbasis *ESP32.* Salah satu fitur yang paling sering dimanfaatkan dalam pengembangan sistem *monitoring* adalah *serial monitor* bawaan *Arduino IDE*.

Fitur ini memungkinkan pengembang melihat data yang dikirimkan oleh mikrokontroler secara langsung melalui koneksi *serial*, sehingga sangat membantu dalam proses pengujian dan *debugging* program sebelum sistem dijalankan secara penuh. *Arduino IDE* digunakan untuk memprogram *ESP32* yang bertugas sebagai pengendali utama sistem *monitoring* lingkungan. Program yang dibuat mencakup logika pembacaan data dari sensor *DHT11*, penampilan data pada layar LCD, serta pengiriman data ke *Flask Server* melalui jaringan *WiFi*. Arduino IDE dipilih karena kompatibilitasnya yang sudah teruji dengan *ESP32* dan kelengkapan pustaka pendukungnya yang mempercepat proses pengembangan sistem.

2.2.11 Thonny IDE

Thonny IDE merupakan editor program berbasis *Python* yang dikembangkan dengan tujuan utama membuat proses pemrograman *Python* menjadi lebih mudah dan ramah pengguna, khususnya bagi pemula. Berbeda dengan editor lain yang tampilannya sering terasa ramai, *Thonny* dirancang dengan antarmuka yang bersih dan minimalis sehingga pengguna bisa langsung fokus menulis kode tanpa terganggu oleh banyaknya menu dan opsi yang tidak diperlukan. Selain untuk pemrograman *Python* biasa, *Thonny* juga mendukung pengembangan perangkat berbasis *MicroPython* seperti *ESP32*, lengkap dengan fitur koneksi serial untuk mengunggah program langsung ke perangkat.

Penggunaan *Thonny* dipilih karena merupakan editor *Python* yang ringan dan mudah dioperasikan. Menurut Walingkas dan Saian (2023) dalam Jurnal JTIK, *Python* merupakan bahasa pemrograman yang fleksibel dan banyak digunakan untuk membangun aplikasi berbasis *Web* maupun sistem *monitoring* karena sintaksnya yang sederhana dan dukungan pustakanya yang lengkap. Kemampuan Thonny dalam menangani koneksi ke perangkat mikrokontroler secara langsung dari antarmukanya menjadi keunggulan tersendiri. Pengguna bisa melihat *output* dari perangkat, menjalankan kode secara langsung di *shell MicroPython,* hingga mengelola file yang ada di dalam memori *ESP32* semuanya dari satu jendela yang sama tanpa perlu berpindah aplikasi.

2.2.12 TP4056

TP4056 merupakan adalah modul pengisian daya baterai lithium yang banyak ditemui pada berbagai proyek elektronik portabel.

Modul ini berukuran sangat kecil namun sudah lengkap dengan sirkuit pengisian daya yang cukup cerdas yang menggunakan metode *constant current/constant voltage* yang memastikan baterai diisi dengan aman tanpa risiko kelebihan daya. Beberapa varian TP4056 bahkan sudah dilengkapi dengan rangkaian proteksi yang melindungi baterai dari kondisi *overcharge*, *overdischarge*, dan *short circuit*.Cara kerja TP4056 cukup mudah. Ketika sumber daya dari USB atau adaptor terhubung, modul ini langsung mengalirkan arus ke baterai lithium dengan besar arus yang dikontrol secara otomatis.

Ketika *voltase* baterai sudah mencapai level penuh (biasanya 4,2V), modul secara otomatis mengurangi arus pengisian hingga nol untuk menghindari kerusakan baterai. Indikator LED merah dan biru pada modul memudahkan pengguna mengetahui status pengisian tanpa perlu mengukur *voltase* secara manual. Yang dapat dilihat pada Gambar 2.7 dibawah ini.

Gambar 2. 7 *TP4056*

(Sumber: https://quartzcomponents.com)

Pada Gambar 2.7 TP4056 digunakan modul pengisian daya untuk baterai yang menjadi sumber energi cadangan perangkat *monitoring* berbasis *ESP32.* Penggunaannya memungkinkan perangkat bekerja secara portabel tanpa harus selalu terhubung ke sumber listrik utama. TP4056 dipilih karena ukurannya yang kecil, harganya yang terjangkau, dan kemudahan integrasinya dengan sistem elektronik berbasis baterai lithium.

2.2.13 MT3608

MT3608 merupakan modul *DC-DC converter* tipe *step-up* yang berfungsi menaikkan tegangan masukan menjadi tegangan keluaran yang lebih tinggi sesuai kebutuhan. Cara kerjanya memanfaatkan prinsip induksi elektromagnetik melalui rangkaian *boost converter* energi disimpan sementara di induktor lalu dilepaskan ke keluaran dengan voltase yang lebih tinggi dari masukan.

MT3608 mampu menerima tegangan masukan mulai dari 2V hingga 24V dan menghasilkan tegangan keluaran yang bisa diatur hingga 28V dengan efisiensi konversi yang cukup baik. Yang dapat diihat pada Gambar 2.8 dibawah ini.

