BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

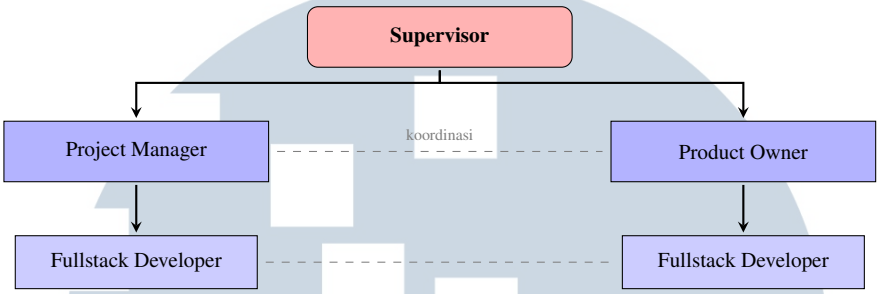
Selama masa magang di PT. Moonlay Technologies, posisi yang dijalankan adalah sebagai *Fullstack Developer Intern* dalam tim pengembangan proyek CORA *(AI Receptionist)*. Bimbingan langsung dilakukan oleh *Software Development Manager*, dengan pelaksanaan pekerjaan secara kolaboratif bersama tim yang terdiri dari beberapa peran utama, yaitu *Fullstack Developer*, *Product Owner*, dan *Project Manager*.

Struktur organisasi proyek dan pembagian tanggung jawab tiap jabatan dapat dilihat pada **Tabel 3.1.1** berikut.

Nama Jabatan	Tanggung Jawab Utama
Supervisor Head of Technology	Memberikan arahan strategis dan supervisi terhadap pengembangan teknologi, serta memastikan seluruh proses teknis selaras dengan tujuan perusahaan.
Project Manager	Mengelola timeline proyek, alokasi sumber daya, dan menjamin proyek berjalan sesuai dengan rencana dan target yang ditentukan.
Product Owner	Menyusun dan memprioritaskan backlog produk, serta menjadi penghubung antara tim teknis dan pemangku kepentingan (stakeholder).
Fullstack Developer	Mengembangkan dan mengintegrasikan bagian frontend dan backend aplikasi, serta memastikan komunikasi antar komponen berjalan optimal.

Tabel 3.1. Struktur Tim Proyek CORA

Struktur Tim CORA



Gambar 3.1. Struktur Tim CORA di PT. Moonlay Technologies

Koordinasi tim dilakukan secara Agile melalui pertemuan rutin seperti *daily stand-up*, *refinement*, dan *sprint retrospective*. Platform komunikasi yang digunakan mencakup Microsoft Teams untuk diskusi harian, dan GitHub untuk kolaborasi kode.

3.2 Tugas yang Dilakukan

Selama masa magang di PT. Moonlay Technologies, terdapat berbagai tanggung jawab yang mencakup aspek teknis maupun non-teknis. Pada sisi teknis, kegiatan berfokus pada pengembangan proyek CORA (*AI Receptionist*) yang melibatkan implementasi *frontend, backend*, serta integrasi layanan pihak ketiga. Di sisi non-teknis, aktivitas lebih banyak melibatkan kerja tim berbasis *Agile*, termasuk perencanaan sprint, koordinasi dengan tim terkait, serta partisipasi dalam evaluasi dan pelaporan kemajuan. Penjabaran berikut merinci tugas-tugas yang dijalankan selama masa magang.

3.2.1 Tugas Teknis

Selama masa magang, berkontribusi dalam pengembangan proyek CORA melalui pembagian backlog berdasarkan sprint. Setiap sprint dilaksanakan secara iteratif dan terukur dengan tujuan utama meningkatkan fungsionalitas serta stabilitas sistem. Berikut uraian detail pelaksanaan tugas teknis berdasarkan backlog yang telah ditetapkan.

Backlog 1 – Sprint 1 (11 Februari – 25 Februari 2025)

Judul Backlog	Pengembangan Fitur Dasar Sistem
User Story	Sebagai pengguna, saya ingin mengajukan pertanyaan secara suara dan mendapatkan respons agar dapat berinteraksi dengan sistem CORA secara alami.

- Mengembangkan fitur input suara menggunakan *Web Audio API*.
- Menghubungkan input suara ke *backend* untuk proses *Speech-to-Text (STT)* dan respons.
- Menampilkan indikator status proses (*loading*/berjalan).

Output:

- Fitur input suara yang terhubung ke *backend* dan dapat digunakan.
- Respons muncul berdasarkan hasil transkripsi STT.

Backlog 2 – Sprint 2 (25 Februari – 07 Maret 2025)

Judul Backlog	Respons Suara (TTS) dan Pemahaman Konteks
User Story	Sebagai pengguna, saya ingin jawaban dikembalikan dalam bentuk suara agar interaksi terasa lebih manusiawi.

