# Tugas Besar

# Perancangan Platform IoT

# II2210 - Teknologi Platform

# Incident Response Information System


Dipersiapkan oleh:

Vandega - 18223010

Keane Lim- 18223056

M Khalfani Shaquille - 18223104

https://drive.google.com/drive/folders/1gHeuxWhP8tCal837KLrQeOVurMDBRz05?usp=drive_link

# Pendahuluan

Kebakaran di area perkantoran merupakan risiko serius yang dapat menyebabkan kerugian besar, baik dari sisi material maupun keselamatan jiwa. Peraturan Menteri PUPR No. 26/PRT/M/2008 menekankan bahwa setiap bangunan wajib memiliki sistem peringatan dini dan evakuasi yang memadai, namun pada prakteknya, sistem-sistem yang tersedia masih bersifat konvensional dan reaktif.

Ada beberapa permasalahan utama yang perlu diselesaikan. Pertama, kurangnya sistem peringatan dan penanganan kebakaran yang dapat memberikan notifikasi dan informasi evakuasi secara *real-time* berdasarkan kondisi aktual gedung. Kedua, tidak adanya integrasi antara sistem deteksi awal dengan sistem komunikasi darurat seperti notifikasi secara massal dan pelaporan otomatis ke pemadam kebakaran.

Proyek ini bertujuan untuk merancang dan mengimplementasikan sistem peringatan kebakaran berbasis IoT dan AI agentic di lingkungan perkantoran untuk meningkatkan tingkat keamanan dalam lingkup perkantoran terhadap terjadinya kebakaran sehingga keselamatan jiwa dan aset dapat terjaga. Terdapat beberapa manfaat yang diharapkan, yaitu bahwa proyek ini dapat meningkatkan efektivitas deteksi dan respons terhadap kebakaran di lingkungan perkantoran, meminimalisasi risiko korban jiwa dan kerugian aset dengan menyediakan rute evakuasi yang tepat, personal, dan menunjukkan penerapan nyata dari sistem IoT dan AI dalam mitigasi bencana.

Teknologi IoT memiliki peran yang besar dalam menciptakan solusi untuk menyelesaikan permasalahan ini dibandingkan alarm kebakaran konvensional. Hal ini dikarenakan alarm konvensional bersifat isolasi yang berarti ketika satu detektor aktif, maka tidak ada data yang dikirim ke pusat kendali untuk dianalisis secara holistik. Proses pelaporan ke pemadam kebakaran mengenai keadaan dan penyebaran informasi evakuasi masih bergantung berat pada intervensi manusia yang cenderung lebih lambat dan lebih rentan kesalahan di tengah kepanikan terutama kebakaran yang mampu merenggut nyawa. Sebaliknya, solusi berbasis IoT mengubah setiap sensor menjadi sumber data cerdas. Dengan menghubungkan semua sensor ke sebuah platform terpusat, sistem dapat memperoleh gambaran menyeluruh dan real-time

dari kondisi gedung. Kemampuan inilah yang menjadi fondasi untuk integrasi sistem dan notifikasi dinamis—sebuah kapabilitas yang tidak mungkin dicapai oleh arsitektur tradisional. Lebih lanjut, dengan memanfaatkan AI Agentic, data yang terkumpul tidak hanya ditampilkan, tetapi juga dapat dianalisis untuk membuat keputusan otonom, seperti menentukan rute evakuasi teraman secara dinamis berdasarkan penyebaran api, sebuah lompatan besar dari papan evakuasi statis yang ada saat ini

## Metodologi Pengembangan Karya Inovasi

Untuk mengatasi masalah utama, yaitu memberikan peringatan dan panduan evakuasi yang efektif saat terjadi kebakaran, dua alternatif solusi utama dipertimbangkan. Setiap alternatif ditinjau dari sisi perangkat keras (MK II2260) dan perangkat lunak (MK II2210).

---

**Alternatif A: Sistem Peringatan Cerdas Terintegrasi Berbasis Cloud (Solusi yang Dipilih)**

Solusi ini berfokus pada penyediaan instruksi evakuasi yang cerdas, dinamis, dan dipersonalisasi, dengan memanfaatkan tumpukan teknologi modern yang terdiri dari IoT, otomasi alur kerja, dan kecerdasan buatan.

Dari sisi perangkat keras, sistem ini mengandalkan sensor IoT modern dan kamera yang ditempatkan di setiap ruangan. Sensor-sensor ini mampu menyediakan data kuantitatif yang spesifik (misalnya, suhu 60°C, nilai asap 1000) dan mengirimkannya secara *real-time* melalui jaringan Wi-Fi atau Ethernet. Ketika ambang batas bahaya terlampaui, maka AI agent akan mulai melaksanakan alur kerja yang sudah dirancang dan kamera akan mulai mendeteksi jumlah orang pada ruangan secara real time agar sistem mampu mengetahui jumlah orang yang terjebak pada ruangan. Keunggulan utama dari pendekatan ini adalah tersedianya data yang kaya untuk analisis tingkat keparahan yang akurat, meskipun memiliki kelemahan berupa ketergantungan pada stabilitas jaringan di dalam gedung dan pemrosesan data.

Dari sisi perangkat lunak, arsitektur yang digunakan bersifat terdistribusi dan mengorkestrasi beberapa platform. Ketika batas ambang bahaya terlampaui, backend akan memicu dua proses secara paralel yaitu proses alur kerja n8n dan juga proses pendeteksi objek.

Proses pertama adalah mendeteksi jumlah orang berbasis *computer vision.* Secara bersamaan dengan pemicu alur kerja, sistem mengaktifkan subsistem deteksi objek. Proses ini memanfaatkan model machine learning YOLOv8 yang telah di-fine-tuned untuk menganalisis stream video dari kamera yang ditempatkan di titik-titik strategis. Tujuannya adalah untuk mendeteksi dan menghitung jumlah orang di setiap ruangan secara real-time. Data jumlah penghuni ini kemudian disimpan dan diperbarui secara periodik (setiap 5 detik) ke dalam sebuah database SQL, menciptakan sumber data dinamis mengenai tingkat kepadatan di seluruh gedung.

Di waktu yang sama, proses kedua dijalankan secara paralel yaitu alur kerja ai agent. backend memicu alur kerja (workflow) pada platform otomasi n8n. N8n berfungsi sebagai pusat kendali yang mengorkestrasi respons cerdas dengan memanggil Google Gemini API. Berbeda dari pendekatan standar, arsitektur ini menerapkan metode Retrieval-Augmented Generation (RAG). Artinya, AI tidak hanya memproses prompt, tetapi secara aktif mengambil (retrieve) data jumlah penghuni terkini dari database SQL dan menggabungkannya dengan pengetahuan statis mengenai tata letak gedung dari knowledge base PDF. Kombinasi data ini memungkinkan AI menghasilkan analisis dan instruksi evakuasi yang sangat akurat dan sadar-penghuni (occupancy-aware). Hasilnya kemudian disalurkan melalui notifikasi multi-kanal, termasuk panggilan suara AI (AI voice call) untuk memberikan instruksi verbal secara langsung kepada pengguna.

Kelebihan utama dari arsitektur canggih ini adalah kemampuannya untuk memberikan instruksi evakuasi yang sangat relevan dan kontekstual, yang secara dramatis meningkatkan kemungkinan evakuasi yang berhasil dan aman. Namun, kelemahannya adalah peningkatan kompleksitas implementasi yang signifikan. Solusi ini tidak hanya bergantung pada ketersediaan layanan cloud, tetapi juga menuntut pemeliharaan model machine learning (YOLOv8), pengelolaan database SQL, dan implementasi alur kerja RAG yang lebih rumit dibandingkan sistem peringatan standar.

**Alternatif B: Sistem Alarm Kebakaran Konvensional Terpusat**

Solusi ini merepresentasikan pendekatan standar industri yang telah teruji, di mana fokus utamanya adalah deteksi dan peringatan umum tanpa kecerdasan tambahan.

Dari sisi perangkat keras, alternatif ini menggunakan detektor asap dan panas konvensional yang terhubung secara fisik melalui kabel ke sebuah Fire Alarm Control Panel (FACP) pusat, sebagaimana yang juga terpasang di gedung Graha Inovasi. Notifikasi disampaikan melalui perangkat fisik seperti lonceng alarm, lampu strobo, dan sistem pengumuman suara darurat (EVAS). Pendekatan ini menawarkan keandalan yang sangat tinggi karena sifatnya sebagai sistem tertutup yang tidak bergantung pada jaringan Wi-Fi dan telah menjadi standar keamanan gedung. Namun, kelemahannya adalah informasi yang diberikan hanya bersifat zonal (misalnya, "Zona 3, Lantai 2") dan tidak menyediakan data kuantitatif yang spesifik untuk analisis mendalam.

Dari sisi perangkat lunak, logikanya bersifat tertanam (*embedded*) di dalam FACP dengan aturan "IF-THEN" yang kaku, yang memungkinkan respons peringatan yang sangat cepat dengan latensi minimal. Namun, keterbatasan utamanya adalah ketidakmampuan total untuk memberikan panduan evakuasi yang dinamis dan kontekstual. Akibatnya, penghuni gedung sepenuhnya bergantung pada denah evakuasi statis yang terpasang di dinding dan pelatihan yang pernah diterima, tanpa adanya panduan yang relevan dengan lokasi spesifik dari sebuah insiden.

Setelah meninjau kedua alternatif, Alternatif A: Sistem Peringatan Cerdas Terintegrasi Berbasis Cloud dipilih sebagai solusi yang akan dikembangkan.

Keputusan ini didasarkan pada analisis bahwa meskipun Alternatif B (Sistem Konvensional) menawarkan keandalan operasional yang sangat tinggi karena sifatnya yang tertutup, ia secara fundamental tidak mampu mengatasi masalah utama yang telah diidentifikasi: kurangnya panduan evakuasi yang cerdas dan dinamis. Sistem konvensional hanya menyediakan peringatan zonal yang bersifat umum, memaksa penghuni untuk bergantung pada denah statis di tengah situasi darurat yang kacau.

Sebaliknya, Alternatif A adalah satu-satunya pendekatan yang secara langsung menjawab tantangan tersebut. Dengan memanfaatkan data sensor yang kaya dan orkestrasi perangkat lunak cerdas, solusi ini dapat memberikan instruksi yang relevan dan personal, bahkan mengubah rute evakuasi secara real-time sesuai kondisi lapangan. Oleh karena itu, meskipun memiliki tantangan berupa kompleksitas implementasi dan ketergantungan pada jaringan, kapabilitasnya untuk memitigasi risiko korban jiwa secara proaktif dianggap sebagai nilai yang jauh melampaui kelemahannya. Kompleksitas tersebut diterima sebagai trade-off yang diperlukan untuk mencapai lompatan kuantum dalam keselamatan gedung.

## Kebutuhan Sistem dan Spesifikasi Desain

### Functional Requirement

1.1 Pengumpulan dan Transmisi Data

| No | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 1.1.1 | Pendeteksi Konsentrasi Asap | Perangkat Keras (Firmware) | Untuk memberikan deteksi dini terhadap salah satu indikator utama kebakaran, memungkinkan respons yang lebih cepat sebelum api membesar dan asap memenuhi ruangan. |
| 1.1.2 | Pendeteksi Suhu Ruangan | Perangkat Keras (Firmware) | Untuk mengidentifikasi kenaikan suhu abnormal yang menjadi indikator kuat adanya sumber panas atau api, yang seringkali tidak terdeteksi oleh indra manusia pada tahap awal. |
| 1.1.3 | Pendeteksi Pemicu | Perangkat Keras | Untuk menyediakan sarana bagi |

| | | | |
|---|---|---|---|
| | Kepanikan | (Firmware) | penghuni untuk secara manual memicu alarm dalam situasi darurat yang mungkin belum terdeteksi oleh sensor, mengatasi masalah deteksi yang lambat atau kegagalan sensor. |
| 1.1.4 | Komunikasi Otomatis ke Pihak Pemadam Kebakaran | Perangkat Lunak (Platform) | Untuk secara otomatis meneruskan informasi insiden seperti struktur bangunan, lokasi, jumlah orang yang terdeteksi, dan lain lain yang telah terverifikasi kepada pihak berwenang (seperti pemadam kebakaran), mempercepat waktu respons eksternal tanpa bergantung pada intervensi manual. |
| 1.1.5 | Pendeteksi jumlah orang pada setiap ruangan | Perangkat Keras (Firmware) dan Perangkat Lunak (Platform) | Untuk secara otomatis mendeteksi jumlah orang pada ruangan melalui kamera yang terintegrasi lalu menyimpan pada SQL. Sistem pendeteksi menyala ketika alur AI agent mulai berjalan (tidak selalu mendeteksi pada setiap waktu) |
| 1.1.6 | Penyimpanan data jumlah orang yang terdeteksi | Perangkat Lunak(Platform) | Mengirimkan informasi jumlah orang yang masih terperangkap pada ruangan tertentu kepada pihak berwenang (seperti pemadam kebakaran) agar mempercepat preparasi dan penanganan. Sistem harus menyimpan data jumlah orang ke database SQL |

| | | | |
|---|---|---|---|
| | | | secara periodik (setiap 5 detik) selama mode darurat aktif |

1.2 Penanganan dan Pemantauan Data

| No | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 1.2.1 | Visualisasi Data Real Time | Perangkat Lunak (Platform) | Untuk menyediakan dashboard pemantauan bagi tim keamanan guna mendapatkan kesadaran situasional (situational awareness) yang lengkap mengenai kondisi seluruh gedung secara langsung. Jika terjadi kebakaran atau asap yang tinggi, maka dashboard akan menampilkan jumlah orang yang terdeteksi pada setiap ruangan |
| 1.2.2 | Penyimpanan Data Sensor | Perangkat Lunak (Platform) | Untuk mencatat semua data yang diterima dari sensor, menciptakan jejak audit digital yang dapat digunakan untuk analisis pasca-insiden atau pemeliharaan sistem. |
| 1.2.3 | Visualisasi data historis | Perangkat Lunak (Platform) | Untuk memungkinkan analisis tren data dari waktu ke waktu, membantu dalam mengidentifikasi |

| | | | |
|---|---|---|---|
| | | | anomali jangka panjang atau pola yang mungkin mengindikasikan potensi masalah sebelum menjadi keadaan darurat. Export to Sheets |

1.3 Deteksi Anomali dan Pemicu Peringatan

| No | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 1.3.1 | Deteksi Ambang Batas Asap | Perangkat Lunak (Platform) | Untuk secara otomatis mengidentifikasi tingkat asap yang berbahaya berdasarkan aturan yang telah ditentukan, menghilangkan ambiguitas dan kebutuhan akan interpretasi manusia pada tahap awal peringatan. |
| 1.3.2 | Deteksi Ambang Batas Suhu | Perangkat Lunak (Platform) | Untuk secara otomatis mendeteksi suhu yang tidak normal dan berpotensi membahayakan, yang berfungsi sebagai lapisan verifikasi kedua untuk kondisi kebakaran. |
| 1.3.3 | Deteksi Kondisi Kebakaran | Perangkat Lunak (AI Agent) | Untuk menetapkan "kondisi kebakaran" hanya ketika beberapa indikator (suhu dan asap) terlampaui secara bersamaan, bertujuan untuk mengurangi potensi alarm palsu (*false alarms*) yang dapat menurunkan kepercayaan pengguna |

| | | | |
|---|---|---|---|
| | | | terhadap sistem. |
| 1.3.4 | Deteksi Perangkat Hilang Koneksi | Perangkat Lunak (Platform) | Untuk memantau kesehatan sistem secara proaktif. Kebutuhan ini mengatasi masalah "kegagalan senyap" (silent failure) di mana sebuah sensor rusak atau offline tanpa diketahui, sehingga memastikan integritas sistem setiap saat. |

## Non Functional Requirement

2.1 Keamanan

| ID | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 2.1.1 | Enkripsi Data | Perangkat Lunak (Platform) & Perangkat Keras (Firmware) | Semua komunikasi antara perangkat ESP dan server pusat, serta antara server pusat dan n8n, dan n8n dengan API eksternal (Gemini, Twilio) harus dienkripsi menggunakan HTTPS/SSL. |
| 2.1.2 | Otentikasi Pengguna / Akses Kontrol | Perangkat Lunak (Platform) | Akses ke dasbor server pusat dan antarmuka n8n harus dibatasi untuk personil yang berwenang melalui autentikasi yang aman (misalnya, nama pengguna/kata sandi, kunci API). |

| | | | |
|---|---|---|---|
| 2.1.3 | Manajemen API Key | Perangkat Lunak (Platform) | Kunci API untuk layanan eksternal (Gemini, Twilio) harus disimpan dan dikelola dengan aman |

2.2 Usability

| ID | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 2.2.1 | Notifikasi jelas | Perangkat Lunak (Platform) | Pesan notifikasi harus jelas, ringkas, dan dapat segera ditindaklanjuti oleh penerima. |
| 2.2.2 | Konfigurasi yang mudah | Perangkat Lunak (Platform) | Parameter konfigurasi (ambang batas, detail kontak, tata letak kantor) harus mudah dimodifikasi oleh administrator yang berwenang |
| 2.2.3 | Dashboard yang mudah dipakai | Perangkat Lunak (Platform) | Dasbor pemantauan langsung harus intuitif dan mudah dipahami oleh personel pemantau. |

2.3 Maintainability

| ID | Kebutuhan | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 2.3.1 | Modularitas | Perangkat Lunak (Platform) & Perangkat Keras (Firmware) | Komponen sistem (firmware ESP, backend server, alur kerja n8n) harus bersifat modular dan memungkinkan pengembangan, |