Gambar 2. 8 MT3608

(Sumber: https://components101.com)

Pada Gambar 2.8 MT3608 banyak digunakan pada proyek IoT berbasis baterai adalah kemampuannya menjaga kestabilan tegangan keluaran meskipun voltase baterai terus menurun seiring pemakaian. Ini penting karena tegangan baterai lithium bisa turun dari 4,2V hingga sekitar 3V selama digunakan, sementara *ESP32* membutuhkan tegangan operasi yang stabil di sekitar 3,3V. Dengan MT3608, voltase keluaran tetap terjaga stabil meski baterai sudah mulai habis.
MT3608 digunakan untuk menaikkan tegangan dari baterai menuju tegangan kerja yang dibutuhkan *ESP32* dan komponen lainnya. Modul ini bekerja bersama TP4056 dalam sirkuit manajemen daya perangkat *monitoring* TP4056 mengurus pengisian baterai saat ada sumber listrik, sementara MT3608 memastikan tegangan yang dihasilkan baterai cukup dan stabil untuk menjalankan seluruh komponen sistem.

2.2.14 Higrometer

Hygrometer adalah alat ukur yang digunakan untuk mengetahui tingkat kelembapan udara di suatu lingkungan. Alat ini sudah lama digunakan di berbagai bidang, mulai dari meteorologi, pertanian, hingga industri manufaktur yang sensitif terhadap kelembapan. Dalam sistem *monitoring* lingkungan, higrometer berperan sebagai alat referensi pembanding untuk mengetahui seberapa akurat sensor yang digunakan dalam sistem. Prinsip kerja higrometer bervariasi tergantung jenisnya. Higrometer digital seperti HTC-1 yang digunakan dalam penelitian ini menggunakan sensor resistif atau kapasitif untuk mendeteksi perubahan kadar uap air di udara, lalu menampilkan hasilnya secara langsung di layar dalam satuan persen kelembapan relatif (%RH).

Nkosi et al. (2025) dalam *IAR Journal of Engineering and Technology* menyebutkan bahwa pengukuran kelembapan udara secara akurat dan kontinu merupakan salah satu parameter kunci dalam sistem *monitoring* lingkungan yang efektif, terutama untuk mendeteksi kondisi ruangan yang berpotensi menimbulkan ketidaknyamanan atau kerusakan pada perangkat elektronik. Yang dapat dilihat pada Gambar 2.9 dibawah ini.

Gambar 2. 9 *Hygrometer*
(Sumber: https://id.my-best.com)

Pada Gambar 2.9 Higrometer digital HTC-1 digunakan sebagai alat ukur referensi untuk memvalidasi akurasi pembacaan sensor *DHT11* pada sistem yang dibangun. Data dari HTC-1 dibandingkan langsung dengan data yang ditampilkan oleh sistem *monitoring* untuk menghitung nilai error dan tingkat akurasi sensor. Pemilihan HTC-1 sebagai referensi didasarkan pada ketersediaannya yang mudah, harga yang terjangkau, serta akurasinya yang cukup memadai untuk keperluan validasi sensor pada sistem *monitoring* lingkungan dalam ruangan.

2.2.15 MQ-2

Sensor MQ-2 adalah sensor gas semikonduktor yang dirancang untuk mendeteksi keberadaan berbagai jenis gas berbahaya dan asap di lingkungan sekitarnya. Tidak seperti sensor suhu atau kelembapan yang menghasilkan data secara langsung, sensor MQ-2 bekerja berdasarkan prinsip perubahan konduktivitas pada bahan sensitif yang terbuat dari timah dioksida (SnO_2). Pada kondisi udara *NORMAL* dan bersih, tingkat konduktivitas SnO_2 cenderung rendah.

Namun ketika sensor terpapar gas target seperti LPG, metana (CH_4), asap, hidrogen, atau karbon monoksida (CO), konduktivitasnya akan meningkat secara proporsional seiring naiknya konsentrasi gas yang terdeteksi. Perubahan

penghantaran ini kemudian dikonversi menjadi sinyal tegangan analog yang dapat dibaca oleh mikrokontroler melalui pin ADC (*Analog to Digital Converter*), lalu diolah menjadi nilai dalam satuan PPM (*Parts Per Million*) sebagai representasi konsentrasi gas di udara. Sensor MQ-2 banyak dipilih dalam sistem *monitoring* kualitas udara berbasis IoT karena beberapa keunggulan praktis yang dimilikinya. Selain mampu mendeteksi berbagai jenis gas sekaligus dalam satu modul yang ringkas, sensor ini juga memiliki harga yang sangat terjangkau dan mudah diintegrasikan dengan berbagai jenis mikrokontroler yang umum digunakan dalam pengembangan sistem IoT. Easterline et al. (2024) dalam *Procedia Computer Science* membuktikan hal ini melalui penelitiannya yang mengembangkan sistem *monitoring* kualitas udara berbasis IoT menggunakan ESP32 dan sensor MQ-2 untuk mendeteksi kadar gas berbahaya secara *real-time*. Dalam penelitian tersebut, pin analog keluaran sensor dihubungkan ke pin ADC pada ESP32, dan data berhasil dikirimkan ke platform *monitoring* berbasis *Web* dengan pembacaan yang akurat dan responsif terhadap perubahan kondisi udara di lingkungan dalam ruangan. Yang dapat dilihat pada Gambar 2.10 dibawah ini.

Gambar 2. 10 *Sensor MQ-2*
(Sumber: https://ai.thestempedia.com)