- Integrasi layanan *Text-to-Speech* (Azure TTS).
- Implementasi pemutaran audio secara otomatis.
- Integrasi model OpenAI untuk menghasilkan respons.

Output:

- Sistem dapat menjawab dalam bentuk suara.
- Respons yang diberikan mempertimbangkan konteks input.

Backlog 3 – Sprint 3 (07 Maret – 17 Maret 2025)

Judul Backlog	Perbaikan Fitur Greeting
User Story	Sebagai pengguna, saya ingin CORA menyapa hanya saat tidak ada interaksi lain agar tidak mengganggu sesi berjalan.

- Perbaikan logika pendeteksian wajah (*cooldown* dan *session check*).
- Sinkronisasi antara *event greeting* dan sesi interaksi.

Output:

- CORA menyapa hanya satu kali per pengguna dalam satu sesi.
- Tidak terjadi gangguan selama sesi aktif berlangsung.

Backlog 4 – Sprint 4 (24 Maret – 07 April 2025)

Judul Backlog	Fitur Memori Konteks
User Story	Sebagai pengguna, saya ingin CORA mengingat percakapan sebelumnya agar respons lebih relevan.

- Implementasi `LangChain` menggunakan `ConversationSummaryBufferMemory`.
- Menyimpan dan memanfaatkan konteks percakapan sebelumnya.
- Penambahan tombol “Akhiri Percakapan” untuk menghapus memori.

Output:

- Sistem dapat mengingat dan merespons berdasarkan riwayat percakapan.
- Pengguna dapat mengatur ulang memori secara manual.

Backlog 5 – Sprint 5 (08 April – 21 April 2025)

Judul Backlog	Mode Respons General dan Detail
User Story	Sebagai pengguna, saya ingin memilih antara jawaban ringkas atau detail sesuai kebutuhan saya.

- Menambahkan *UI toggle* untuk mode *General* dan *Detail*.
- Pengaturan prompt berbeda untuk setiap mode.
- *Caching* untuk respons yang bersifat umum.

Output:

- Pengguna dapat memilih mode jawaban yang diinginkan.
- Respons tampil sesuai dengan mode yang dipilih.

Backlog 6 – Sprint 6 (21 April – 02 Mei 2025)

Judul Backlog	Mode Bahasa, Ulang Jawaban, dan Volume
User Story	Sebagai pengguna, saya ingin mengganti bahasa, mengulang jawaban, dan mengatur volume agar lebih fleksibel.

- Fitur ganti bahasa (Indonesia/Inggris).
- Tombol pengulangan audio respons.
- *Slider* pengatur volume suara.

Output:

- Fitur bahasa, repeat, dan pengaturan volume dapat digunakan.
- Preferensi pengguna tersimpan secara lokal.

Backlog 7 – Sprint 7 (05 Mei – 16 Mei 2025)

Judul Backlog	Halaman Admin untuk Manajemen File RAG
User Story	Sebagai admin, saya ingin mengunggah dokumen agar sistem bisa menjawab berdasarkan isi dokumen tersebut.

- Membuat halaman admin untuk *upload* dan *delete* file.
- Integrasi dengan Supabase dan *Vector Database*.
- Proses *indexing* dan penyimpanan *embedding*.

Output:

- Admin dapat mengelola file yang menjadi sumber jawaban sistem.
- CORA mampu menjawab berdasarkan isi dokumen.

Backlog 8 – Sprint 8 (19 Mei – 02 Juni 2025)

Judul Backlog	Pembaruan UI Mikrofon dan Kontrol Respons
User Story	Sebagai pengguna, saya ingin bisa menghentikan suara sistem agar lebih nyaman.

- *Redesign* tampilan tombol mikrofon.
- Menambahkan tombol untuk menghentikan suara saat sedang berjalan.

Output:

- UI lebih interaktif dan informatif.
- Pengguna dapat menghentikan pemutaran suara sesuai kebutuhan.

Backlog 9 – Sprint 9 (02 Juni – 15 Juni 2025)

Judul Backlog	Ketahanan Sistem dan Penanganan Gangguan
User Story	Sebagai pengguna, saya ingin sistem tetap stabil meskipun jaringan buruk atau proses tertunda.

- Menambahkan skenario *fallback* dan notifikasi error.
- Implementasi *retry mechanism* dan *delay handling*.

Output:

- CORA dapat tetap digunakan meskipun terjadi gangguan teknis.
- Sistem memberikan *feedback* yang sesuai jika terjadi error.