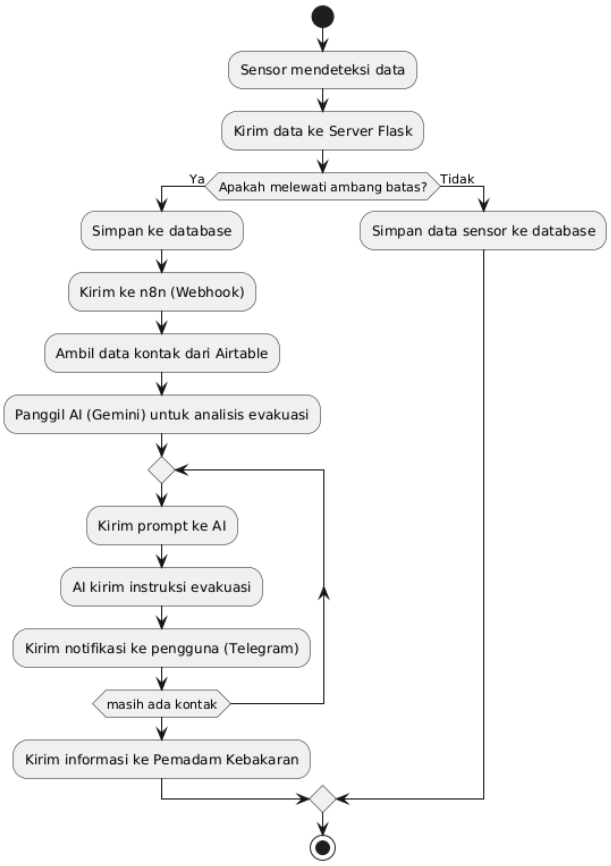
| | | | |
|---|---|---|---|
| | | | pengujian, serta penerapan yang independen |
| 2.3.2 | Skalabilitass sensor | Perangkat Lunak (Platform) | Sistem harus mampu menskalakan untuk mendukung setidaknya 50 perangkat ESP tanpa penurunan performa yang signifikan. |
| 2.3.3 | Skalabilitas notifikasi | Perangkat Lunak (Platform) | Sistem notifikasi harus mampu menangani sejumlah besar penerima secara bersamaan. |
| 2.3.5 | Keterbacaan Kode | Perangkat Lunak (Platform) | Semua kode sumber harus terdokumentasi dengan baik dan mengikuti standar pengkodean yang ditetapkan. |

2.4 Performa

| ID | Kebutuhan Implementasi | Domain Implementasi | Penjelasan |
|---|---|---|---|
| 2.4.1 | Tingkat Pembaruan Dasbor | Perangkat Lunak (Platform) | Dasbor pemantauan langsung harus memperbarui data sensor setidaknya setiap 2 detik. |
| 2.4.2 | Latensi Pembuatan Panggilan Pemadam Kebakaran | Perangkat Lunak (Platform) | Sistem harus memulai panggilan ke dinas pemadam kebakaran dalam waktu 20 detik setelah menerima peringatan kebakaran |
| 2.4.3 | Latensi Pemicu Peringatan | Perangkat Lunak (Platform) & Perangkat | Sistem harus memicu peringatan (mengirim sinyal HTTP ke n8n) |

| | | | |
|---|---|---|---|
| | | Keras (Firmware) | dalam waktu 2 detik setelah mendeteksi kondisi kebakaran atau perangkat yang hilang. |

## Desain Fungsional Sistem




Gambar 3.1 Activity Diagram

Activity diagram tersebut menggambarkan alur kerja sistem peringatan kebakaran berbasis IoT secara menyeluruh, mulai dari proses deteksi oleh sensor hingga pengiriman notifikasi ke pengguna dan pemanggilan pemadam kebakaran. Proses dimulai ketika sensor (seperti sensor suhu, asap, atau tombol panik) mendeteksi data di lingkungan gedung. Data ini kemudian dikirim ke backend server berbasis Flask yang bertugas memproses dan

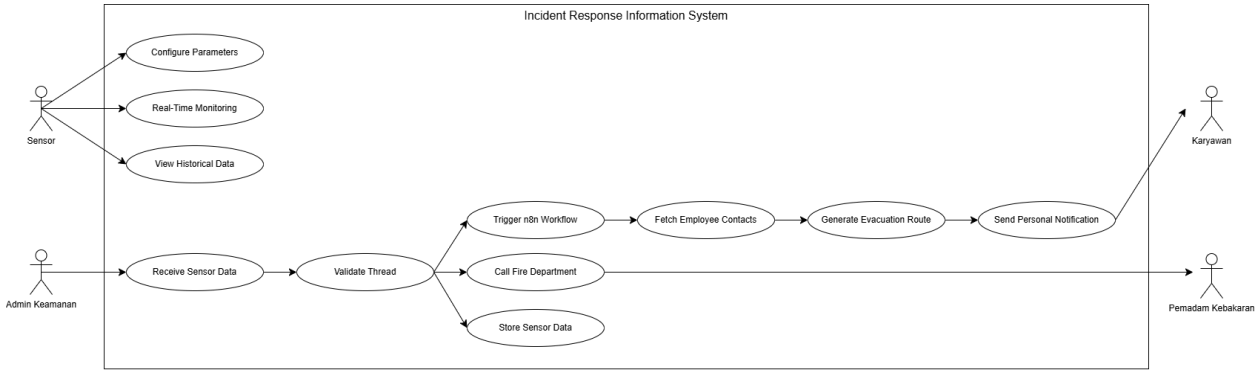
mengevaluasi apakah data tersebut melampaui ambang batas kritis yang menunjukkan potensi kebakaran.

Setelah data diterima, sistem melakukan pengambilan keputusan. Jika nilai data sensor tidak melewati ambang batas (misalnya, suhu masih normal, tidak ada asap), maka data tersebut hanya disimpan ke database sebagai catatan historis tanpa memicu alur darurat lebih lanjut. Namun, jika nilai sensor melebihi ambang batas—menandakan adanya kemungkinan kebakaran—maka data tersebut tidak hanya disimpan, tetapi juga diteruskan ke sistem otomasi (n8n) melalui webhook. n8n kemudian menjalankan serangkaian proses untuk menangani kondisi darurat. Langkah pertama adalah mengambil data kontak karyawan dari Airtable, termasuk lokasi ruangan mereka. Setelah itu, sistem memanggil layanan AI (Gemini) untuk menganalisis situasi kebakaran dan menentukan rute evakuasi terbaik bagi masing-masing karyawan berdasarkan lokasi api dan lokasi pengguna. Untuk setiap karyawan, n8n membentuk prompt yang dikirim ke AI, lalu menerima respon berupa instruksi evakuasi personal dalam bahasa alami.

Instruksi evakuasi yang dihasilkan oleh AI kemudian dikirimkan ke masing-masing karyawan melalui Telegram. Proses ini dilakukan satu per satu hingga seluruh kontak telah menerima instruksi. Setelah semua notifikasi terkirim, sistem secara otomatis menghubungi pihak pemadam kebakaran dengan memberikan informasi lokasi kebakaran, memungkinkan respons yang cepat tanpa harus menunggu intervensi manual.

Secara keseluruhan, diagram ini mencerminkan sistem IoT yang cerdas dan reaktif: tidak hanya mendeteksi bahaya secara real-time, tetapi juga memanfaatkan automasi dan AI untuk mengoordinasikan evakuasi serta pelaporan secara efisien dan personal. Alur ini juga menunjukkan bagaimana sistem menangani kondisi normal dan darurat secara terpisah namun terintegrasi, serta meminimalkan false alarm dengan kombinasi sensor dan validasi multi-indikator.

## Skenario Penggunaan Perangkat/Pemanfaatan Produk



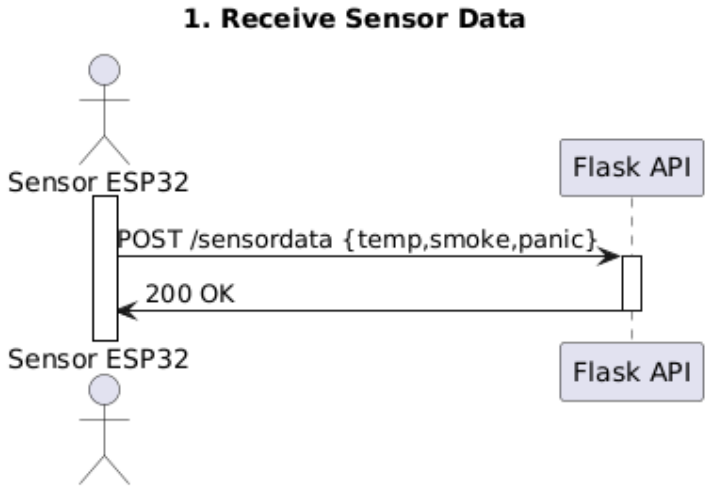

Gambar 4.1 Use Case Diagram

Use case diagram tersebut menggambarkan sistem Incident Response Information System yang dirancang untuk menangani kondisi darurat kebakaran di lingkungan perkantoran secara otomatis dan cerdas. Sistem ini melibatkan empat aktor utama: Sensor, Admin Keamanan, Karyawan, dan Pemadam Kebakaran, masing-masing berperan dalam berbagai proses yang mendukung deteksi, pengambilan keputusan, dan respons terhadap potensi kebakaran. Sensor merupakan perangkat IoT seperti ESP32 yang dilengkapi dengan modul pendeteksi suhu, asap, dan tombol panik. Sensor memiliki peran utama dalam menginisiasi alur sistem dengan mengirimkan data ke server dalam use case Receive Sensor Data. Setelah itu, sistem akan melakukan Validate Threshold, yaitu mengevaluasi apakah data tersebut melebihi ambang batas tertentu yang menandakan potensi bahaya. Berdasarkan hasil validasi, sistem dapat menyimpan data sensor secara normal (Store Sensor Data), atau jika kondisi darurat terdeteksi, sistem akan memicu workflow otomatis melalui Trigger n8n Workflow.

Workflow ini melibatkan pengambilan data karyawan (Fetch Employee Contacts), pemrosesan data oleh AI untuk menentukan jalur evakuasi optimal (Generate Evacuation Route), dan pengiriman instruksi evakuasi personal kepada masing-masing karyawan melalui Telegram (Send Personal Notification). Dalam proses ini, AI berperan sebagai penasihat evakuasi cerdas yang memperhitungkan lokasi api dan lokasi pengguna untuk menentukan rute teraman. Seluruh alur ini memastikan bahwa setiap karyawan mendapatkan panduan evakuasi yang relevan dan langsung dapat ditindaklanjuti. Selain notifikasi ke pengguna internal, sistem juga secara otomatis menghubungi pihak eksternal, yaitu Pemadam Kebakaran,

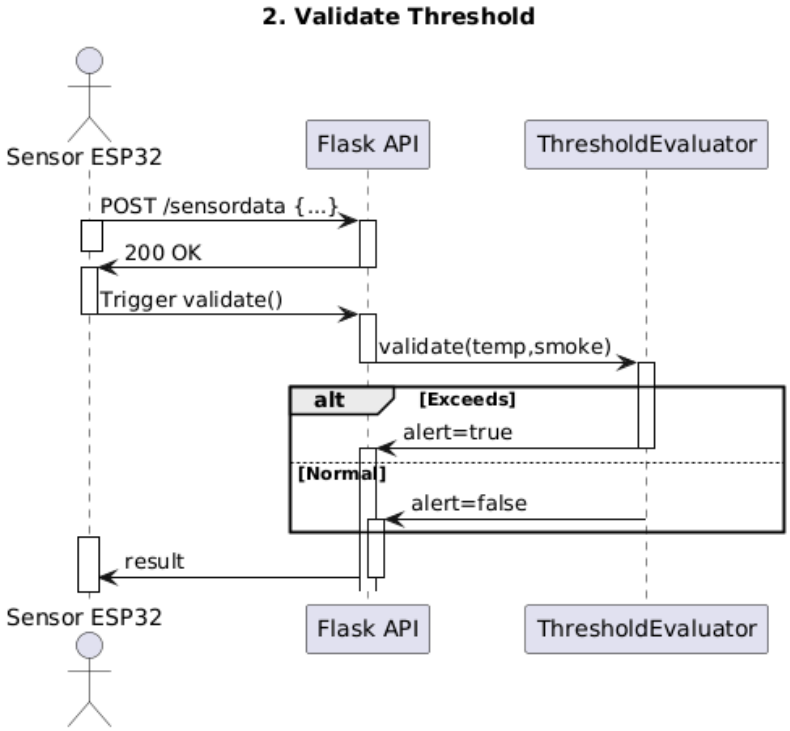
melalui use case Call Fire Department. Ini memungkinkan respons cepat tanpa keterlambatan akibat proses pelaporan manual. Di sisi lain, Admin Keamanan memiliki akses terhadap fitur-fitur sistem lainnya seperti Configure Parameters untuk mengubah pengaturan sistem, Real-Time Monitoring untuk memantau kondisi secara langsung dari dashboard, dan View Historical Data untuk menganalisis data-data sebelumnya yang telah terekam.

Secara keseluruhan, use case diagram ini menggambarkan arsitektur logis dari sistem respons kebakaran berbasis IoT yang tidak hanya reaktif terhadap insiden, tetapi juga proaktif dalam mengoordinasikan evakuasi dan pelaporan secara otomatis. Dengan integrasi sensor, AI, workflow automation, dan komunikasi instan ke pihak terkait, sistem ini berfungsi sebagai solusi menyeluruh yang mengoptimalkan keselamatan penghuni gedung di saat darurat.



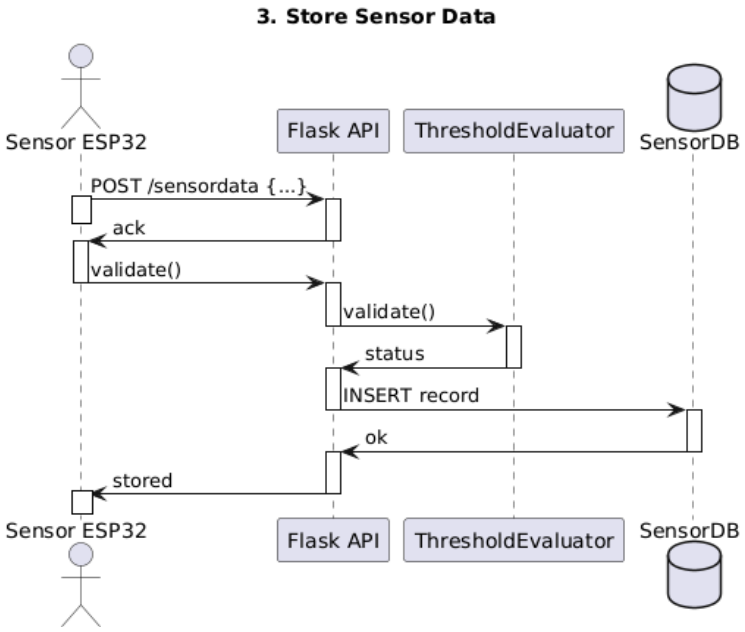

Gambar 4.2 Sequence Diagram Receive Sensor Data

Diagram ini menunjukkan awal proses ketika sensor ESP32 mendeteksi data suhu, asap, atau tombol panik. Sensor mengirimkan data melalui HTTP POST ke server Flask. Flask API menerima data dan memberi respons 200 OK. Ini adalah proses pertama yang memicu seluruh sistem. Belum ada evaluasi atau tindakan lebih lanjut dilakukan. Fungsinya hanya sebagai penerima awal. Aktivasi dan deaktivasi mengikuti urutan komunikasi dua arah.



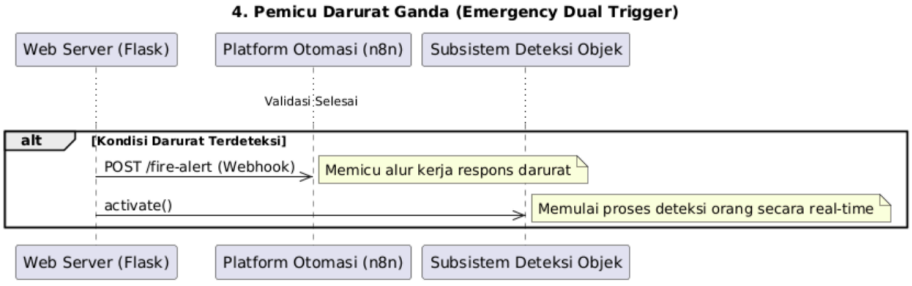

Gambar 4.3 Sequence Diagram Validate Threshold

Diagram ini memperlihatkan proses validasi data sensor. Setelah data diterima oleh Flask API, sistem memanggil modul evaluasi ambang batas. ThresholdEvaluator akan mengecek apakah suhu/asap melebihi ambang. Jika iya, nilai alert = true dikirim balik. Jika tidak, status tetap normal. Proses ini penting untuk menentukan apakah sistem harus beralih ke mode darurat. Aktor yang memicu tetap Sensor ESP32.




Gambar 4.4 Sequence Diagram Store Sensor Data

Diagram ini melanjutkan dari validasi data. Setelah validasi dilakukan, data sensor disimpan ke dalam database. Proses dimulai dari pengiriman data oleh Sensor ke Flask. Kemudian Flask memanggil evaluator untuk mengecek kondisi. Setelah itu, data disimpan ke SensorDB. Akhirnya, konfirmasi disampaikan kembali ke sensor bahwa data telah disimpan. Ini memastikan setiap input tercatat, baik darurat maupun tidak.



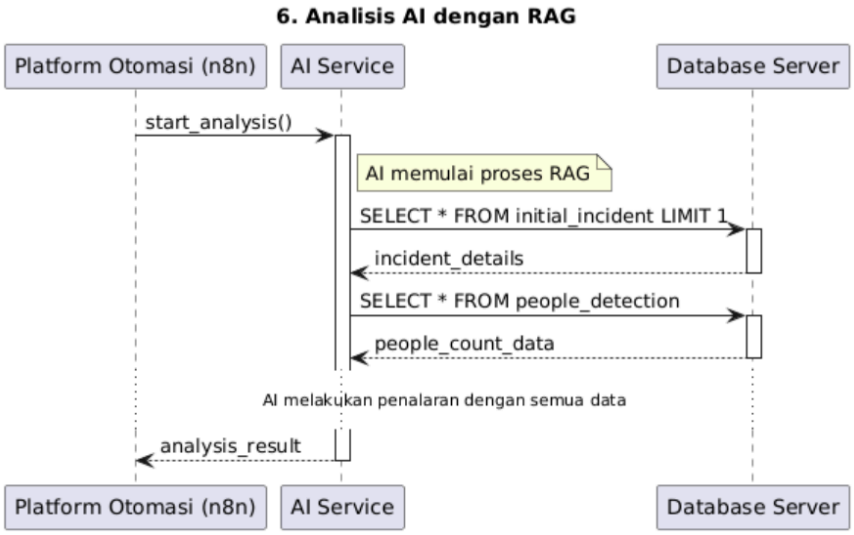

Gambar 4.5 Sequence Diagram Trigger n8n Workflow

Diagram ini menjelaskan sekuens kejadian segera setelah Web Server menerima data sensor dan memvalidasinya sebagai kondisi darurat. Alih-alih satu aksi, Web Server secara

paralel menginisiasi dua alur proses yang independen: mengaktifkan alur kerja respons di Platform Otomasi dan memulai Subsistem Deteksi Objek.


Gambar 4.6 Sequence Diagram Fetch Employee Contacts

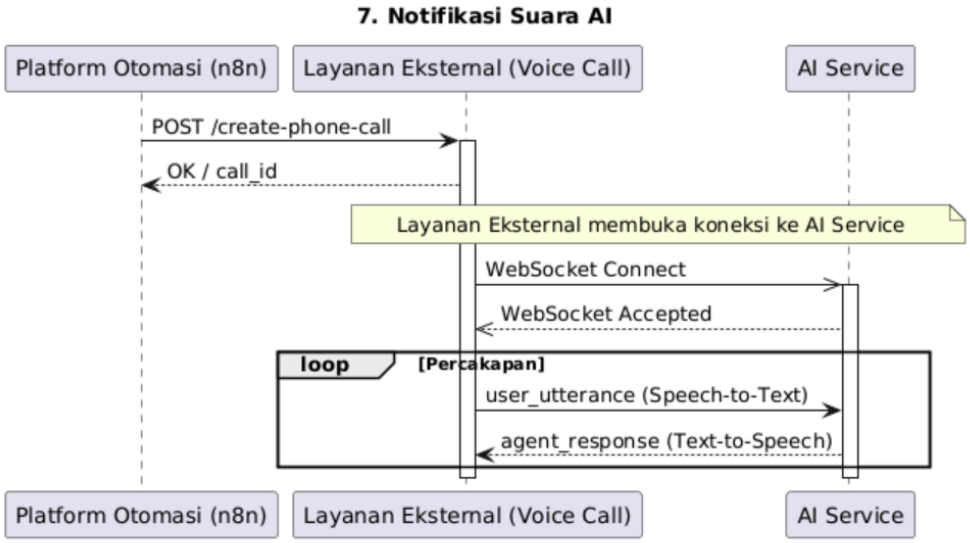
Setelah diaktifkan, Subsistem Deteksi Objek berjalan sebagai layanan latar belakang yang terspesialisasi. Diagram ini menunjukkan siklus hidupnya, di mana ia secara terus-menerus melakukan inferensi pada stream video dan secara periodik memperbarui Database Server dengan data jumlah orang terkini.




Gambar 4.7 Sequence Diagram Generate Evacuation Route

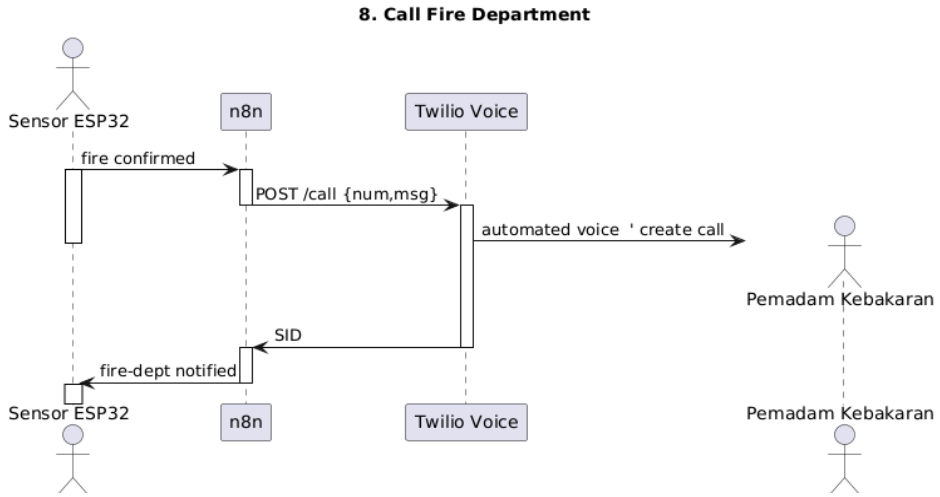
Diagram ini menunjukkan kecerdasan sistem. Setelah dipicu oleh Platform Otomasi, AI Service tidak hanya menunggu input. Ia secara proaktif menjalankan proses

Retrieval-Augmented Generation (RAG) dengan melakukan beberapa query ke Database Server untuk mengumpulkan semua konteks yang diperlukan (detail insiden awal dan data hunian real-time) sebelum menghasilkan analisisnya.



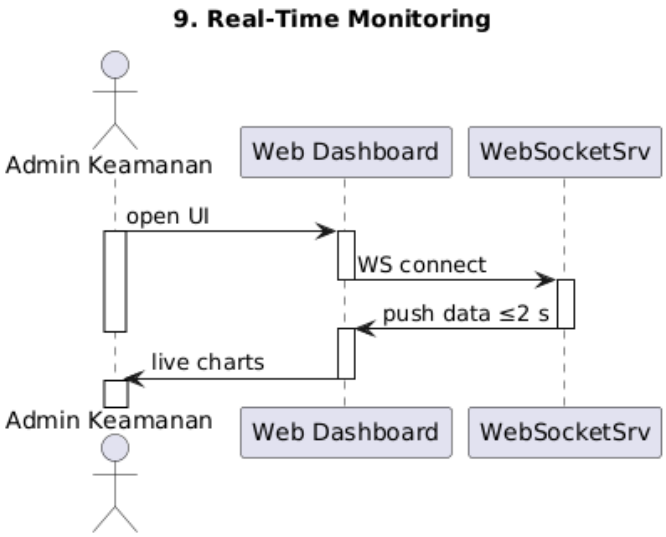

Gambar 4.8 Sequence Diagram Send Personal Notification

Diagram final ini menggambarkan bagaimana notifika si suara yang cerdas dieksekusi. Platform Otomasi mendelegasikan tugas panggilan ke Layanan Eksternal (misalnya, Retell AI). Layanan ini kemudian membuka koneksi websocket yang persisten ke AI Service, yang selanjutnya mengambil alih dan mengelola seluruh percakapan secara real-time.



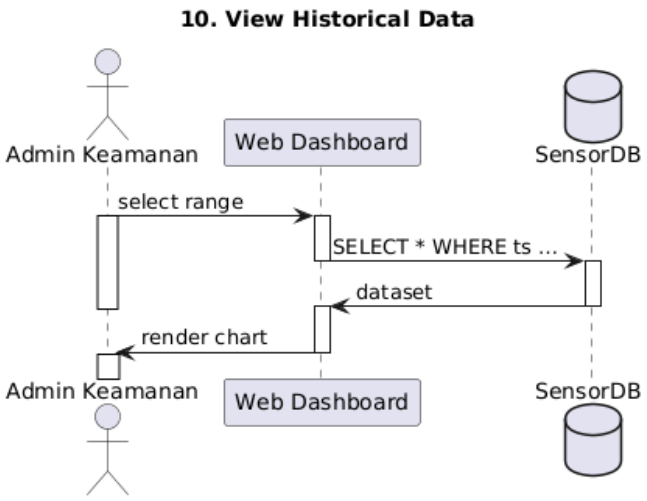
Gambar 4.9 Sequence Diagram Call Fire Department

Setelah notifikasi selesai, diagram ini menunjukkan pemanggilan pemadam kebakaran. n8n menghubungi Twilio untuk melakukan panggilan otomatis. Twilio akan menelepon nomor darurat dengan pesan suara yang sudah ditentukan. Pemadam Kebakaran adalah aktor penerima di tahap ini. Proses ini dilakukan tanpa intervensi manusia. Setelah panggilan dilakukan, SID dikirim kembali sebagai bukti. Konfirmasi juga dikirim ke sensor.



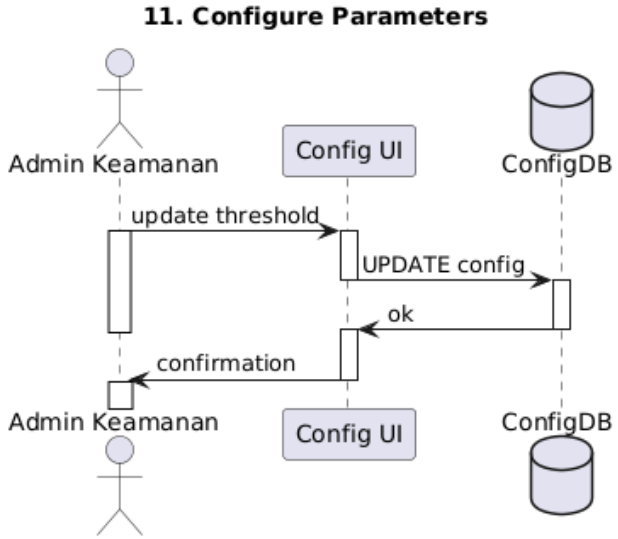

Gambar 4.10 Sequence Diagram Real-Time Monitoring

Diagram ini menunjukkan proses monitoring oleh admin. Admin membuka dashboard web untuk melihat kondisi secara langsung. Dashboard terkoneksi dengan WebSocket Server. Server mengirim data sensor secara real-time setiap dua detik. Admin melihat hasil dalam bentuk grafik dan notifikasi. Tidak ada penyimpanan atau logika kompleks di sini. Proses ini bersifat visualisasi dan pemantauan saja.




Gambar 4.11 Sequence Diagram View Historical Data

Diagram ini menjelaskan proses akses data historis oleh admin. Admin memilih rentang waktu dari dashboard. Dashboard mengirim query ke database untuk mengambil data sesuai tanggal. Database mengembalikan dataset. Dashboard kemudian menampilkan grafik atau tabel ke admin. Proses ini penting untuk audit dan analisis tren. Tidak ada interaksi sensor dalam use case ini.




Gambar 4.12 Sequence Diagram Configure Parameters

Diagram ini menggambarkan pengaturan sistem oleh admin. Admin membuka antarmuka konfigurasi dan mengubah parameter seperti ambang batas. UI mengirimkan update ke database konfigurasi. Database menyimpan perubahan. Konfirmasi dikembalikan ke admin. Ini memungkinkan sistem tetap fleksibel dan dapat disesuaikan. Tidak ada elemen darurat dalam alur ini—hanya administratif.

## Spesifikasi Perangkat Lunak dan Anggaran Biaya

Terdapat beberapa komponen perangkat keras yang dibutuhkan dalam merealisasikan project ini. Berikut adalah list komponen IoT yang dipakai :

| Komponen | Fungsi Utama dalam Proyek | Alasan Pemilihan |
|---|---|---|
| ESP32 Dev | Otak/Mikrokontroler Utama. Membaca data dari semua sensor, memproses logika awal, dan mengirimkan data ke server melalui Wi-Fi. | Dipilih karena memiliki prosesor dual-core yang kuat, modul Wi-Fi & Bluetooth bawaan, serta jumlah pin GPIO yang memadai untuk semua sensor dan aktuator yang digunakan. |
| Sensor Asap MQ-2 | Detektor Asap dan Gas. Mendeteksi keberadaan asap dan gas mudah terbakar, yang merupakan indikator utama kebakaran. | Merupakan sensor yang sangat umum, terjangkau, dan sensitif terhadap berbagai jenis gas, memberikan deteksi dini yang andal. |
| Sensor Suhu DHT11 | Detektor Suhu & Kelembapan. Mengukur suhu ambien untuk mendeteksi kenaikan panas abnormal yang mengindikasikan adanya api. | Sensor digital yang hemat biaya dan mudah diintegrasikan untuk memberikan data suhu sebagai lapisan verifikasi kedua. *(Catatan: Untuk akurasi lebih tinggi di versi* |

| | | |
|---|---|---|
| | | *produksi, dapat di-upgrade ke DHT22)*. |
| Push Button | Pemicu Alarm Manual. Memungkinkan pengguna di lokasi untuk secara manual memicu alarm darurat jika mereka melihat bahaya sebelum terdeteksi sensor. | Komponen dasar berbiaya sangat rendah yang memenuhi kebutuhan fungsional "Panic Button" secara efektif. |
| Grove - 16x2 LCD | Display Antarmuka Perangkat Keras. Menampilkan status lokal, pembacaan suhu, atau pesan peringatan langsung pada perangkat. | Menyediakan antarmuka pengguna visual di tingkat perangkat keras. Konektor Grove menyederhanakan proses wiring dan mengurangi potensi kesalahan koneksi pada setiap ruangan. |
| Breadboard & Kabel Jumper | Platform Prototyping. Menghubungkan semua komponen secara non-permanen untuk kemudahan perakitan, pengujian, dan iterasi desain. | Alat standar untuk prototyping cepat tanpa memerlukan penyolderan. |
| Kabel USB Micro | Catu Daya & Pemrograman. Menyediakan daya ke ESP32 dan sebagai jalur untuk mengunggah firmware dari komputer. | Merupakan standar konektor yang umum dan mudah didapatkan. |

Biaya Anggaran Project IRIS:

| **Komponen** | **Biaya** |
|---|---|

| | |
|---|---|
| ESP32 | Rp. 65.000 |
| MQ2 Smoke Sensor | Rp. 14.900 |
| DHT 11 Sensor | Rp. 10.900 |
| Button | Rp. 300 |
| Grove-16x2 LCD | Rp. 17.000 |
| Breadboard | Rp. 9.500 |
| USB Cable | Rp. 8.800 |
| **Total** | Rp. 126.400 |

# Desain Perangkat Lunak

## Desain Diagram (Arsitektur) Blok Perangkat Lunak Sistem

Block Diagram adalah sebuah ilustrasi dari sebuah sistem dimana komponen utama direpresentasikan oleh sebuah block dan setiap kotak block memiliki hubungan dengan blok tertentu. Pada project ini, terdapat delapan blok alias delapan komponen utama yang merepresentasikan sistem secara keseluruhan. Block yang dimaksud pada sistem project ini mencakupi :

1. Web Server

   Web Server adalah sebuah komponen penting dimana web server bertindak sebagai jantung dari sistem. Ini dikarenakan web server merupakan titik masuk utama untuk semua data. Web server berfungsi sebagai backend yang menangani logika sistem dan komunikasi antar komponen. Tujuan dari hadirnya web server adalah untuk menerima

data sensor, menampilkan dashboard monitoring dan juga bertindak sebagai pengontrol yang mengaktifkan proses darurat lainnya saat batas bahaya terlampaui. Web server akan menerima data dari sensor melalui API. Disaat batas bahaya terlampaui, maka web server akan mengirimkan signal pemicu pada kamera pendeteksi orang dan n8n workflow agent. Web server akan membaca dan menulis data di waktu yang ditentukan ke database server dan menampilkan data pada dashboard.

2. Database Server

   Database server adalah komponen yang menyimpan data yang diterima dari sensor. Komponen ini menyimpan beberapa jenis informasi seperti data sensor historis, data jumlah penghuni yang terdeteksi oleh kamera ketika batas bahaya terlampaui. Database server mencatat semua kejadian dan menyediakan dari sensor dengan menampilkan pada dashboard monitoring dalam bentuk grafik. Data data yang diambil oleh sensor langsung disimpan dan menjadi sumber informasi untuk AI service secara real time agar AI service mampu membuat keputusan evakuasi yang akurat.

3. AI Service

   AI service pada sistem ini merupakan komponen AI yang berfungsi untuk menganalisis situasi darurat secara mendalam dan detail menggunakan metode Retrieval-Augmented Generation (RAG). Lalu, komponen ini juga menelpon pemadam kebakaran dan memberikan informasi kritis secara akurat dan secara real time.  AI service diaktifkan oleh N8N workflow agent dan setelah diaktifkan, AI service akan melakukan query terhadap Database Server untuk mengambil detail insiden dan data jumlah orang pada ruangan. Lalu, data yang diambil akan digabungkan dengan knowledge base internal untuk menghasilkan sebuah informasi yang benar pada saat sesi menelepon pemadam kebakaran

4. Pendeteksi Orang

   Pendeteksi orang adalah sebuah kamera yang mampu mendeteksi jumlah orang pada ruangan melalui metode *computer vision.* Kamera ini hadir untuk mendapatkan data yang tidak bisa diambil oleh sensor lingkungan seperti jumlah orang pada ruangan.

Dengan hadirnya informasi ini, sistem dan pemadam kebakaran memiliki strategi yang lebih mudah dalam memprioritaskan evakuasi dan penyelamatan. Komponen ini dipicu oleh web server saat kondisi darurat terdeteksi. Setelah komponen ini diaktifkan, kamera akan mengambil data dan mengirimkan hasil perhitungan jumlah orang pada database server

5. N8N workflow agent

   Komponen ini berfungsi sebagai orkestrator atau "manajer tugas" untuk proses respons darurat. Tujuannya adalah untuk menjalankan serangkaian langkah yang telah ditentukan secara otomatis ketika alarm terpicu, memisahkan logika respons darurat dari logika bisnis utama di Web Server. Ini membuat sistem lebih modular. Ia menerima sinyal pemicu (*webhook*) dari Web Server, kemudian ia bertugas untuk mengaktifkan layanan lain seperti AI Service (untuk memulai panggilan suara) dan Layanan Notifikasi Eksternal (untuk mengirim pesan teks).