3.2.2 Tugas Non-Teknis

Selain tanggung jawab teknis, keterlibatan juga dilakukan dalam sejumlah kegiatan non-teknis yang mendukung kelancaran pengembangan proyek secara tim. Kegiatan ini dilaksanakan secara rutin berdasarkan metode kerja *Agile* yang diterapkan oleh perusahaan. Adapun kegiatan non-teknis yang dimaksud adalah sebagai berikut:

- **Refinement Meeting**: Sesi diskusi untuk membahas rincian backlog sebelum dimulai sprint berikutnya. Dalam pertemuan ini, dilakukan klarifikasi cakupan pekerjaan, estimasi tingkat kesulitan, serta pembagian tugas secara merata kepada seluruh anggota tim.
- **Daily Stand-Up**: Pertemuan harian berdurasi singkat (±15 menit) di mana setiap anggota tim melaporkan progres, rencana pekerjaan hari ini, dan kendala yang dihadapi. Tujuan dari kegiatan ini adalah menjaga transparansi kerja dan memastikan koordinasi berjalan secara efektif.
- **Demo Proyek**: Presentasi rutin di akhir sprint untuk menyampaikan progres fitur yang telah dikerjakan kepada seluruh anggota tim dan supervisor. Kegiatan ini bertujuan untuk memperlihatkan hasil pengembangan secara nyata dan memperoleh umpan balik secara langsung.

- **Sprint Retrospective**: Evaluasi yang dilakukan di akhir sprint untuk merefleksikan proses kerja yang telah berjalan. Sesi ini digunakan untuk mengidentifikasi hal-hal yang berjalan baik, hambatan yang muncul, serta menyusun langkah-langkah perbaikan untuk sprint selanjutnya.

3.3 Tools dan Teknologi yang Digunakan

Berbagai teknologi digunakan untuk mendukung proses pengembangan sistem CORA selama masa magang. Tools dan teknologi ini dapat dikelompokkan berdasarkan fungsinya sebagai berikut:

3.3.1 Frontend Development

Pada sisi frontend, pengembangan antarmuka pengguna dilakukan menggunakan **React.js**, sebuah library JavaScript yang populer untuk membangun UI berbasis komponen. Bahasa pemrograman yang digunakan adalah **TypeScript**, yang memberikan dukungan sistem tipe statis dan meningkatkan keandalan dalam penulisan kode. Seluruh pengembangan frontend dilakukan menggunakan **Visual Studio Code** sebagai editor utama karena fleksibilitas dan dukungan ekstensinya yang luas.

Tools	**Keterangan**
Visual Studio Code	Editor utama untuk menulis kode *frontend* dan *backend*.
TypeScript	Bahasa pemrograman untuk frontend dengan sistem tipe statis.
React.js	Library JavaScript untuk membangun antarmuka pengguna (UI).

Tabel 3.2. Tools untuk Frontend Development

3.3.2 Backend Development

Untuk sisi backend, pengembangan dilakukan menggunakan **Flask**, sebuah microframework berbasis Python yang ringan dan fleksibel. Flask digunakan untuk membangun API yang menghubungkan frontend dengan berbagai layanan eksternal. Proses pengembangan backend dan eksekusi perintah dilakukan melalui **Command Prompt (CMD)** sebagai terminal bawaan Windows.

Tools	Keterangan
Flask	Framework Python untuk membuat backend dan REST API.
Command Prompt (CMD)	Terminal untuk menjalankan server lokal dan perintah CLI.

Tabel 3.3. Tools untuk Backend Development

3.3.3 Database dan Data Management

Manajemen file dan data pada sistem ini menggunakan **Supabase**, yaitu layanan backend-as-a-service yang berfungsi untuk menyimpan file dokumen yang diunggah oleh admin. Selain itu, digunakan pula **Vector Database (Vector DB)** untuk melakukan pencarian jawaban yang relevan dari isi dokumen saat pengguna mengajukan pertanyaan, terutama pada implementasi fitur Retrieval-Augmented Generation (RAG).

Tools	Keterangan
Supabase	*Backend-as-a-service* yang digunakan untuk menyimpan file unggahan admin.
Vector DB	Digunakan untuk pencarian informasi yang relevan dari dokumen.

Tabel 3.4. Tools untuk Database dan Data Management

3.3.4 AI, *Machine Learning*, dan Layanan API

Dalam pengolahan bahasa alami dan interaksi berbasis suara, digunakan beberapa layanan berbasis AI. OpenAI dimanfaatkan untuk menghasilkan jawaban berbasis model ChatGPT. Whisper ASR digunakan untuk mengubah masukan suara pengguna menjadi teks (Speech-to-Text), sedangkan Microsoft Azure *Text-to-Speech* (TTS) digunakan untuk mengubah teks balasan menjadi suara. Untuk mengelola alur kerja RAG secara modular, digunakan pula LangChain sebagai *library* yang memfasilitasi orkestrasi antara dokumen, *prompt*, dan output AI.

Tools	Keterangan
OpenAI	API untuk pemrosesan bahasa alami menggunakan ChatGPT.
Whisper ASR	*Model Automatic Speech Recognition* dari OpenAI untuk STT.
Microsoft Azure TTS	Layanan *Text-to-Speech* untuk menghasilkan suara balasan.
LangChain	Library untuk mengatur alur logika jawaban berbasis dokumen (RAG).

Tabel 3.5. Tools untuk AI, Machine Learning, dan Layanan API

3.3.5 Manajemen Proyek dan Kolaborasi

Dalam proses pengembangan yang menggunakan metode Agile, tim memanfaatkan Jira sebagai alat bantu manajemen sprint dan pembagian tugas. Untuk mendokumentasikan dan mereview pekerjaan, digunakan GitHub sebagai platform version control. Selain itu, Parabol digunakan untuk pelaksanaan retrospektif tim secara kolaboratif setelah setiap sprint berakhir.

Tools	Keterangan
GitHub	Platform version control untuk kolaborasi tim dan manajemen repositori.
Jira	Alat manajemen tugas dan sprint dengan metode Agile.
Parabol	Alat kolaboratif untuk retrospektif sprint dan evaluasi tim.

Tabel 3.6. Tools untuk Manajemen Proyek dan Kolaborasi

3.4 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang di PT. Moonlay Technologies dimulai dengan masa orientasi dan pengenalan lingkungan kerja. Pada minggu pertama, peserta magang dibimbing untuk memahami metode kerja yang digunakan di perusahaan, yaitu metode Agile. Minggu kedua difokuskan pada pengenalan tools pendukung seperti Jira untuk manajemen tugas dan kolaborasi tim. Memasuki minggu ketiga, peserta mulai dikenalkan dengan proyek utama yang akan dikembangkan, yaitu CORA (AI Receptionist). Di minggu ini juga dilakukan persiapan teknis untuk proyek seperti instalasi kebutuhan sistem, setup repository GitHub,

serta pembelajaran bahasa pemrograman TypeScript yang digunakan dalam pengembangan frontend proyek.

Minggu	Tanggal	Aktivitas
Minggu ke-1	22 Jan – 24 Jan	Pengenalan lingkungan kerja, peraturan magang, serta metode kerja Agile (Scrum, Sprint, dan Stand-up Meeting).
Minggu ke-2	27 Jan – 31 Jan	Pengenalan tools manajemen proyek seperti Jira, serta alur komunikasi dan kolaborasi tim.
Minggu ke-3	03 Feb – 07 Feb	Pengenalan proyek CORA, pemahaman alur dan fitur sistem, serta persiapan teknis berupa instalasi tools dan library, setup GitHub, serta pembelajaran dasar TypeScript.

Tabel 3.7. Aktivitas Mingguan pada Tahap Orientasi dan Persiapan Proyek

Kegiatan magang dimulai dengan observasi dan pemahaman struktur proyek serta arsitektur sistem CORA, dilanjutkan dengan pengembangan tampilan antarmuka menggunakan React.js dan TypeScript, serta integrasi backend python berbasis Flask. Selain itu, dilakukan pula integrasi dengan layanan pihak ketiga seperti OpenAI API untuk pemrosesan bahasa alami dan *Azure Speech Service* untuk fitur suara. Pelaporan progres dilakukan setiap hari melalui pertemuan *daily stand-up*, sementara evaluasi mingguan disampaikan dalam sesi demo kepada supervisor dan tim.

3.4.1 Uraian Pekerjaan Tiap Sprint

Sprint	Tanggal	Pekerjaan yang Dilakukan
1	11 Feb – 25 Feb	Melakukan pengembangan fitur input suara, integrasi pertanyaan pengguna ke *backend*, serta implementasi indikator proses respons.
2	25 Feb – 07 Mar	Mengimplementasikan fitur *Text-to-Speech* (*TTS*) agar CORA dapat memberikan respons dalam bentuk suara.
3	07 Mar – 17 Mar	Melakukan perbaikan pada fitur *greeting*, termasuk logika pendeteksian wajah agar tidak mengganggu sesi interaksi.
4	24 Mar – 07 Apr	Menambahkan fitur memori konteks agar interaksi antara pengguna dan CORA bersifat berkelanjutan dengan menggunakan langchain.
5	08 Apr – 21 Apr	Mengembangkan fitur mode respons dengan pilihan "General" dan "Detail" sesuai kebutuhan informasi pengguna.
6	21 Apr – 02 Mei	Menambahkan pengaturan mode bahasa (Indonesia/Inggris) dan fitur pengulangan jawaban.
7	05 Mei – 16 Mei	Mengembangkan halaman admin untuk mengunggah dan menghapus *file*, serta memproses pertanyaan berdasarkan isi *file* tersebut.
8	19 Mei – 02 Jun	Melakukan pembaruan antarmuka pada ikon mikrofon dan menambahkan fitur untuk menghentikan respons suara.
9	02 Jun – 15 Jun	Merancang skenario penanganan gangguan seperti koneksi tidak stabil, keterlambatan proses, dan kegagalan pemanggilan *API*.