6. Sensor dan Kamera

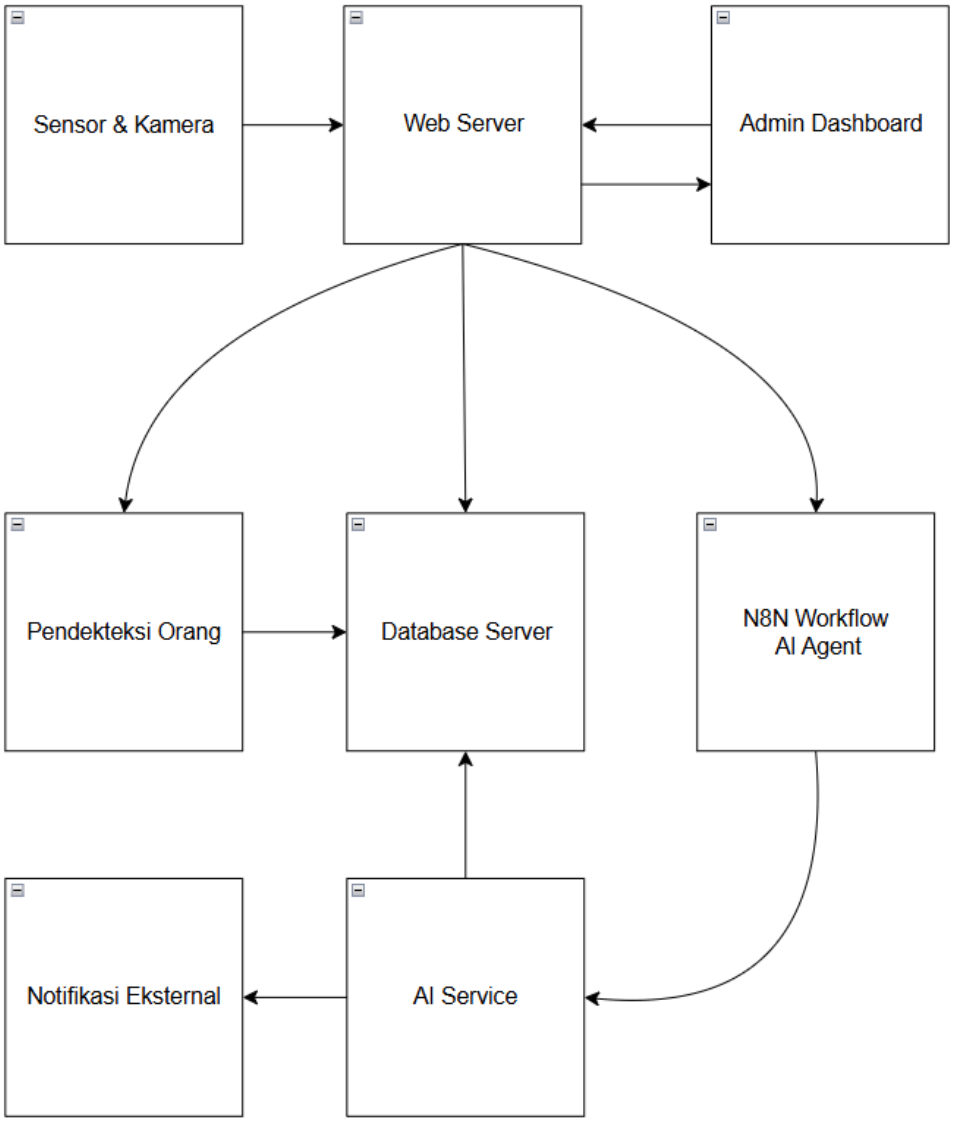
   Ini adalah unit perangkat fisik (ESP32 beserta sensornya) yang ditempatkan di seluruh gedung. Tujuannya adalah menjadi "mata dan telinga" sistem di lapangan, yang secara terus-menerus memantau kondisi lingkungan (suhu, asap) dan menerima input manual (tombol panik). Komponen inilah yang menjadi sumber data utama dan menginisiasi seluruh alur kerja dengan mengirimkan data hasil pembacaan sensor ke Web Server melalui panggilan API.

7. Admin Dashboard

   Ini adalah antarmuka pengguna berbasis web yang diakses oleh personel keamanan atau admin melalui *web browser*. Tujuannya adalah untuk menyediakan visualisasi data secara terpusat, memberikan kesadaran situasional (*situational awareness*) secara *real-time* mengenai kondisi setiap ruangan, dan memungkinkan analisis data historis. Ia berkomunikasi secara dua arah dengan Web Server: meminta halaman web dan data, lalu menerima pembaruan data secara berkala untuk ditampilkan.

8. Notifikasi Eksternal

Blok ini merepresentasikan layanan pihak ketiga yang digunakan sistem untuk mengirimkan peringatan ke dunia nyata. Tujuannya adalah untuk menjadi jembatan komunikasi antara sistem digital dengan manusia. Komponen ini tidak memulai komunikasi, melainkan menerima perintah dari Platform Otomasi atau AI Service untuk menjalankan tugas spesifik, seperti mengirim pesan teks melalui API Telegram atau melakukan panggilan suara melalui platform komunikasi seperti Retell AI.



Arsitektur perangkat lunak pada sistem ini bersifat microservice. Ini terlihat pada komponen komponen utama seperti web server, pendeteksi orang dan juga ai service yang dirancang sebagai proses atau layanan yang berjalan secara independen. Faktor faktor yang mengkategorikan sistem ini merupakan microservice dikarenakan adanya alasan pemisahan tanggung jawab, skalabilitas, dan independensi teknologi.

Pertama, pada sistem ini terdapat pemisahan pertanggung jawaban (web serverm pendeteksi objek, ai service, dll) yang merupakan prinsip utama dari adopsi *microservices* yaitu untuk mendelegasikan setiap fungsi bisnis yang berbeda ke layanan yang terisolasi. Hal ini secara signifikan meningkatkan modularitas sistem, sehingga lebih mudah untuk dipahami, dikembangkan, dan diuji secara terpisah.

Kedua, setiap layanan memiliki profil kebutuhan sumber daya komputasi yang unik. Arsitektur *microservices* memungkinkan alokasi sumber daya yang optimal dan efisien sesuai dengan tuntutan beban kerja masing-masing layanan.

Ketiga, pemisahan layanan ke dalam unit-unit independen memungkinkan setiap layanan untuk memiliki tumpukan teknologi (*tech stack*) dan set dependensinya sendiri tanpa menimbulkan konflik. Contohnya, AI Service memerlukan library spesifik seperti fastapi, retell-sdk, dan google-generativeai sedangkan web server dibangun menggunakan Flask.

Menggabungkan semua dependensi ini dalam satu lingkungan pada aplikasi monolitik akan menciptakan risiko konflik dependensi dan ketidakcocokan versi antar-pustaka. Dengan *microservices*, setiap layanan memiliki lingkungan dependensinya sendiri yang terisolasi. Hal ini secara drastis menyederhanakan siklus hidup pengembangan perangkat lunak (*software development lifecycle*), mulai dari pengembangan, pengujian, hingga pembaruan, karena perubahan pada dependensi satu layanan tidak akan mempengaruhi layanan lainnya.

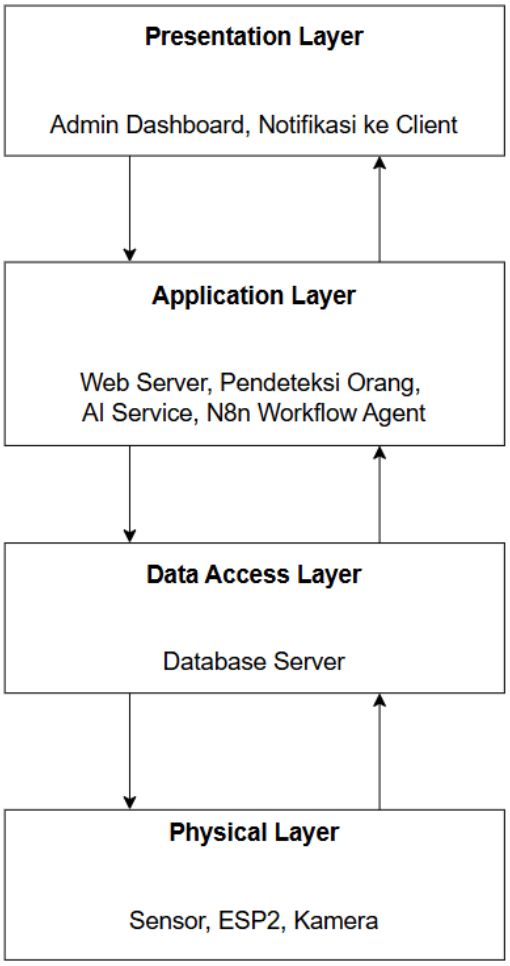
Untuk tahap implementasi dan pengembangan saat ini, setiap layanan dalam arsitektur ini dijalankan sebagai satu instance. Artinya, Web Server, Pendeteksi Orang, dan AI Service masing-masing beroperasi sebagai proses tunggal pada server. Pendekatan ini dipilih karena kepraktisan dan efisiensi untuk lingkup pengembangan prototipe.

Meskipun demikian, salah satu keunggulan fundamental dari pemilihan arsitektur microservices adalah kemampuannya untuk mendukung orkestrasi di masa depan. Jika sistem ini akan diimplementasikan dalam skala produksi yang lebih besar, setiap layanan dapat dengan mudah dikemas (misalnya, menggunakan kontainer Docker) dan dijalankan dalam berbagai instance untuk mencapai ketersediaan tinggi (high availability) dan distribusi beban (load balancing). Sebagai contoh, jika lalu lintas ke dashboard meningkat, hanya Web Server yang perlu diskalakan dengan menambahkan instance-nya, tanpa memengaruhi layanan lain. Arsitektur ini secara inheren telah dirancang untuk siap tumbuh dan beradaptasi dengan kebutuhan skalabilitas di masa depan.

## ☆ Desain Diagram Arsitektur Lapis Perangkat Lunak Sistem

Layered architecture dalam perangkat lunak adalah salah satu pola arsitektur perangkat lunak yang memiliki pendekatan yang paling fundamental dan teruji. Pendekatan ini mengorganisir sistem dengan melakukan partisi secara hierarkis ke dalam beberapa kelompok subsistem horizontal yang disebut lapisan (*layers*). Setiap lapisan memiliki peran dan tanggung jawab yang spesifik serta menyediakan serangkaian layanan kohesif untuk lapisan yang berada tepat di atasnya. Prinsip utama dari arsitektur ini adalah untuk mencapai pemisahan tanggung jawab (*separation of concerns*) secara tegas di seluruh sistem.

Struktur ini memberlakukan aturan interaksi yang ketat, di mana sebuah lapisan umumnya hanya diizinkan untuk berkomunikasi dengan lapisan yang berada tepat di bawahnya (sebagai penyedia layanan) atau di atasnya (sebagai klien). Komunikasi yang melompati lapisan secara langsung sangat tidak dianjurkan karena akan merusak prinsip isolasi dan meningkatkan ketergantungan antar-lapisan (*coupling*). Dengan membatasi dependensi ini, arsitektur lapis secara efektif mengabstraksikan detail implementasi dari satu lapisan terhadap lapisan lainnya, sehingga menciptakan sebuah sistem yang lebih terstruktur dan modular.




Physical layer ini merupakan fondasi dari arsitektur dan terdiri dari seluruh komponen perangkat keras (*hardware*) yang di-deploy di lapangan untuk berinteraksi langsung dengan lingkungan fisik. Tanggung jawab utamanya adalah melakukan akuisisi data mentah (*raw data acquisition*) dari dunia nyata. Dalam proyek ini, komponen pada lapisan ini mencakup unit-unit Node Sensor IoT—yang terdiri dari mikrokontroler ESP32, sensor suhu, dan sensor asap—serta unit Kamera yang berfungsi sebagai input untuk visi komputer. Alur komunikasi dari lapisan ini bersifat unidirectional ke atas; ia secara periodik mengirimkan data sensorik dan visual yang telah dikumpulkan ke Lapis Aplikasi untuk diproses lebih lanjut.

Data access layer berfungsi untuk menyediakan abstraksi terhadap semua operasi penyimpanan dan pengambilan data, sehingga memisahkan logika inti aplikasi dari detail teknis implementasi database. Tanggung jawab utamanya adalah mengelola persistensi

seluruh data sistem secara andal dan terstruktur. Ini mencakup penyimpanan log data sensor deret waktu, data hunian (*occupancy*) yang dihasilkan oleh subsistem deteksi objek, serta data detail insiden awal yang krusial. Lapisan ini mengekspos antarmuka standar untuk manipulasi data (*Create, Read, Update, Delete*) secara eksklusif kepada Lapis Aplikasi. Akses langsung dari lapisan lain, terutama Lapis Presentasi, dilarang keras untuk menjaga integritas data dan menegakkan prinsip pemisahan arsitektural.

Application layer sebagai inti dari arsitektur, lapisan ini menampung seluruh logika bisnis (business logic), aturan sistem, dan fungsionalitas pemrosesan utama. Sesuai dengan arsitektur microservices yang diadopsi, lapisan ini merupakan komposisi dari beberapa layanan independen: Web Server, Subsistem Deteksi Objek, Platform Otomasi Alur Kerja, dan AI Service. Tanggung jawabnya sangat luas, meliputi: validasi data dari Lapis Fisik, penegakan aturan bisnis (seperti pemeriksaan ambang batas bahaya), orkestrasi alur kerja respons darurat, eksekusi analisis berbasis AI yang kompleks (termasuk Retrieval-Augmented Generation), dan koordinasi transaksi data dengan Lapis Akses Data di bawahnya. Lapisan ini bertindak sebagai perantara utama yang menghubungkan data mentah dengan presentasi yang bermakna.

Presentation layer adalah lapisan teratas yang bertanggung jawab atas semua interaksi yang menghadap pengguna akhir, baik itu pengguna manusia maupun sistem eksternal. Fungsinya adalah untuk merepresentasikan data dan hasil olahan dari Lapis Aplikasi ke dalam format yang dapat dipahami, serta meneruskan input dari pengguna ke Lapis Aplikasi untuk dieksekusi. Komponen pada lapisan ini meliputi Admin Dashboard berbasis web yang menyediakan visualisasi data real-time dan historis untuk personel keamanan, serta Klien Notifikasi yang menjadi titik akhir pengiriman peringatan, seperti antarmuka pesan teks (Telegram) dan layanan panggilan suara (AI Voice Call). Lapisan ini bersifat agnostik terhadap implementasi logika bisnis dan detail penyimpanan data, yang memungkinkannya untuk dikembangkan atau dimodifikasi secara independen.

## Pemilihan Teknologi

Pemilihan *technology stack* merupakan keputusan arsitektural fundamental yang menentukan kapabilitas, keandalan, dan skalabilitas dari Incident Response Information System (IRIS). Setiap teknologi dipilih bukan hanya sebagai komponen individual, tetapi sebagai bagian dari ekosistem yang terintegrasi untuk mendukung tujuan utama sistem: menyediakan respons insiden kebakaran yang cepat, cerdas, dan dapat ditindaklanjuti. Proses seleksi ini secara cermat mempertimbangkan berbagai alternatif, dengan pilihan akhir didasarkan pada analisis mendalam terhadap kebutuhan fungsional, non-fungsional , serta strategi deployment on-premise yang menjadi landasan utama sistem ini.

1. Sistem Operasi

   Sistem operasi (OS) berfungsi sebagai platform dasar yang mengelola interaksi antara perangkat keras server dan seluruh perangkat lunak aplikasi. Pilihan OS berdampak langsung pada stabilitas, keamanan, dan kemudahan pengelolaan sistem secara keseluruhan.

   a. Alternatif Teknologi:
      i. Distribusi Linux (Ubuntu Server, Debian): Sistem operasi sumber terbuka yang menjadi standar industri untuk hosting aplikasi web. Dikenal dengan ekosistem yang matang, keamanan yang solid, dan dukungan komunitas yang masif.
      ii. Windows Server: Sistem operasi komersial dari Microsoft yang menawarkan antarmuka grafis yang familiar dan integrasi mendalam dengan produk Microsoft lainnya. Namun, ini datang dengan biaya lisensi.
   b. Pilihan dan Justifikasi:


Pilihan yang optimal untuk menjalankan server IRIS adalah Ubuntu Server. Keputusan ini didasarkan pada beberapa pilar justifikasi yang kuat:

i. Kompatibilitas Unggul: Seluruh komponen inti dari sistem ini dirancang dan dioptimalkan untuk berjalan di lingkungan Linux. Ini mencakup runtime Python untuk mengeksekusi framework Flask, database SQLite, Object Detection, serta seluruh dependencies yang diperlukan oleh perangkat lunak. Penggunaan Linux memastikan tidak ada masalah kompatibilitas yang dapat menghambat pengembangan atau operasi.
ii. Efisiensi Biaya dan Kepatuhan Proyek: Ubuntu Server sepenuhnya gratis dan bersifat sumber terbuka, sejalan dengan ketentuan proyek untuk tidak menggunakan teknologi yang bersifat eksklusif dan menghabiskan biaya.
iii. Kinerja dan Efisiensi Sumber Daya: Lingkungan Linux dikenal memiliki overhead yang minimal, memungkinkan alokasi sumber daya komputasi (CPU dan RAM) yang lebih besar untuk aplikasi itu sendiri. Ini sangat krusial untuk strategi deployment on-premise, di mana sistem mungkin dijalankan pada perangkat keras khusus seperti Mini PC yang memiliki sumber daya terbatas.
iv. Keamanan: Linux dibangun dengan model keamanan multi-pengguna dan sistem perizinan file yang granular. Arsitektur ini, secara default, memberikan permukaan serangan (attack surface) yang lebih kecil dibandingkan dengan sistem operasi lain dan didukung oleh komunitas global yang secara proaktif merilis pembaruan keamanan.

v. Dukungan dan Dokumentasi: Sebagai salah satu distribusi Linux paling populer, Ubuntu Server memiliki repositori perangkat lunak yang sangat luas dan didukung oleh jutaan tutorial, forum, dan dokumentasi resmi. Ini secara signifikan mempercepat proses penyelesaian masalah dan implementasi fitur baru.

2. Virtualisasi dan Kontainerisasi

Teknologi virtualisasi dan kontainerisasi menyediakan lapisan abstraksi antara aplikasi dan sistem operasi, yang bertujuan untuk standardisasi lingkungan, isolasi proses, dan portabilitas.

a. Alternatif Teknologi:
   i. Bare Metal (Tanpa Virtualisasi): Menjalankan aplikasi secara langsung di atas OS host. Pendekatan ini paling sederhana namun sangat rentan terhadap konflik dependensi antar aplikasi dan membuat proses replikasi lingkungan menjadi manual dan rawan kesalahan.
   ii. Mesin Virtual (Virtual Machines - VMs): Teknologi ini mensimulasikan seluruh mesin fisik, termasuk perangkat keras dan sistem operasinya. VM menawarkan isolasi terkuat tetapi dengan overhead kinerja yang sangat tinggi karena setiap VM menjalankan OS lengkap.
   iii. Kontainerisasi (Docker): Teknologi OS-level virtualization yang membungkus aplikasi beserta seluruh dependensinya (pustaka, biner, file konfigurasi) ke dalam unit standar yang disebut kontainer. Kontainer berbagi kernel OS host, membuatnya sangat ringan, cepat, dan efisien.
b. Pilihan dan Justifikasi:


Untuk deployment sistem IRIS, Docker dipilih sebagai teknologi kontainerisasi. Ini adalah praktik terbaik industri yang memberikan keuntungan signifikan:

i. Konsistensi dan Eliminasi Konflik Dependensi: Sebuah Dockerfile mendefinisikan secara eksplisit semua dependensi sistem, termasuk versi Python dan pustaka yang tercantum di requirements.txt. Ini menciptakan sebuah image yang identik dan dapat dijalankan di mana saja, memastikan tidak ada lagi masalah "berjalan di mesin saya, tetapi tidak di server".
ii. Portabilitas Maksimal: Seluruh aplikasi, termasuk server Flask, skrip deteksi objek, dan konfigurasinya, dikemas menjadi sebuah image Docker. Image ini dapat dengan mudah dipindahkan dan dijalankan di server lain hanya dengan satu perintah, menyederhanakan proses migrasi dan pemulihan bencana.
iii. Isolasi Proses yang Aman: Setiap komponen sistem (misalnya, aplikasi web dan skrip detektor) dapat dijalankan dalam kontainer terpisah yang terisolasi. Mereka berkomunikasi melalui jaringan virtual yang didefinisikan oleh Docker, yang meningkatkan keamanan dan stabilitas. Jika satu kontainer gagal, itu tidak akan mempengaruhi yang lain.
iv. Efisiensi untuk On-Premise: Dibandingkan dengan VM, kontainer secara drastis lebih hemat sumber daya. Hal ini memungkinkan beberapa

layanan untuk berjalan secara efisien pada satu server fisik tanpa memerlukan perangkat keras kelas atas, sejalan dengan model deployment on-premise yang hemat biaya.

3. Web Server

Komponen web server adalah tulang punggung dari sisi backend, bertugas untuk menerima, memproses, dan merespons permintaan dari berbagai klien. Arsitekturnya terdiri dari kerangka kerja aplikasi dan stack server produksi

a. Kerangka Kerja Aplikasi (Application Framework)


Pilihan dan Justifikasi: Sistem ini dibangun menggunakan Flask, sebuah micro-framework Python. Flask dipilih karena filosofinya yang minimalis namun dapat diperluas. Ini memberikan kebebasan untuk membangun fungsionalitas yang dibutuhkan—seperti API endpoint /sensordata untuk menerima data dari sensor, /get_live_data untuk dashboard, dan /get_people_count untuk tool AI—tanpa memaksakan struktur atau komponen yang tidak perlu. Sifatnya yang ringan menjadikannya pilihan ideal untuk aplikasi dengan tujuan spesifik seperti server IRIS.

b. Stack Server Produksi (Production Server Stack)

Pilihan dan Justifikasi: Meskipun server pengembangan bawaan Flask memadai untuk pengujian, lingkungan produksi memerlukan stack yang lebih tangguh dan berkinerja tinggi. Oleh karena itu, arsitektur yang dipilih adalah kombinasi Gunicorn dan Nginx.

i. Gunicorn (WSGI Server): Gunicorn bertindak sebagai server aplikasi yang menjalankan kode Flask. Peran utamanya adalah mengelola worker processes, memungkinkannya menangani beberapa permintaan dari klien secara bersamaan (concurrently). Ini adalah peningkatan krusial dari server pengembangan Flask yang bersifat single-threaded dan tidak dapat menangani beban kerja nyata.

ii. Nginx (Reverse Proxy): Nginx ditempatkan di depan Gunicorn sebagai gerbang utama untuk semua lalu lintas masuk. Nginx memainkan beberapa peran vital:

iii. Proxy Penerus: Nginx menerima semua permintaan dari luar dan meneruskan permintaan dinamis (panggilan API) ke Gunicorn.

iv. Penyaji File Statis: Nginx sangat efisien dalam menyajikan file statis secara langsung kepada klien, seperti style.css dan chart.js. Ini secara drastis mengurangi beban pada aplikasi Python, membebaskannya untuk fokus pada logika bisnis.

v. Keamanan dan Load Balancing: Nginx dapat dikonfigurasi sebagai lapisan keamanan pertama untuk menangani terminasi SSL/TLS, pembatasan laju permintaan (rate limiting), dan dapat berfungsi sebagai

load balancer jika di masa depan aplikasi perlu dijalankan di beberapa instance Gunicorn.

4. Aplikasi Klien

Aplikasi klien adalah komponen perangkat lunak yang berfungsi sebagai titik interaksi dengan pengguna akhir atau sebagai pengirim data ke server. Dalam arsitektur IRIS, terdapat dua jenis klien yang berbeda secara fungsional dan teknologi.

a. Klien Perangkat Keras IoT (ESP32)

Deskripsi: Klien ini berupa firmware yang ditanamkan pada setiap mikrokontroler ESP32 di setiap ruangan. Tugas utamanya adalah secara kontinu membaca data dari sensor-sensor yang terhubung (DHT11 untuk suhu dan MQ-2 untuk asap), memformat data tersebut ke dalam struktur JSON, dan mengirimkannya ke endpoint /sensordata di server pusat melalui jaringan Wi-Fi.

i. Teknologi:

Bahasa Pemrograman: C++ dengan kerangka kerja Arduino. Ini adalah standar de-facto untuk pengembangan pada platform ESP32, menyediakan abstraksi tingkat tinggi untuk operasi perangkat keras dan didukung oleh ekosistem pustaka yang sangat luas.

ii. Pustaka Kunci:

1. WiFi.h: Untuk mengelola koneksi ke jaringan Wi-Fi lokal.
2. HTTPClient.h: Untuk membuat permintaan HTTP POST ke server Flask, yang merupakan mekanisme pengiriman data utama.
3. ArduinoJson.h: Pustaka esensial untuk serialisasi data (mengubah pembacaan sensor menjadi format teks JSON) dan deserialisasi (jika diperlukan untuk menerima konfigurasi dari server).
4. Pustaka Sensor (DHT.h, Adafruit_Sensor.h): Untuk berinteraksi langsung dengan sensor suhu dan kelembapan.

b. Klien Aplikasi Web (Dasbor Pemantauan)

Deskripsi: Klien ini adalah sebuah Single-Page Application (SPA) yang diakses oleh admin keamanan melalui peramban web. Dasbor ini menyajikan visualisasi data real-time dari semua sensor, status setiap ruangan, grafik historis, dan status darurat keseluruhan sistem.

i. Teknologi:
    1. HTML (HyperText Markup Language): Digunakan untuk membangun struktur semantik dari halaman dasbor, termasuk penempatan kartu-kartu ruangan, tajuk, dan kaki halaman.
    2. CSS (Cascading Style Sheets): Bertanggung jawab atas seluruh aspek visual dan tata letak, termasuk skema warna, tipografi, desain responsif, dan animasi status darurat, memastikan antarmuka yang intuitif dan mudah dipahami.
    3. JavaScript (Vanilla JS): Merupakan otak dari fungsionalitas dinamis dasbor. Skrip ini secara periodik (setiap 2 detik, seperti yang tertera pada kode) melakukan pemanggilan fetch ke endpoint /get_live_data pada server. Setelah menerima data JSON, skrip ini memanipulasi Document Object Model (DOM) untuk memperbarui nilai-nilai, mengubah kelas CSS (misalnya, dari status-NORMAL ke status-ALERT_FIRE), dan me-render ulang grafik tanpa memerlukan muat ulang halaman.

ii. *Package* JavaScript Pihak Ketiga:
    1. Chart.js: Sebuah pustaka grafik yang kuat dan fleksibel, digunakan untuk membuat grafik garis (line charts) yang menampilkan tren data suhu dan asap dari waktu ke waktu untuk setiap ruangan.
    2. Luxon: Sebuah pustaka untuk bekerja dengan tanggal dan waktu. Dalam sistem ini, Luxon digunakan untuk mengubah stempel waktu ISO dari server menjadi format yang relatif dan mudah dibaca manusia (contoh: "beberapa detik yang lalu"), memberikan informasi yang lebih intuitif tentang kesegaran data.

5. Database Server

Database server berfungsi sebagai repositori data persistensi untuk sistem. Dalam arsitektur IRIS, ada dua jenis database yang digunakan untuk tujuan yang berbeda: database relasional untuk data operasional dan ***vector database*** untuk *knowledge base* AI.

a. Database Operasional

Alternatif Teknologi:

i. RDBMS berbasis Server (MySQL/PostgreSQL): Sistem database relasional yang berjalan sebagai proses server terpisah, dirancang untuk skalabilitas tinggi dan konkurensi.
ii. Database NoSQL (MongoDB): Database non-relasional yang menawarkan fleksibilitas skema, cocok untuk data tidak terstruktur.
iii. Database Embedded (SQLite): Mesin database relasional serverless yang terintegrasi langsung ke dalam aplikasi.

Pilihan dan Justifikasi:


Teknologi yang dipilih adalah SQLite. Pilihan ini sangat strategis dan cocok untuk model deployment *on-premise* yang diusulkan:

1. Arsitektur Serverless: SQLite tidak memerlukan instalasi, konfigurasi, atau proses server yang berjalan terpisah. Ia berfungsi sebagai pustaka yang dipanggil langsung oleh aplikasi

Python, secara drastis menyederhanakan setup dan pemeliharaan.

2. Portabilitas Tinggi: Seluruh database disimpan dalam satu file (misalnya, fire_incident.db). Hal ini membuat proses backup, restorasi, dan penyalinan—sebuah kebutuhan fungsional untuk menyediakan data bagi agen LLM—menjadi sangat trivial.
3. Kecukupan untuk Beban Kerja: Beban kerja sistem ini terdiri dari penulisan data sensor yang teratur dan pembacaan periodik untuk dasbor. SQLite sangat efisien untuk beban kerja dengan konkurensi rendah hingga menengah dan lebih dari mampu menangani volume data dari puluhan sensor dalam skenario ini.

b. *Vector Database* untuk *Knowledge Base*

Deskripsi: Vector database adalah jenis database yang dirancang khusus untuk menyimpan, mengelola, dan melakukan pencarian pada data dalam bentuk vektor berdimensi tinggi. Vektor ini merupakan representasi numerik dari data tak terstruktur seperti teks, yang dihasilkan oleh model embedding.


Pilihan dan Justifikasi: Sistem ini memanfaatkan Qdrant, seperti yang terdefinisi dalam alur kerja n8n. Qdrant dipilih untuk mengimplementasikan pola arsitektur *Retrieval-Augmented Generation* (RAG), yang merupakan inti dari kecerdasan AI dalam sistem ini.

Peran dalam Sistem:

i. Penyimpanan *Knowledge Base*: Informasi statis seperti denah kantor, detail sistem pemadam kebakaran, dan prosedur darurat diubah menjadi

vektor numerik (embeddings) oleh model AI (dalam hal ini, Embeddings Google Gemini). Vektor-vektor ini kemudian disimpan dan diindeks dalam koleksi di Qdrant.

ii. Pencarian Kontekstual *(Retrieval)*: Ketika insiden terjadi, *AI Agent* di n8n tidak langsung bertanya ke LLM. Sebaliknya, ia mengambil data dinamis (misalnya, "kebakaran di R202") dan menggunakannya untuk melakukan pencarian kemiripan (*similarity search*) di Qdrant.

iii. Augmentasi Prompt: Qdrant akan mengembalikan potongan-potongan teks dari knowledge base yang paling relevan secara semantik dengan kueri (misalnya, data tentang R202, lokasinya, rute evakuasi terdekat, dan keberadaan sistem pemadam khusus di sana).

iv. Generasi Respons: Potongan teks yang relevan ini kemudian "disuntikkan" ke dalam *prompt* yang dikirim ke LLM (Gemini). Dengan konteks tambahan yang kaya ini, LLM dapat menghasilkan respons yang sangat akurat dan relevan dengan situasi, seperti rute evakuasi yang menghindari area berbahaya.

Justifikasi: Penggunaan vector database seperti **Qdrant** adalah suatu keharusan untuk implementasi RAG yang efisien. Tanpa itu, pencarian informasi relevan dari dokumen teks yang besar akan sangat lambat dan tidak praktis, sehingga mustahil bagi AI untuk memberikan respons yang cerdas dan *context-aware* secara *real-time.*

6. Layanan AI

Kecerdasan Buatan (AI) adalah komponen sentral yang mengubah IRIS dari sistem peringatan reaktif menjadi penasihat respons insiden yang proaktif dan cerdas. Penerapan AI, yang mencakup Large Language Models (LLM), Computer Vision, dan platform otomasi, memungkinkan sistem untuk tidak hanya mendeteksi bahaya, tetapi juga memahami konteks, melakukan penalaran, dan mengoordinasikan respons yang kompleks secara otonom. Arsitektur AI dalam sistem ini dibagi menjadi tiga pilar teknologi utama.

a. Model Bahasa (Large Language Model): Google Gemini

Landasan Teori: Large Language Model (LLM) adalah model AI yang dilatih pada miliaran dokumen teks dan kode, memungkinkannya untuk memahami, meringkas, dan menghasilkan bahasa manusia dengan tingkat kefasihan yang tinggi. Berbasis arsitektur Transformer, LLM mampu menangkap konteks yang kompleks dan melakukan penalaran logis berdasarkan informasi yang diberikan dalam sebuah prompt. Dalam sistem ini, LLM tidak hanya menghasilkan teks, tetapi bertindak sebagai otak analitis untuk pengambilan keputusan kritis.

Alternatif Teknologi: OpenAI GPT-4, Anthropic Claude.


Pilihan dan Justifikasi: Sistem ini secara spesifik menggunakan model Google Gemini. Pilihan ini didasarkan pada analisis bahwa Gemini menawarkan integrasi yang paling mulus, berperforma tinggi, dan efisien, terutama jika sistem di masa depan akan di-deploy pada Google Cloud Platform (GCP). Gemini memainkan dua peran vital dalam arsitektur IRIS:

i. Generator Rute Evakuasi (dalam n8n): Ketika alur kerja darurat di n8n aktif, Gemini dipanggil untuk setiap karyawan. Ia menerima sebuah prompt yang telah diperkaya (augmented) dengan knowledge base statis (denah gedung, detail pintu keluar, dll.) dan data dinamis (lokasi spesifik kebakaran dan lokasi karyawan saat itu). Berdasarkan kombinasi data ini,

Gemini melakukan penalaran untuk menghasilkan instruksi rute evakuasi yang paling aman dan dipersonalisasi, yang kemudian dikirim melalui Telegram.

ii. Agen Suara Cerdas (dalam Custom LLM Server): Untuk panggilan ke pemadam kebakaran, sistem menggunakan server LLM kustom yang ditenagai oleh Gemini, seperti yang diimplementasikan pada gemini_llm.py. Server ini terintegrasi dengan Retell AI dan memiliki akses ke tools atau fungsi khusus (get_initial_incident_details dan get_people_count_from_db). Tools ini memungkinkan Gemini untuk secara aktif melakukan kueri ke database operasional secara real-time untuk mengambil detail insiden awal dan jumlah orang yang terdeteksi di dalam gedung. Kemampuan ini mengubah Gemini menjadi agen informan yang dinamis, mampu menjawab pertanyaan spesifik dari petugas pemadam kebakaran dengan data terbaru.

b. Platform Otomasi dan Orkestrasi AI: n8n.io


Landasan Teori: n8n.io adalah platform otomasi alur kerja low-code yang berfungsi sebagai perekat digital, menghubungkan berbagai aplikasi dan layanan (API) untuk menjalankan proses yang kompleks secara otomatis. Pengguna dapat merancang alur kerja secara visual dengan menghubungkan "node-node" yang masing-masing memiliki fungsi spesifik.

Peran dalam Sistem: n8n adalah orkestrator utama dari respons darurat IRIS. Perannya adalah sebagai berikut:

i. Pemicu (Trigger): Alur kerja dimulai ketika node Webhook menerima permintaan HTTP POST dari app.py, yang menandakan bahwa ambang batas kebakaran telah terlampaui.
ii. Pengambilan Data: Node Airtable segera mengambil data kontak seluruh karyawan, termasuk nama, ID Telegram, dan lokasi ruangan mereka.
iii. Orkestrasi AI: Alur kerja kemudian melakukan iterasi (loop) untuk setiap kontak. Di dalam loop, ia memanggil node LLM yang telah dikonfigurasi dengan prompt Gemini untuk menghasilkan rute evakuasi yang unik bagi setiap karyawan.
iv. Notifikasi Multi-kanal: Hasil dari LLM kemudian diteruskan ke node Telegram untuk mengirim pesan notifikasi yang telah dipersonalisasi. Secara paralel, alur kerja juga dapat memicu node HTTP Request untuk melakukan panggilan otomatis ke pemadam kebakaran melalui API eksternal seperti Twilio via Retell AI.

c. Computer Vision: YOLOv8


Landasan Teori: YOLO (You Only Look Once) adalah keluarga algoritma deteksi objek yang dikenal karena kemampuannya mendeteksi objek dalam gambar atau video secara real-time dengan kecepatan dan akurasi yang tinggi.