Tabel 3.8. Pekerjaan yang Dilakukan Tiap Sprint Selama Pelaksanaan Kerja Magang

3.4.2 Deskripsi Setiap Sprint

Sprint 1: Pengembangan Fitur Dasar Sistem

Pada Sprint 1, fokus utama pengembangan sistem CORA adalah membangun kemampuan dasar interaksi antara pengguna dan sistem melalui input suara. Pada tahap ini, CORA telah mampu menerima input suara dari pengguna untuk mengajukan pertanyaan, kemudian memproses dan memberikan respons sesuai dengan kebutuhan yang diajukan. Selain itu, sistem dirancang untuk dapat melakukan koreksi otomatis terhadap pertanyaan yang kurang jelas atau keliru,

sehingga dapat meningkatkan akurasi pemahaman terhadap maksud pengguna. Sebagai penanda proses, ditambahkan indikator visual yang menampilkan status pemrosesan jawaban, sehingga pengguna dapat mengetahui bahwa sistem sedang bekerja dan merespons pertanyaan yang diajukan. Implementasi ini menjadi fondasi awal bagi pengembangan interaksi berbasis suara yang lebih kompleks pada sprint-sprint selanjutnya.

Proses

Pada Sprint 1, sistem CORA dirancang untuk memungkinkan pengguna berinteraksi menggunakan suara, melibatkan beberapa proses utama yang saling terintegrasi.

1. Pengguna menekan tombol mikrofon pada antarmuka untuk memulai percakapan.

```
    const startListening = () => {
      setIsListening(true);
      // mulai rekam audio dengan MediaRecorder
    };

```

Kode 3.1: Fungsi untuk mulai merekam audio di frontend

2. Suara pengguna direkam melalui browser dan dikirimkan ke *backend*.

```
    const stopListening = () => {
      mediaRecorder.stop();
      // setelah stop, kirim audio blob ke backend
    };

```

Kode 3.2: Mengirim audio ke backend setelah rekaman selesai

3. *Backend* memproses audio menggunakan fitur *speech-to-text* (STT) untuk mengubahnya menjadi teks pertanyaan.

```
    @app.route("/speech-to-text", methods=["POST"])
    def speech_to_text():
        audio_file = request.files["audio"]
        transcript = openai.audio.transcriptions.create(
            model="whisper-1", file=audio_file
```

```
        )
        return jsonify({"text": transcript.text})

```

Kode 3.3: Endpoint Flask untuk proses speech-to-text

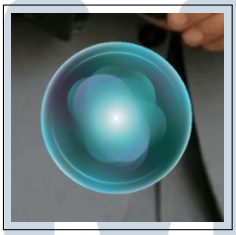
4. Sementara *backend* memproses jawaban, *frontend* menampilkan *bubble chat* dengan status *loading*.... Setelah jawaban diterima, transkrip jawaban ditampilkan dalam *bubble* tersebut.

```
    setLoading(true);
    const response = await fetch("/chat", ...);
    const data = await response.json();
    setLoading(false);
    setChatBubble(data.response);

```

Kode 3.4: Menampilkan status loading dan hasil jawaban

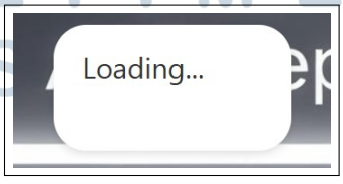
Tampilan *User Interface* (UI)

Berikut ini adalah *screenshot* dari antarmuka pengguna pada sistem CORA saat pengguna menekan tombol mikrofon untuk mulai merekam suara.



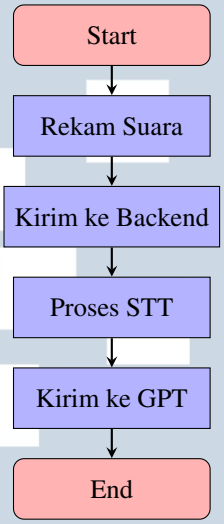
Gambar 3.2. Tampilan tombol mikrofon

Selain itu, sistem juga menampilkan indikator loading saat backend sedang memproses suara pengguna.



Gambar 3.3. Indikator loading saat pemrosesan suara

Flowchart



Gambar 3.4. Flowchart Proses Tanya-Jawab Suara pada Sistem CORA

Kendala

Beberapa kendala yang ditemukan selama Sprint 1 adalah sebagai berikut:

- **Delay saat proses *Speech-to-Text* (STT)**
  Proses konversi suara ke teks menggunakan API seperti OpenAI Whisper memerlukan waktu beberapa detik untuk menyelesaikan transkripsi. Jika tidak terdapat indikator proses yang jelas, pengguna dapat mengira sistem tidak merespons.
- **Hasil STT tidak akurat**
  Faktor seperti kebisingan latar belakang, aksen, atau pengucapan pengguna dapat memengaruhi akurasi hasil transkripsi. Akibatnya, pertanyaan yang dikirim ke sistem tidak sesuai dengan maksud pengguna, sehingga jawaban yang diberikan juga tidak relevan.