Arsitektur single-shot detector-nya memungkinkan pemrosesan frame video dengan sangat cepat, menjadikannya ideal untuk aplikasi yang memerlukan latensi rendah.

Pilihan dan Justifikasi: Sistem ini menggunakan YOLOv8, versi terbaru dan tercanggih dari keluarga YOLO. YOLOv8 dipilih karena keseimbangan yang sangat baik antara kecepatan dan akurasi, yang krusial dalam situasi darurat di mana informasi jumlah penghuni harus diperbarui secepat mungkin.

Peran dalam Sistem:

i. Aktivasi Kondisional: Subsistem deteksi objek, yang diimplementasikan dalam skrip detector.py, tidak berjalan terus-menerus untuk menghemat sumber daya. Ia diaktifkan oleh server utama (app.py) hanya ketika kondisi darurat kebakaran terverifikasi.
ii. Deteksi dan Penghitungan Real-time: Setelah aktif, skrip ini menggunakan model YOLOv8 yang telah di-fine-tuned (disimpan sebagai best.pt) untuk menganalisis stream video dari kamera di setiap ruangan. Model ini secara spesifik dilatih untuk mendeteksi dan menghitung jumlah orang (human).
iii. Pembaruan Database Dinamis: Hasil penghitungan jumlah orang diperbarui ke dalam tabel people_detection di database fire_incident.db secara periodik setiap 5 detik. Ini menciptakan sumber data dinamis yang mencerminkan situasi kepadatan penghuni di seluruh gedung selama insiden berlangsung. Data ini kemudian dapat diakses oleh agen suara AI (Gemini) untuk dilaporkan kepada tim penyelamat.

7. Protokol Komunikasi

Protokol komunikasi adalah seperangkat aturan yang mengatur bagaimana data dipertukarkan antara berbagai komponen dalam sistem. Arsitektur IRIS mengadopsi pendekatan polyglot protocol, menggunakan beberapa protokol berbeda yang dipilih secara spesifik untuk mengoptimalkan efisiensi, keandalan, dan kecepatan pada setiap jalur komunikasi.

Alternatif Teknologi:

a. HTTP (Hypertext Transfer Protocol): Protokol permintaan-respons (request-response) yang menjadi tulang punggung World Wide Web. Bersifat stateless dan mudah diimplementasikan.
b. MQTT (Message Queuing Telemetry Transport): Protokol ringan berbasis model publish-subscribe, dirancang khusus untuk perangkat IoT dengan sumber daya terbatas dan jaringan yang tidak andal.
c. WebSockets: Protokol yang menyediakan kanal komunikasi dua arah (full-duplex) secara persisten melalui satu koneksi TCP, ideal untuk aplikasi real-time.

Pilihan dan Justifikasi:

Sistem ini mengimplementasikan kombinasi protokol berikut:

a. HTTP (untuk Sensor dan Webhook)

   Jalur Komunikasi:

   - Perangkat IoT (ESP32) → Server Flask
   - Server Flask (app.py) → Platform Otomasi (n8n)

   Justifikasi: HTTP dipilih untuk jalur ini karena sifat interaksinya yang transaksional. Setiap pengiriman data dari sensor adalah sebuah peristiwa diskrit yang dikirim melalui permintaan POST ke endpoint /sensordata. Demikian pula, ketika kondisi darurat terdeteksi, server hanya perlu mengirim satu sinyal pemicu (trigger) ke webhook n8n. Kesederhanaan implementasi HTTP pada firmware ESP32 dan di sisi server Flask, ditambah dengan keandalan jaringan lokal (LAN) dalam deployment on-premise, menjadikan HTTP pilihan yang paling praktis dan andal untuk tugas ini.

b. WebSockets (untuk Dasbor Real-time)

Jalur Komunikasi: Server Flask ↔ Dasbor Web Klien (Peramban)

Justifikasi: Untuk menampilkan data secara live di dasbor pemantauan, diperlukan komunikasi dengan latensi sangat rendah. Seperti yang digambarkan dalam diagram sekuens sistem, dasbor akan membuka koneksi WebSocket persisten ke server. Melalui koneksi ini, server dapat secara aktif "mendorong" (push) pembaruan data sensor ke dasbor setiap beberapa detik. Pendekatan ini jauh lebih efisien daripada menggunakan HTTP polling, di mana klien harus terus-menerus mengirim permintaan baru. WebSockets mengurangi overhead jaringan secara signifikan dan memastikan data yang ditampilkan di dasbor selalu yang terbaru.

## Penerapan AI dalam Sistem

Dalam arsitektur solusi sistem peringatan kebakaran cerdas ini, Kecerdasan Buatan (AI) memainkan peran sentral dan krusial. AI berfungsi sebagai otak dari sistem respons, mengubah peringatan sensorik mentah menjadi instruksi evakuasi yang personal, kontekstual, dan dapat ditindaklanjuti. Penerapan AI, yang dalam kasus ini menggunakan model bahasa besar (LLM) seperti Google Gemini melalui API dan mengadopsi arsitektur RAG (Retrieval-Augmented Generation), memungkinkan sistem untuk melampaui alarm tradisional dan memberikan panduan cerdas yang dinamis. Berbeda dengan model AI standar yang hanya merespons berdasarkan informasi yang diberikan dalam sebuah prompt, model RAG secara proaktif mengambil (retrieve) informasi relevan dari berbagai sumber data sebelum menghasilkan (generate) responnya.

AI dalam sistem ini tidak hanya sekadar pemicu notifikasi, melainkan bertindak sebagai Penasihat Evakuasi Cerdas (Intelligent Evacuation Advisor). Peran ini dapat dipecah menjadi tiga fungsi utama. Pertama, AI menjadi analis situasi darurat dimana AI bertugas untuk memahami konteks dari sebuah insiden. Ini tidak hanya mengetahui bahwa ada kebakaran,

tetapi juga lokasi persisnya dan lokasi karyawan yang perlu dievakuasi. Kedua, AI menjadi mesin pengambil keputusan rute. Ini adalah peran paling vital. Berdasarkan lokasi kebakaran dan lokasi karyawan, AI secara dinamis menentukan rute evakuasi yang paling aman dan efisien. AI mampu melakukan penalaran untuk menghindari area berbahaya dan memprioritaskan penggunaan jalur darurat yang telah ditentukan. Ketiga, generator instruksi. AI mengubah keputusan rute yang kompleks menjadi instruksi yang sederhana, jelas, dan mudah dipahami oleh manusia dalam bahasa alami. Alih-alih memberikan kode atau data mentah, AI menghasilkan kalimat lengkap yang bisa langsung dikirimkan sebagai pesan peringatan. Selain perannya sebagai penasihat evakuasi untuk karyawan, AI juga memiliki peran krusial kedua sebagai Pelapor Insiden Cerdas untuk pihak eksternal, di mana ia secara otonom dapat menelepon pihak berwenang dan melaporkan detail insiden secara verbal

Agar dapat menjalankan perannya, AI diberi dua jenis data: Data Statis (Knowledge Base) yang berfungsi sebagai pengetahuannya, dan Data Dinamis yang merupakan informasi spesifik untuk setiap insiden.

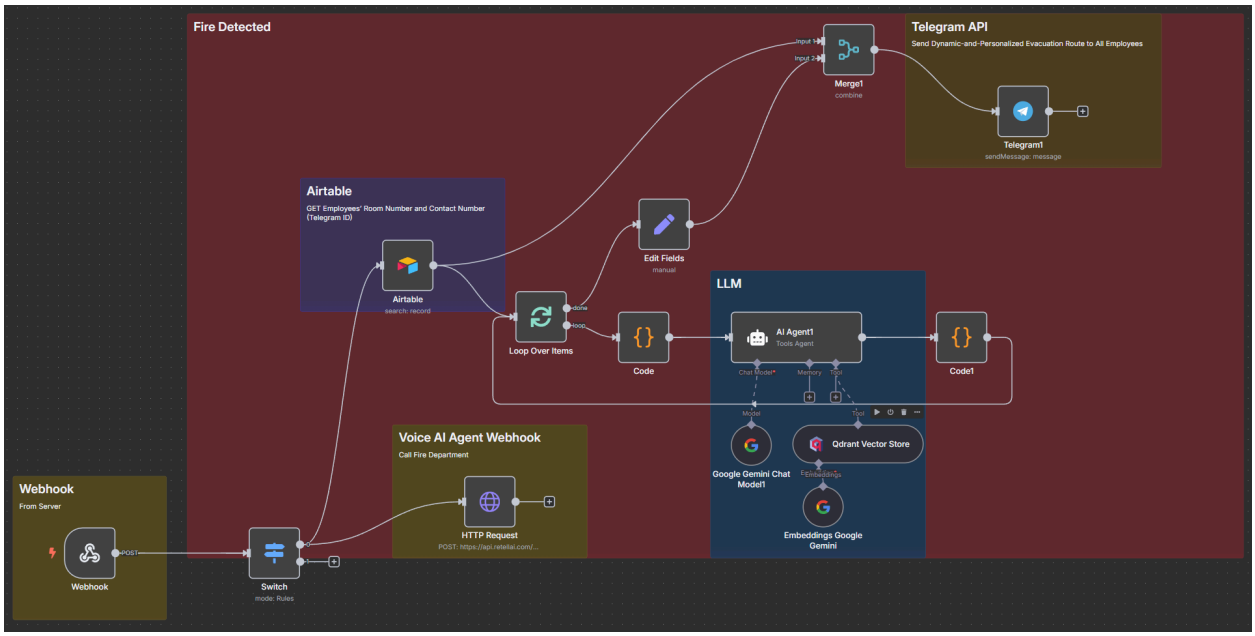
Data Statis berupa *knowledge base* dari gedung perusahaan. Data ini bertindak sebagai buku panduan bagi LLM mengenai gedung Graha Inovasi. Data ini diekstrak dari dokumen yang telah disediakan dan dimasukkan ke dalam System Message pada prompt LLM. Informasi yang diekstrak dari *knowledge base* mencakup denah dan tata letak ruang, rute keluar terdekat, dan detail tangga darurat.

Data Dinamis juga termasuk pada *knowledge base.* Data dinamis berisi informasi insiden secara real-time. Ini adalah data yang berubah-ubah untuk setiap insiden dan diberikan kepada AI melalui User Message (Prompt) pada setiap eksekusi. Data dinamis ini mencakup dua sumber utama: pertama, detail insiden awal (lokasi, suhu, asap) yang dicatat saat alarm pertama kali terpicu. Kedua, dan yang terpenting, adalah data hunian *real-time* yang bersumber dari database SQL. Database ini secara dinamis diperbarui oleh Subsistem Deteksi Objek (YOLOv8) yang menghitung jumlah orang di setiap ruangan. Berbeda dengan arsitektur lama, data dinamis ini tidak lagi "didorong" ke AI, melainkan "ditarik" secara proaktif oleh AI itu sendiri menggunakan *tools* yang telah didefinisikan sebagai bagian dari arsitektur RAG

Berikut adalah alur kerja detail bagaimana AI beroperasi dalam workflow n8n setelah insiden terdeteksi:

1. Aktivasi: Workflow n8n menerima sinyal dari app.py melalui Webhook. Sebuah node IF mengonfirmasi bahwa data sensor (suhu dan asap) telah melampaui ambang batas kritis.
2. Aktivasi Paralel: Secara bersamaan saat n8n diaktifkan, Web Server juga memicu Subsistem Deteksi Objek untuk mulai menganalisis *stream* video dan mengisi database SQL dengan data jumlah orang secara *real-time*.
3. Inisiasi Panggilan Suara AI: Sebagai salah satu langkah pertama dalam alur kerja n8n, sebuah panggilan suara AI langsung diinisiasi ke pihak berwenang melalui layanan eksternal. Proses inilah yang akan menggunakan arsitektur RAG untuk melaporkan insiden secara cerdas.
4. Pengambilan Data Kontak: Node Airtable mengambil seluruh daftar kontak yang relevan, termasuk nama, TelegramChatID, dan Ruangan mereka. Workflow kini memiliki daftar orang yang perlu diberi tahu dan di mana lokasi mereka masing-masing.
5. Eksekusi AI (Berulang untuk Setiap Kontak): Workflow kemudian memasuki sebuah loop, di mana node LLM dieksekusi satu per satu untuk setiap kontak yang diambil dari Airtable.
6. Kontekstualisasi dan Pembentukan Prompt: Untuk setiap eksekusi, n8n membentuk prompt yang lengkap untuk AI. System Message: Berisi peran AI, aturan, dan seluruh data statis dari Knowledge Base yang telah diekstrak. User Message: Berisi data dinamis: roomId kebakaran (dari Webhook) dan Ruangan karyawan yang sedang diproses saat itu (dari Airtable).
7. Proses Penalaran AI: Dengan semua data ini, AI melakukan proses penalaran. Misalnya: "Api ada di R101. Karyawan ada di R207. Berdasarkan denah gedung, rute terdekat dari R207 adalah S101 via H202. S101 tidak berada di dekat api (R101) dan merupakan tangga darurat yang aman. S101 mengarah ke pintu keluar EX001. Maka, rute yang direkomendasikan adalah 'Dari ruangan Anda (R207), segera menuju Tangga Darurat S101, turun ke Lantai Dasar dan keluar melalui pintu EX001'."

8. Generasi Respon: AI menghasilkan respons berupa teks dalam bahasa alami yang berisi instruksi rute evakuasi yang telah dipersonalisasi tersebut.
9. Integrasi ke Notifikasi: Workflow n8n mengambil teks respons dari AI ini dan memasukkannya ke dalam template pesan Telegram. Pesan akhir yang berisi detail insiden dan rute evakuasi personal dari AI kemudian dikirim ke TelegramChatID karyawan yang bersangkutan.




## Web Server/Database Server/Client

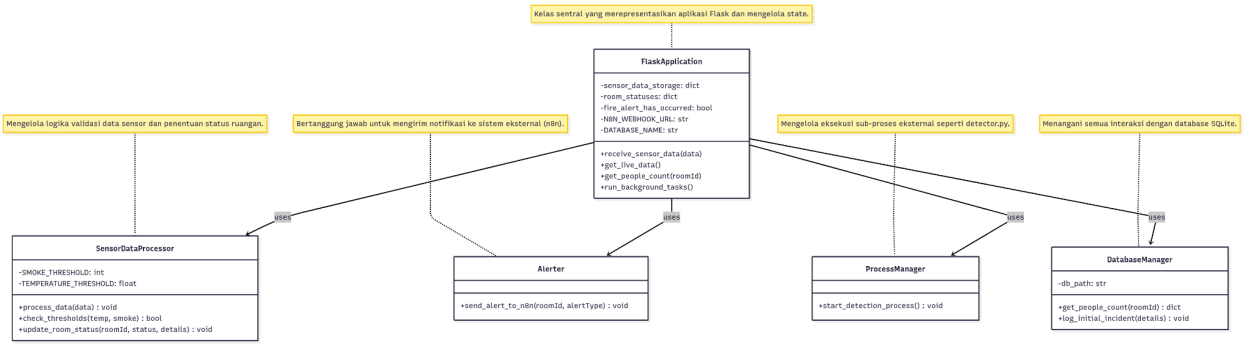
Bagian ini merinci desain teknis dari tiga komponen utama dalam arsitektur perangkat lunak IRIS: Web Server sebagai pusat logika, Database Server sebagai repositori data, dan berbagai Klien yang berinteraksi dengan sistem.

1. Web Server

Web server bertindak sebagai jantung dari sistem IRIS.  Ia tidak hanya berfungsi sebagai titik akhir API untuk menerima data, tetapi juga sebagai orkestrator yang memulai alur kerja darurat dan menyajikan data untuk pemantauan.

Fitur-fitur Web Server:

a. Penerimaan Data Sensor: Menyediakan endpoint API (/sensordata) yang menerima data suhu dan asap dari setiap perangkat IoT (ESP32) melalui permintaan HTTP POST dengan payload JSON.
b. Validasi dan Pemrosesan Data: Melakukan validasi data yang masuk, memeriksa nilai suhu dan asap terhadap ambang batas yang telah ditentukan (TEMPERATURE_THRESHOLD dan SMOKE_THRESHOLD) untuk mendeteksi potensi kebakaran.
c. Manajemen Status Real-time: Secara aktif mengelola dan memperbarui status setiap ruangan (misalnya, NORMAL, ALERT_FIRE, STALE, ALERT_MISSING) dalam memori untuk penyajian data secara langsung.
d. Pemicu Alur Kerja Darurat: Ketika kondisi kebakaran terverifikasi, server secara otomatis mengirimkan sinyal peringatan ke webhook n8n untuk memulai alur kerja notifikasi dan panggilan darurat.
e. Aktivasi Subsistem Pendukung: Pada deteksi kebakaran pertama, server mengaktifkan skrip computer vision untuk mulai menghitung jumlah orang dan mencatat detail insiden awal ke dalam database khusus untuk diakses oleh agen AI.
f. Penyajian Data untuk Dasbor: Menyediakan endpoint API (/get_live_data) yang menyajikan data terstruktur—termasuk data terkini, data historis untuk grafik, dan status ruangan—ke aplikasi klien dasbor web.
g. Penyediaan Endpoint untuk Tool AI: Menyediakan endpoint API (/get_people_count) yang berfungsi sebagai tool yang dapat dipanggil oleh agen suara AI (Retell) untuk mendapatkan data jumlah orang secara real-time dari database.




2. Database Server

Sistem IRIS menggunakan dua database SQLite terpisah untuk mencatat jenis data yang berbeda, memastikan data operasional dan data insiden kritis tersimpan secara efisien.