Solusi

Beberapa solusi yang diterapkan atau direncanakan untuk mengatasi kendala pada Sprint 1 adalah sebagai berikut:

- **Menambahkan indikator loading saat proses STT**
  Untuk meningkatkan pengalaman pengguna, ditambahkan indikator visual

berupa teks *"loading..."* atau animasi saat sistem sedang melakukan transkripsi maupun memproses jawaban.

- **Pengguna berbicara lebih jelas**
  Pengguna diarahkan untuk berbicara dengan intonasi yang jelas dan artikulasi yang baik agar hasil transkripsi oleh sistem STT lebih akurat.

Sprint 2: Pengembangan Respons Suara dan Pemahaman Pertanyaan

Pada Sprint 2, pengembangan sistem difokuskan pada kemampuan CORA dalam memberikan respons dalam bentuk suara, serta meningkatkan pemahaman sistem terhadap maksud pertanyaan pengguna. Sistem dirancang untuk tidak hanya mengenali teks dari input suara, tetapi juga memahami konteksnya menggunakan pemrosesan bahasa alami (*Natural Language Processing*). Dengan demikian, CORA dapat memberikan jawaban yang relevan dan sesuai dengan kebutuhan pengguna. Fitur *Text-to-Speech* (TTS) juga diintegrasikan untuk menyampaikan jawaban secara audio, sehingga interaksi terasa lebih alami dan interaktif. Tahap ini merupakan langkah penting dalam membangun pengalaman pengguna yang lebih menyeluruh dan responsif melalui medium suara.

Proses

Implementasi fitur suara pada CORA melibatkan beberapa proses utama yang saling terintegrasi:

1. *Audio Capture*

```
const startRecording = async () => {
  const stream = await navigator.mediaDevices.getUserMedia({ audio
    : true });
  mediaRecorder.current = new MediaRecorder(stream);

  mediaRecorder.current.ondataavailable = (e) => {
    if (e.data.size > 0) {
      audioChunks.current.push(e.data);
    }
  };

  mediaRecorder.current.start();
```

```
};
```

Kode 3.5: Implementasi Perekaman Audio

Kode di atas menunjukkan proses implementasi perekaman audio menggunakan Web API `MediaRecorder`. Fungsi `startRecording` diawali dengan permintaan akses mikrofon dari pengguna melalui method `navigator.mediaDevices.getUserMedia`, yang akan mengembalikan stream audio jika diizinkan. Stream tersebut kemudian digunakan untuk membuat objek `MediaRecorder` yang merekam data audio. Potongan data audio (chunk) yang tersedia akan ditangani oleh event handler `ondataavailable`, dan setiap data yang direkam akan disimpan ke dalam array `audioChunks`. Proses perekaman dimulai dengan pemanggilan method `start()` pada objek `MediaRecorder`. Potongan-potongan data audio ini nantinya dapat digabungkan dan dikirim ke server untuk diproses lebih lanjut dalam tahap transkripsi atau analisis suara.

2. Pengiriman ke *Backend*

```
const stopAndSend = async () => {
  mediaRecorder.current.stop();
  const blob = new Blob(audioChunks.current, {type: 'audio/webm'})
    ;

  const formData = new FormData();
  formData.append('audio', blob, 'recording.webm');

  const response = await fetch('/api/stt', {
    method: 'POST',
    body: formData
  });
  return response.json();
};
```

Kode 3.6: Pengiriman Audio ke Backend

Kode ini merupakan fungsi pada sisi *frontend* untuk menghentikan proses perekaman dan mengirimkan berkas audio ke *backend*. Setelah `mediaRecorder.current.stop()` dipanggil, potongan audio yang telah direkam digabung menjadi satu berkas `Blob` dengan tipe `audio/webm`. Berkas ini kemudian dikemas menggunakan objek `FormData` dan dikirim melalui permintaan `HTTP POST` ke *endpoint* `/api/stt`. Setelah permintaan berhasil diproses oleh server,

hasil transkripsi dalam bentuk *JSON* akan dikembalikan ke *frontend* sebagai respons. Proses ini merupakan bagian dari alur konversi suara menjadi teks sebelum dianalisis oleh sistem.

3. Speech-to-Text Processing

```
@app.route("/speech-to-text", methods=["POST"])
def speech_to_text():
        audio_file = request.files["audio"]
        audio_bytes = io.BytesIO(audio_file.read())
        audio_bytes.name = "audio.webm"

        transcript = openai.audio.transcriptions.create(
            model="whisper-1",
            file=audio_bytes
        )
    return jsonify({"text": transcript.text})
    except Exception as e:
        log_to_supabase("speech-to-text", "failed", str(e), {"
    error_type": type(e).__name__})
        print(f"Error in speech-to-text: {e}")
        return jsonify({"error": str(e)}), 500
```