Entity-Relationship Diagram (ERD):

Sistem ini memiliki dua entitas utama yang tidak saling berhubungan langsung:

a. InitialIncident: Mencatat detail spesifik saat kebakaran pertama kali terdeteksi. Entitas ini hanya ditulis satu kali per insiden.
b. PeopleDetection: Menyimpan data jumlah orang yang terdeteksi di setiap ruangan, yang diperbarui secara periodik selama mode darurat aktif.

Skema Database:

- Database Insiden Awal (incident_details.db)

  Tabel ini bertujuan untuk menyimpan satu catatan definitif dari peristiwa pemicu kebakaran.

  Tabel: initial_incident

  - id (INTEGER, PRIMARY KEY, AUTOINCREMENT): Kunci unik untuk setiap catatan insiden.

    - roomId (TEXT, NOT NULL): ID ruangan tempat alarm pertama kali terdeteksi.
    - temperature (REAL): Nilai suhu (°C) saat alarm terpicu.
    - smokeValue (INTEGER): Nilai sensor asap saat alarm terpicu.
    - alertTime (TEXT, NOT NULL): Stempel waktu dalam format ISO 8601 saat alarm terpicu.

- Database Deteksi Orang (fire_incident.db)

  Tabel ini berfungsi sebagai sumber data dinamis untuk pemantauan penghuni.

  Tabel: people_detection

    - id (INTEGER, PRIMARY KEY, AUTOINCREMENT): Kunci unik internal.
    - ruangan (TEXT, NOT NULL, UNIQUE): ID unik untuk setiap ruangan yang dipantau oleh kamera.
    - peopleCount (INTEGER): Jumlah orang yang terdeteksi. Nilai default adalah -1 sebelum deteksi dimulai.
    - lastDetectedTimeStamp (TEXT): Stempel waktu ISO 8601 saat orang terakhir kali terdeteksi.
    - lastUpdateTimeStamp (TEXT): Stempel waktu ISO 8601 saat baris ini terakhir diperbarui oleh skrip detektor.

## ☆ Desain Peluncuran

Pemilihan strategi peluncuran (deployment) adalah keputusan krusial yang berdampak langsung pada keandalan, kecepatan respons, dan biaya operasional sistem. Setelah mempertimbangkan berbagai faktor secara cermat, strategi deployment yang dipilih untuk solusi ini adalah On-Premise.

Dengan pemilihan On-Premise, setiap komponen perangkat lunak—seperti backend, agen otomasi AI, dan database—akan dijalankan pada sebuah server fisik yang berlokasi dan dikelola sepenuhnya oleh internal perusahaan di dalam gedung itu sendiri. Server ini akan terhubung langsung ke Jaringan Area Lokal (LAN) gedung, memungkinkannya untuk berkomunikasi secara langsung dengan semua perangkat IoT (ESP32) yang tersebar di seluruh ruangan.

Keputusan untuk memilih strategi on-premise didasarkan pada empat pilar utama: keandalan & independensi, efisiensi biaya operasional, latensi komunikasi, dan keamanan data.

1. Keandalan dan Independensi

   Dengan server on-premise, komunikasi antara sensor IoT dan server pusat terjadi sepenuhnya melalui jaringan lokal (LAN/WLAN). Selama jaringan internal gedung berfungsi, sistem deteksi dan pemrosesan peringatan akan tetap berjalan 100%, memastikan peringatan internal dan panduan evakuasi tetap dapat dikirimkan. Sistem hanya memerlukan koneksi internet untuk tugas outbound (keluar) seperti notifikasi ke Telegram atau panggilan ke Damkar, yang bisa menjadi prioritas sekunder setelah peringatan internal tersampaikan. Tetapi, saat kebakaran, ada kemungkinan yang besar dalam infrastruktur komunikasi seperti kabel fiber optik yang menyediakan koneksi internet ikut rusak.

2. Biaya Operasional

   Biayanya sebagian besar bersifat modal di awal (CapEx) untuk pembelian perangkat keras server (misalnya, sebuah Mini PC atau server berdaya rendah). Setelah itu, biaya operasionalnya sangat minimal, hanya mencakup listrik dan pemeliharaan sesekali. Untuk kasus penggunaan data frekuensi tinggi, model ini jauh lebih ekonomis dalam jangka waktu 1-3 tahun.

   Dibandingkan dengan cloud platform, pada project ini yang memiliki pengumpulan data yang banyak di setiap waktu, maka cloud platform diproyeksikan untuk lebih mahal dibandingkan dengan on-premise meskipun biaya awal On-premise lebih kecil di awal. Platform cloud umumnya menggunakan model pay-as-you-go. Dengan puluhan

perangkat mengirim data setiap 5 detik, ini akan menghasilkan jutaan request per bulan. Biaya akan mencakup biaya untuk mengirim dan menerima data (data ingress/egress), biaya per eksekusi jika menggunakan arsitektur serverless (function executions), dan biaya penyimpanan data sensor historis. Biaya ini bersifat operasional (OpEx) yang terus berjalan dan sulit diprediksi secara akurat.

3. Latensi Komunikasi yang Minimal

   Komunikasi melalui jaringan lokal (LAN) memiliki latensi yang sangat rendah (biasanya <10 ms). Komunikasi ke server cloud, meskipun cepat, harus melalui banyak hop di internet, yang secara inheren menambahkan latensi (bisa 50-200 ms atau lebih). Dengan memproses data secara lokal, keputusan untuk memicu alarm dapat dibuat secepat mungkin.

4. Keamanan Data

   Dengan deployment on-premise, data sensor tidak pernah meninggalkan perimeter fisik dan jaringan gedung kecuali untuk notifikasi darurat. Ini memberikan kontrol penuh atas siapa yang dapat mengakses data dan mengurangi permukaan serangan (attack surface) dari ancaman eksternal.

Dalam skenario menggunakan platform cloud, maka terdapat beberapa pilihan dalam memilih cloud provider untuk project ini yakni Amazon Web Services (AWS), Google Cloud Platform (GCP), dan Microsoft Azure. Ketiga cloud provider diatas merupakan pemain besar dalam bisnis cloud provider yang memiliki sistem maintenance dan keamanan yang ternama dalam beberapa tahun ini. Dalam menganalisa pemilihan cloud provider, terdapat beberapa proses yang akan dibandingkan oleh setiap provider untuk menentukan sebuah pilihan terbaik untuk cloud provider. Proses yang akan dilakukan untuk perbandingan mencakupi layanan IoT, integrasi layanan AI, model penetapan harga, dan kemudahan pengguna dan ekosistem untuk pengguna.

| Kriteria | Amazon Web Services (AWS) | Google Cloud Platform (GCP) | Microsoft Azure |
|---|---|---|---|
| Layanan IoT | Menyediakan AWS IoT Core, platform yang diakui sebagai standar industri berkat kelengkapan fitur dan kematangannya yang superior. | Menawarkan Google Cloud IoT Core yang berfokus pada kesederhanaan dan integrasi mendalam dengan ekosistem data dan AI Google. | Menyajikan Azure IoT Hub, solusi yang dirancang untuk keandalan tingkat enterprise dan sangat diadopsi di lingkungan industri serta korporat. |
| Integrasi Layanan AI | Memberikan fleksibilitas tinggi melalui Amazon Bedrock, sebuah layanan terkelola yang memungkinkan integrasi dengan berbagai pilihan model AI. | Menawarkan keuntungan superior karena integrasi *native* dengan Gemini melalui platform Vertex AI, memastikan performa optimal dan latensi minimal. | Memiliki kemitraan strategis dengan OpenAI, memberikan akses teroptimalkan ke model-model GPT melalui Azure OpenAI Service yang aman dan terkelola. |
| Model Harga | Menggunakan model pay-as-you-go yang sangat granular, memberikan kontrol biaya yang detail namun berpotensi menjadi | Menerapkan model pay-as-you-go yang dikenal lebih transparan dan mudah diprediksi, menyederhanakan proses perencanaan anggaran. | Menawarkan skema pay-as-you-go yang kompetitif, dengan keuntungan tambahan bagi organisasi yang sudah berinvestasi dalam |

| | | | |
|---|---|---|---|
| | kompleks untuk dikelola. | | ekosistem dan lisensi Microsoft. |
| Tingkat Gratis | Menyediakan kuota Tingkat Gratis yang sangat besar dan spesifik, contohnya 1 juta pesan per bulan untuk layanan AWS IoT Core. | Menawarkan Tingkat Gratis yang kompetitif dan mencakup berbagai layanan relevan, termasuk kuota eksekusi untuk Cloud Functions dan panggilan API bulanan. | Menarik bagi pengguna baru dengan menyediakan kuota Tingkat Gratis yang substansial, serta seringkali ditambah dengan kredit awal dalam jumlah signifikan. |
| Kemudahan Penggunaan | Menawarkan ekosistem paling komprehensif, yang berimbas pada kurva belajar yang lebih curam bagi pengguna baru akibat luasnya cakupan layanan. | Dikenal dengan antarmuka pengguna yang modern dan bersih, serta dokumentasi yang sangat berfokus pada pengalaman pengembang (*developer experience*). | Memberikan pengalaman yang sangat familiar bagi pengembang di ekosistem Microsoft berkat integrasinya yang erat dengan tools seperti Visual Studio. |

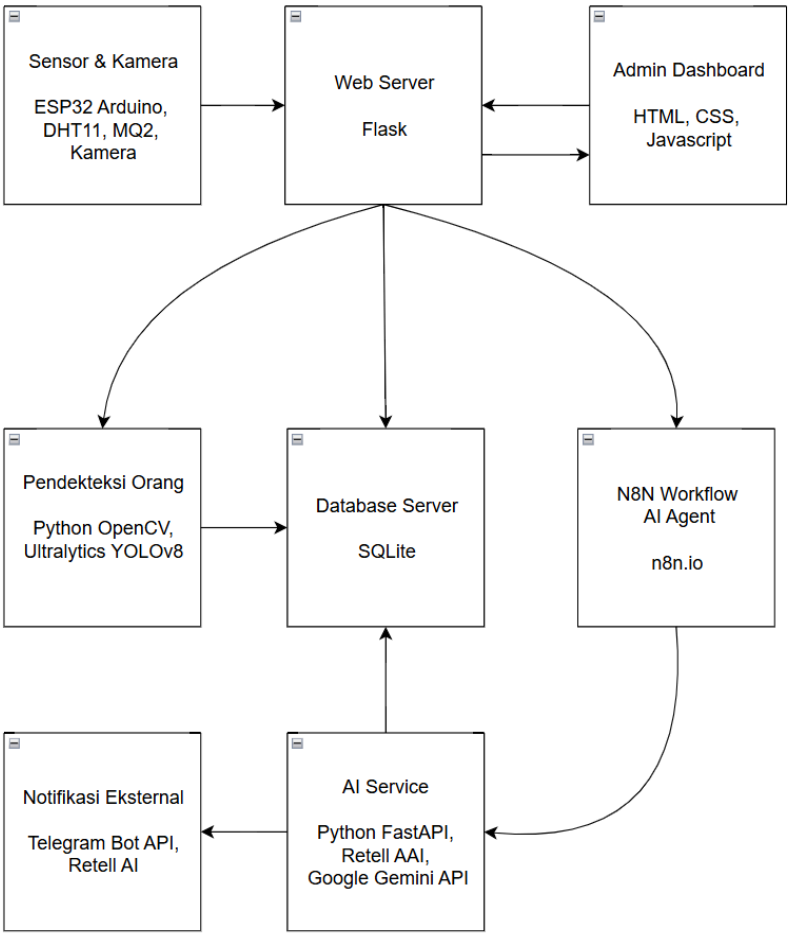
Meskipun ketiga platform cloud ini sangat mumpuni dan mampu menjalankan sistem serupa, kebutuhan spesifik dari proyek ini memberikan bobot lebih pada kriteria tertentu.

Kriteria yang menjadi penentu utama adalah **Integrasi Layanan AI**. Otak dari sistem ini adalah "Penasihat Evakuasi Cerdas" yang ditenagai oleh Google Gemini. Oleh karena itu, platform yang menawarkan integrasi paling mulus, efisien, dan berperforma tinggi dengan model ini akan memberikan keuntungan terbesar selama pengembangan dan operasi.

Dalam hal ini, Google Cloud Platform (GCP) memiliki keunggulan yang tidak dapat ditandingi. Proses untuk memanggil, mengelola, dan memantau Gemini API melalui layanan seperti Vertex AI di GCP secara inheren lebih sederhana dan dioptimalkan dibandingkan melakukannya dari platform cloud lain yang memerlukan lapisan abstraksi atau koneksi antar-cloud. Faktor pendukung lainnya adalah kemudahan penggunaan dan ekosistem yang kohesif dari GCP. Bagi sebuah tim pengembangan yang sedang membangun prototipe, antarmuka yang bersih dan dokumentasi yang jelas dari GCP dapat mempercepat proses pengembangan secara signifikan.

Berdasarkan analisis perbandingan di atas, apabila sistem ini akan di-deploy menggunakan platform cloud, maka **Google Cloud Platform (GCP)** adalah pilihan terbaik dan paling logis.

## ☆ Desain Diagram Blok dan Arsitektur Final Perangkat Lunak Sistem



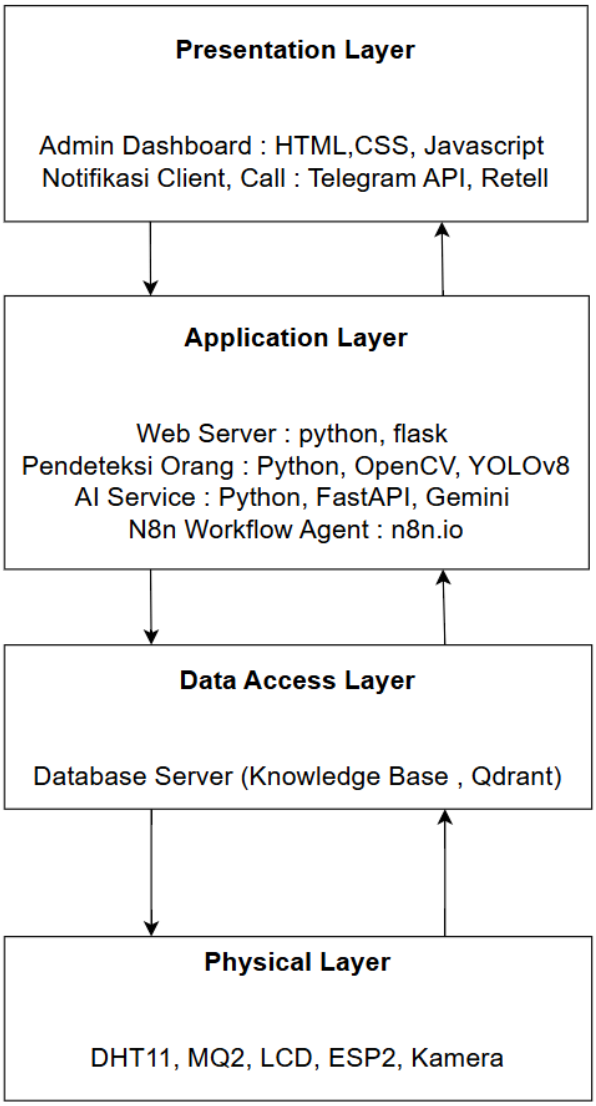
Arsitektur sistem ini diimplementasikan menggunakan serangkaian teknologi yang dipilih secara spesifik untuk mendukung fungsionalitas setiap komponen dalam kerangka kerja *microservices*. Infrastruktur inti pada sisi server dibangun di atas bahasa pemrograman Python. Komponen Web Server utama, yang berfungsi sebagai *API gateway* dan pemicu alur kerja, diimplementasikan menggunakan *micro-framework* Flask karena sifatnya yang ringan dan modular. Untuk lingkungan produksi, aplikasi Flask ini dirancang untuk dijalankan di atas server aplikasi WSGI yang andal seperti Gunicorn. Sebagai pusat orkestrasi, sistem memanfaatkan platform otomasi n8n.io, yang menerima pemicu melalui *webhook* dari Web Server untuk menjalankan alur kerja respons darurat yang telah ditentukan.

Untuk persistensi data, sistem menggunakan SQLite sebagai mesin database utamanya. Teknologi ini dipilih untuk tahap prototipe karena kesederhanaannya yang berbasis file, tidak memerlukan proses server terpisah, dan kemudahan integrasinya dengan Python. Secara konseptual, arsitektur data ini juga diperkaya dengan Qdrant Vector Store, yang digunakan dalam alur kerja n8n untuk memungkinkan pencarian informasi secara semantik oleh layanan AI.

Sistem ini memiliki dua layanan cerdas yang sangat terspesialisasi. Pertama, Subsistem Deteksi Objek yang merupakan skrip Python independen. Layanan ini memanfaatkan pustaka OpenCV untuk pemrosesan *stream* video dan model Ultralytics YOLOv8 untuk inferensi deteksi objek manusia secara *real-time*. Kedua, AI Service yang berfungsi sebagai otak percakapan. Layanan ini dibangun di atas *framework* FastAPI karena performanya yang tinggi dan dukungan untuk operasi asinkron, yang krusial untuk menangani koneksi *websocket*. Layanan ini mengelola logika percakapan dengan Google Gemini API dan berinteraksi dengan platform teleponi melalui Retell AI SDK.

Pada sisi perangkat keras dan klien, Node Sensor & Kamera terdiri dari mikrokontroler ESP32 yang menjalankan *firmware* berbasis C++/Arduino untuk akuisisi data dari sensor DHT11 dan MQ2, serta Webcam sebagai sumber input visual. Antarmuka pengguna utama, Admin Dashboard, dibangun sebagai aplikasi web klien menggunakan teknologi standar HTML5, CSS3, dan JavaScript, dengan pustaka Chart.js untuk visualisasi data grafik yang dinamis.

Terakhir, untuk penyampaian peringatan, sistem menggunakan dua kanal Notifikasi Eksternal utama: Retell AI Platform untuk melakukan panggilan suara AI yang interaktif, dan Telegram Bot API untuk mengirimkan notifikasi berbasis teks.