Kode 3.7: STT Service

Kode di atas merupakan implementasi layanan *Speech-to-Text* (*STT*) menggunakan *framework* `Flask` pada sisi *backend*. Fungsi `speech_to_text` dipanggil ketika *endpoint* `/speech-to-text` menerima permintaan HTTP dengan metode `POST`. Berkas audio yang dikirim dari *frontend* diambil melalui `request.files["audio"]` dan dibaca dalam format `BytesIO` untuk memudahkan proses selanjutnya. Berkas ini kemudian dikirim ke layanan *Whisper API* dari OpenAI untuk dilakukan transkripsi suara menjadi teks. Hasil transkripsi akan dikembalikan dalam bentuk respons *JSON* kepada *frontend*. Dalam kasus terjadi kesalahan, sistem akan mencatat *log* kesalahan ke *Supabase* untuk keperluan *debugging* dan memberikan respons *error* dengan status `500` sebagai tanda kegagalan.

4. Text-to-Speech Generation

```
def generate_speech(text):
    speech_config = speechsdk.SpeechConfig(
        subscription=os.getenv('AZURE_TTS_KEY'),
        region="southeastasia"
    )
    speech_config.speech_synthesis_voice_name = "en-US-JennyNeural
    "

    synthesizer = speechsdk.SpeechSynthesizer(
        speech_config=speech_config,
        audio_config=None
    )

    result = synthesizer.speak_text_async(text).get()
    return result.audio_data
```

Kode 3.8: TTS Service

Kode ini mendefinisikan fungsi `generate_speech` yang digunakan untuk menghasilkan suara dari teks menggunakan layanan *Text-to-Speech* (TTS) dari Azure. Pertama, konfigurasi suara dibuat melalui objek `SpeechConfig`, yang memuat kunci API dan wilayah (*region*) layanan. Kemudian, suara sintetis yang digunakan ditentukan melalui `speech_synthesis_voice_name`, dalam hal ini menggunakan suara neural `en-US-JennyNeural`. Selanjutnya, objek `SpeechSynthesizer` dibuat untuk menjalankan proses sintesis suara secara asinkron melalui metode `speak_text_async`. Hasil akhirnya adalah data audio yang dapat digunakan lebih lanjut, seperti untuk diputar atau dikirimkan ke pengguna melalui *streaming*.

5. Audio Streaming

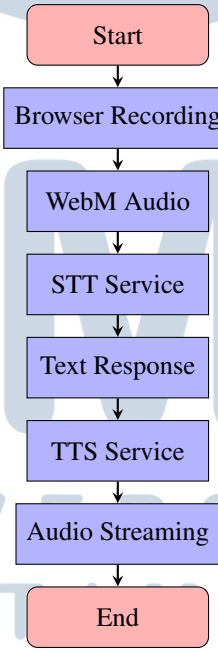
```
@app.route('/api/stream')
def audio_stream():
    def generate():
        while True:
            text = get_next_response()
            audio = generate_speech(text)

            yield f"data: {base64.b64encode(audio).decode()}\n\n"
```

```
9
10      return Response(generate(), mimetype='text/event-stream')
```

Kode 3.9: Audio Streaming Endpoint

Kode ini merupakan implementasi *endpoint* `/api/stream` yang digunakan untuk melakukan *streaming* data audio ke klien secara real-time. Fungsi `generate()` akan terus berjalan dalam sebuah *loop* tak hingga dan secara berkala mengambil respons teks dari fungsi `get_next_response()`. Setiap teks yang diperoleh akan dikonversi menjadi audio melalui fungsi `generate_speech()`, kemudian hasil audio tersebut dikodekan ke dalam format *Base64*. Hasil *encoding* ini kemudian dikirimkan menggunakan protokol *Server-Sent Events* (SSE) dengan `mimetype text/event-stream`. Pendekatan ini memungkinkan klien menerima aliran data audio secara bertahap dan responsif, yang sangat penting dalam membangun pengalaman interaksi suara yang lancar dan real-time.

Flowchart



Gambar 3.5. Flowchart Proses Interaksi Suara dengan Sistem CORA

Kendala

Pada tahap pengembangan Sprint 2, terdapat beberapa tantangan utama dalam implementasi sistem interaksi berbasis suara. Pertama, proses perekaman audio di sisi klien mengalami kendala kompatibilitas lintas browser, terutama dalam hal format file dan izin akses mikrofon. Kedua, transmisi audio ke backend memerlukan pengaturan buffer dan pengelolaan blob secara tepat agar file dapat dikirim tanpa korupsi. Ketiga, penggunaan layanan transkripsi suara (*Speech-to-Text*) dari OpenAI melalui model Whisper menghasilkan latensi yang cukup tinggi, terutama pada koneksi jaringan yang kurang stabil. Selain itu, integrasi fitur *Text-to-Speech* (TTS) menghadapi tantangan dalam pemilihan suara dan bahasa yang sesuai, serta kendala teknis dalam pengelolaan hasil audio yang dihasilkan secara streaming. Proses streaming audio juga memerlukan penyesuaian dari sisi format data agar dapat diputar secara *real-time* di frontend tanpa jeda yang signifikan.