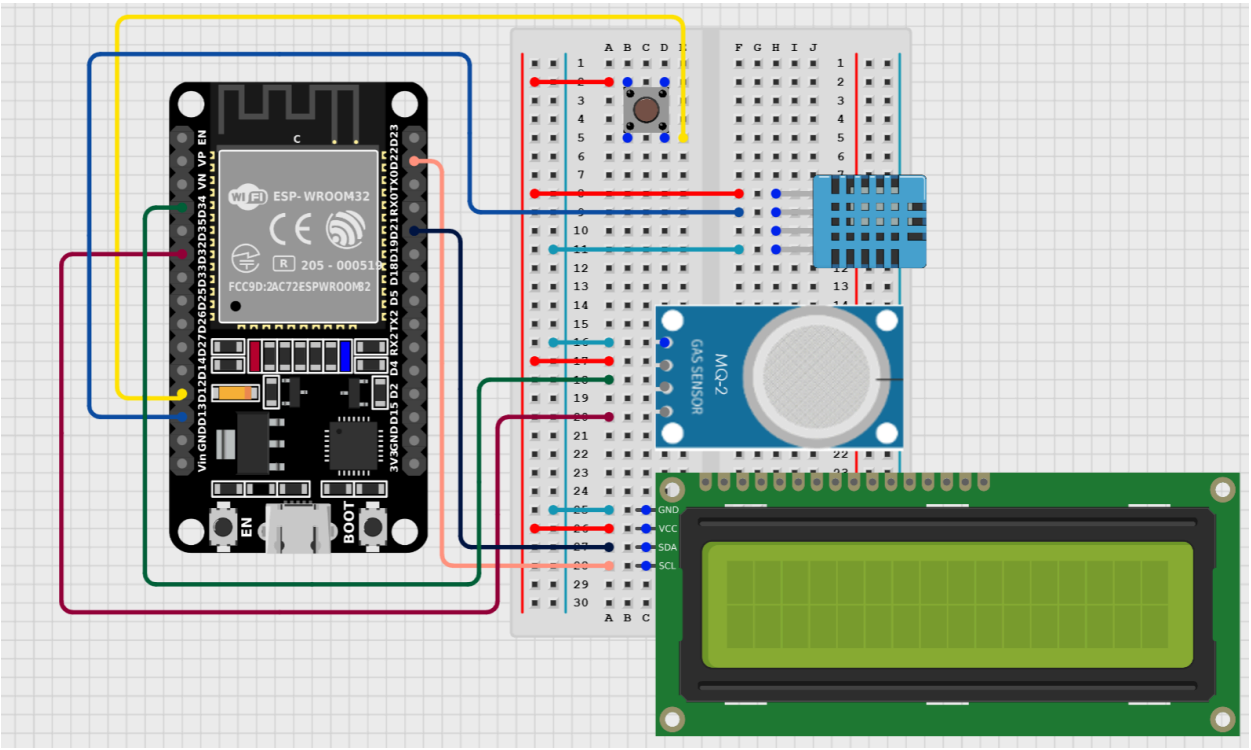

Arsitektur perangkat lunak sistem ini diimplementasikan dengan memetakan teknologi-teknologi yang dipilih ke dalam empat lapisan logis yang telah dirancang sebelumnya. Pemetaan ini memastikan bahwa setiap teknologi berfungsi sesuai dengan peran dan tanggung jawab lapisannya, menciptakan sebuah sistem yang terstruktur dan modular.

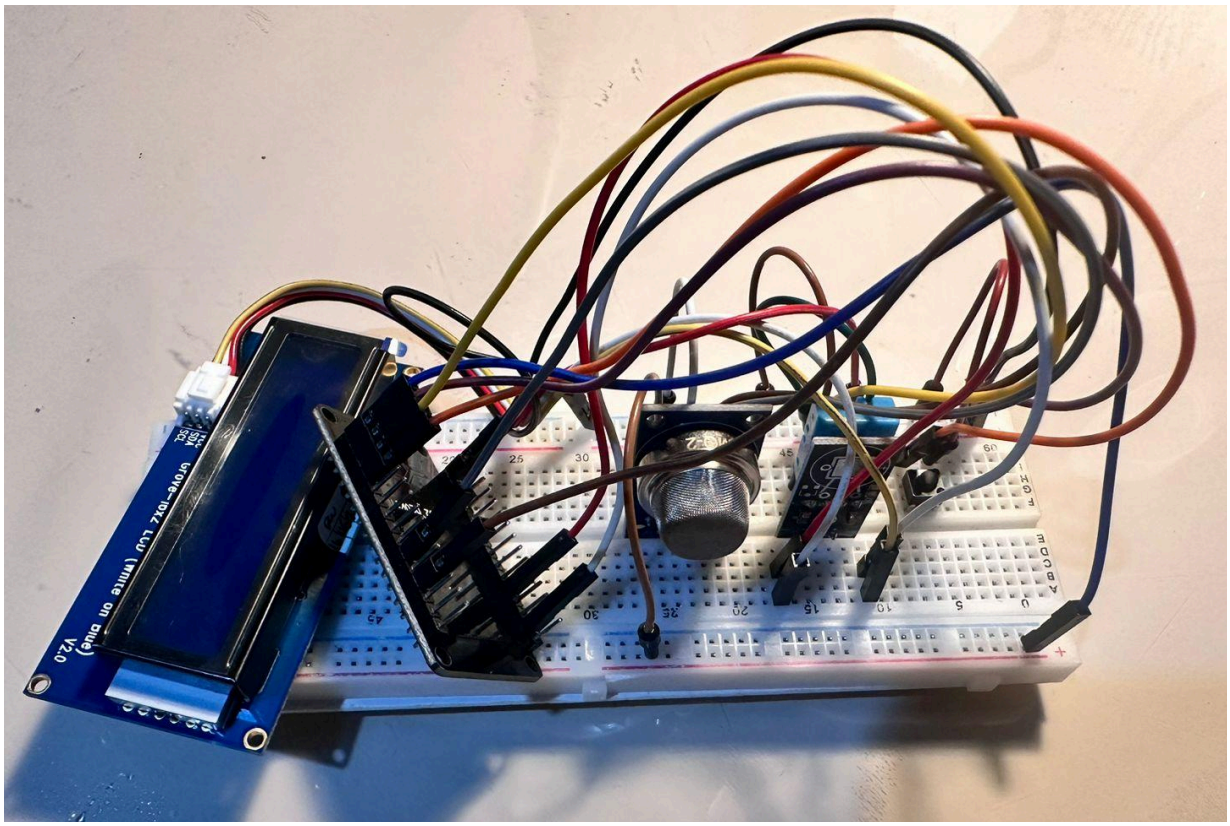
Fondasi dari arsitektur ini adalah Lapis Fisik/Perangkat (Physical/Device Layer), yang terdiri dari perangkat keras yang di-deploy di lapangan. Lapisan ini mencakup unit-unit Node Sensor yang dibangun di atas mikrokontroler ESP32 dengan *firmware* berbasis C++/Arduino, yang terintegrasi dengan sensor suhu DHT11 dan sensor asap MQ2. Selain itu, lapisan ini juga menyertakan Webcam standar sebagai perangkat akuisisi data visual untuk analisis lebih lanjut. Di atasnya, Lapis Akses Data & Persistensi (Data Access & Persistence Layer) diimplementasikan menggunakan SQLite sebagai mesin database utama. Pilihan ini ideal untuk prototipe karena kesederhanaannya yang berbasis file dalam menyimpan data insiden dan data hunian secara persisten. Secara konseptual, lapisan ini juga diperluas dengan adanya Knowledge Base dalam format PDF/Teks dan sebuah Qdrant Vector Store, yang keduanya menjadi sumber data krusial untuk proses *Retrieval-Augmented Generation* (RAG) oleh layanan AI.

Inti dari keseluruhan sistem berada pada Lapis Aplikasi/Layanan (Application/Service Layer), yang diimplementasikan sebagai komposisi dari beberapa *microservices* yang saling berkoordinasi. Web Server utama, yang berfungsi sebagai API *gateway*, dibangun menggunakan *micro-framework* Python Flask. Secara terpisah, Subsistem Deteksi Objek berjalan sebagai skrip Python independen yang memanfaatkan pustaka OpenCV dan model Ultralytics YOLOv8 untuk analisis video. Alur kerja darurat diorkestrasi oleh Platform Otomasi n8n.io, yang dipicu melalui *webhook*. Terakhir, AI Service yang canggih dibangun di atas *framework* Python FastAPI untuk menangani komunikasi asinkron, serta mengintegrasikan Google Gemini API untuk logika RAG dan Retell AI SDK untuk manajemen percakapan suara.

Pada puncak arsitektur terdapat Lapis Presentasi (Presentation Layer), yang menjadi antarmuka ke pengguna akhir. Untuk personel keamanan, lapisan ini menyediakan Admin Dashboard yang dibangun menggunakan teknologi web standar HTML5, CSS3, dan JavaScript, serta diperkaya dengan pustaka Chart.js untuk visualisasi data yang dinamis. Untuk penyampaian peringatan, lapisan ini menggunakan dua klien notifikasi utama: Retell AI Platform sebagai perantara untuk melakukan panggilan suara AI yang interaktif dan Telegram Bot API untuk mengirimkan notifikasi berbasis teks kepada pengguna.

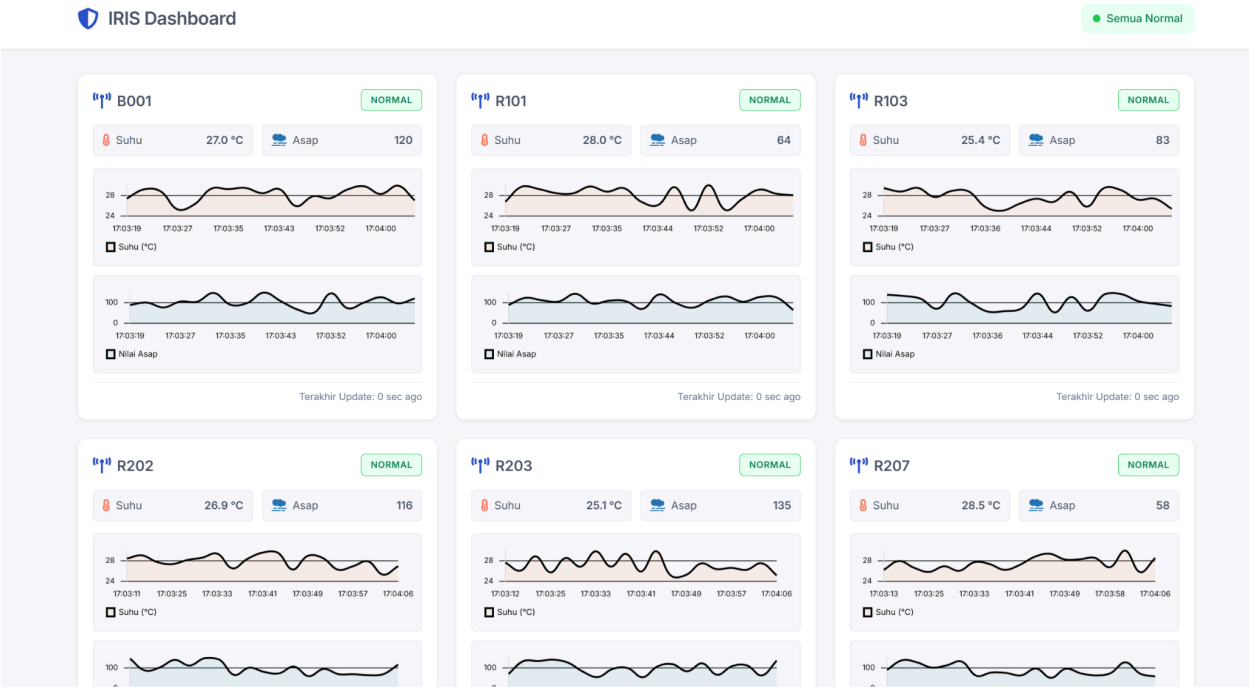
# Desain Perangkat Keras

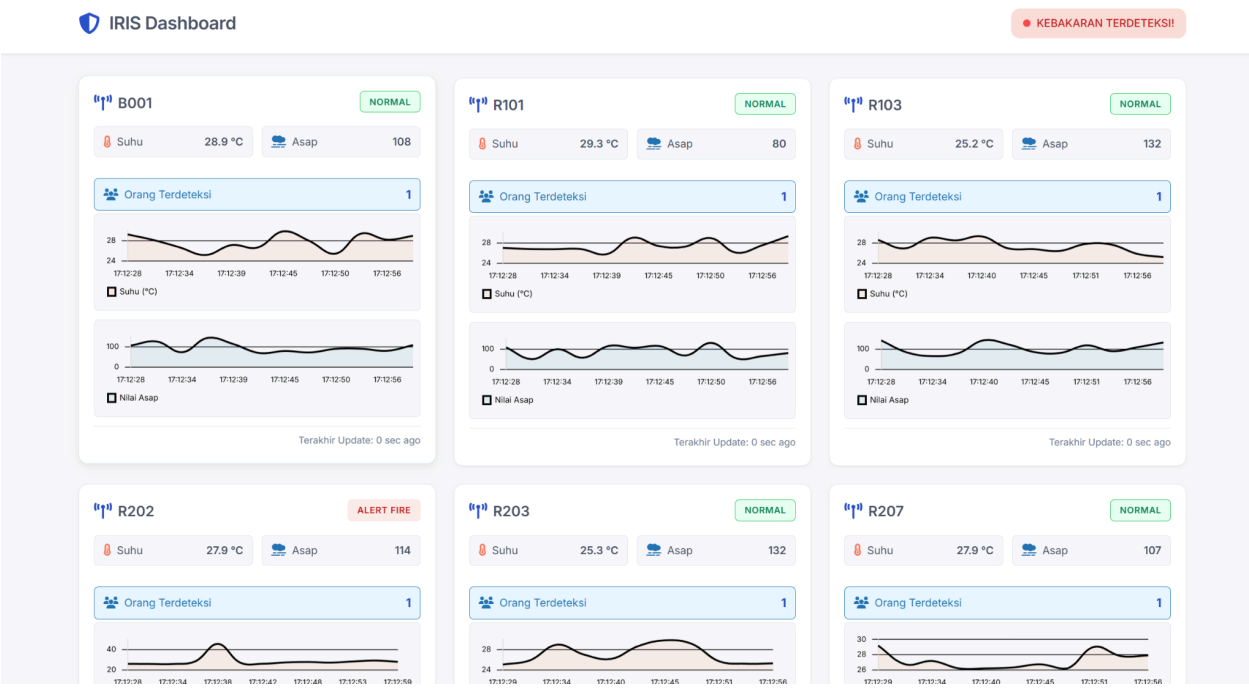


# Desain User Interface

## Desain Display dan Interaksi Pengguna






# Rencana Implementasi Perangkat Lunak

## Pemilihan Tools

Implementasi sistem peringatan kebakaran cerdas ini didukung oleh serangkaian *tools* dan teknologi yang dipilih secara spesifik untuk memastikan integrasi, fungsionalitas, dan keandalan sistem. Pemilihan *tools* ini dikategorikan berdasarkan fungsinya dalam arsitektur solusi, mulai dari pengembangan *backend* dan *frontend*, lingkungan kerja, hingga platform layanan eksternal.

### Pengembangan Aplikasi Inti (Bahasa, Framework, & Library)

Aplikasi sisi server (*backend*) dibangun menggunakan Python, didukung oleh *micro-framework* Flask yang ringan dan efisien untuk membangun *API endpoint* seperti /sensordata dan /get_live_data. Di sisi klien (*frontend*), struktur *dashboard* pemantauan dibangun menggunakan HTML, ditata secara visual dengan CSS, dan fungsionalitas dinamisnya diimplementasikan dengan JavaScript. Proses ini didukung oleh *library* spesifik: Requests di Python untuk

mengirim peringatan ke n8n, serta Chart.js untuk visualisasi data grafik dan Luxon untuk format waktu yang interaktif pada *dashboard*.

**Lingkungan Pengembangan dan Kontrol Versi**

Seluruh proses penulisan kode dikelola dalam Visual Studio Code (VS Code), sebuah *Integrated Development Environment* (IDE) yang dipilih karena fleksibilitas, ekosistem ekstensi yang kaya untuk berbagai bahasa, dan integrasi bawaan dengan sistem kontrol versi. Untuk melacak setiap perubahan pada kode sumber dan mengelola riwayat pengembangan, sistem kontrol versi Git digunakan, dengan repositori yang di-*hosting* pada platform seperti GitHub untuk memastikan kolaborasi dan keamanan kode.

**Platform dan Layanan Eksternal (PaaS/SaaS)**

Inti dari alur kerja cerdas ini di orkestrasi oleh n8n.io, sebuah platform otomasi yang menerima peringatan dari server, memproses logika klasifikasi darurat, dan mengintegrasikan layanan eksternal lainnya. Layanan-layanan ini mencakup Airtable yang berfungsi sebagai *database* untuk mengelola kontak karyawan, Telegram API sebagai kanal pengiriman notifikasi darurat, dan Gemini API (Google AI) yang berperan sebagai otak kecerdasan buatan untuk menganalisis situasi dan menghasilkan rute evakuasi yang dipersonalisasi.

**Perangkat Lunak Pendukung & Pengujian**

Proses pengembangan dan eksekusi sistem ini juga bergantung pada beberapa perangkat lunak pendukung. Interpreter Python diperlukan sebagai *runtime* untuk mengeksekusi kode Python pada app.py. Web Browser seperti Google Chrome dan Firefox berfungsi sebagai lingkungan eksekusi untuk *frontend* sekaligus sebagai alat untuk mengakses *dashboard* n8n. Selain itu, pada tahap pengembangan, *tools* pengujian API seperti Postman dan cURL digunakan untuk memvalidasi fungsionalitas *endpoint* /sensordata dengan mensimulasikan pengiriman data sensor sebelum integrasi penuh dengan perangkat keras di lapangan.

# Lampiran

https://drive.google.com/drive/folders/1gHeuxWhP8tCal837KLrQeOVurMDBRz05