Solusi

Untuk mengatasi kendala tersebut, dilakukan beberapa pendekatan teknis yang terintegrasi. Pada sisi *frontend*, implementasi perekaman audio dilakukan menggunakan API `MediaRecorder`, dengan pengaturan yang memastikan format WebM diterima secara luas. Audio yang direkam dikemas dalam bentuk `Blob` dan dikirim ke *backend* menggunakan `FormData` melalui metode `POST`. Di sisi backend, layanan *Speech-to-Text* menggunakan model `whisper-1` dari OpenAI untuk mendapatkan hasil transkripsi dengan akurasi tinggi. Proses ini dibungkus dengan mekanisme `try-except` untuk menjamin ketahanan terhadap kegagalan input. Untuk fitur TTS, digunakan layanan Azure TTS dengan konfigurasi neural voice untuk menghasilkan suara yang alami. Audio hasil sintesis dikembalikan dalam format PCM dan dikodekan dengan Base64 sebelum dikirim ke *frontend* menggunakan *Server-Sent Events* (SSE). Strategi ini memungkinkan pemutaran suara secara bertahap dengan latensi rendah, sehingga interaksi antara pengguna dan sistem CORA dapat berlangsung secara responsif dan natural.

Sprint 3: Perbaikan Fitur *Greeting* pada CORA

Pada Sprint ke-3, dilakukan pengembangan dan perbaikan terhadap fitur *greeting* untuk menunjang komunikasi interaktif antara pengguna dan aplikasi CORA. Perbaikan ini didasarkan pada temuan saat sesi *demo*, di mana CORA

terdeteksi memberikan sapaan kepada pengguna lain di tengah percakapan yang masih berlangsung. Masalah ini disebabkan oleh deteksi wajah yang tidak membedakan pengguna aktif dengan wajah baru yang muncul di latar belakang. Untuk mengatasi hal tersebut, dilakukan penyesuaian pada logika deteksi wajah dan pengelolaan sesi pengguna agar interaksi menjadi lebih stabil, personal, dan tidak terganggu oleh interupsi dari pengguna lain yang tidak relevan dalam sesi aktif.

Proses

Pada Sprint ke-3, dilakukan penyempurnaan fitur *greeting* untuk meningkatkan stabilitas interaksi pengguna dengan CORA. Masalah utama yang diatasi adalah:

- Deteksi wajah yang tidak akurat menyebabkan greeting berulang
- Interupsi greeting saat percakapan sedang berlangsung
- Tidak adanya manajemen state pengguna yang komprehensif

Implementasi Teknis

Enhanced Face Tracking* dengan *State Management

```
tracked_faces = {}  # {face_id: {"box": (x1,y1,x2,y2), "last_seen": timestamp, "greeted": bool}}
current_active_face = None  # Wajah yang sedang aktif berinteraksi

def get_face_id(current_box, current_time):
    global tracked_faces, current_active_face

    for face_id, data in tracked_faces.items():
        iou = calculate_iou(current_box, data["box"])
        if iou > FACE_TRACKING_THRESHOLD:
            tracked_faces[face_id]["last_seen"] = current_time
            tracked_faces[face_id]["box"] = current_box
            current_active_face = face_id
            return face_id

    face_id = f"face_{int(current_time * 1000)}"
    tracked_faces[face_id] = {
        "box": current_box,
```

```
        "last_seen": current_time,
        "greeted": False
    }
    current_active_face = face_id
    return face_id
```

Kode 3.10: Sistem Pelacakan Wajah Terbaru

Perbaikan Utama:

- Sistem pelacakan wajah berbasis state yang lebih robust
- Penyimpanan informasi bounding box dan timestamp terakhir
- Flag `greeted` untuk menandai wajah yang sudah disapa
- Variabel `current_active_face` untuk melacak pengguna aktif

Mekanisme Greeting yang Lebih Cerdas

```
def should_greet(face_id, current_time):
    global tracked_faces, last_greeting_time, current_active_face

    if current_time - last_greeting_time < COOLDOWN_TIME:
        return False

    face_data = tracked_faces.get(face_id)
    if not face_data:
        return False

    # Jika belum pernah disapa atau absence time melebihi
    threshold
    if (not face_data["greeted"] or
        (current_time - face_data["last_seen"] > MAX_ABSENCE_TIME
    and
         current_active_face == face_id)):
        tracked_faces[face_id]["greeted"] = True
        return True

    return False
```

Kode 3.11: Logika Greeting Terkini

Peningkatan:

- *Cooldown* global 10 detik antar greeting
- Greeting hanya untuk wajah aktif (`current_active_face`)
- Sistem *absence time* (30 detik) untuk wajah yang kembali
- Pembersihan wajah yang tidak terdeteksi dalam waktu lama

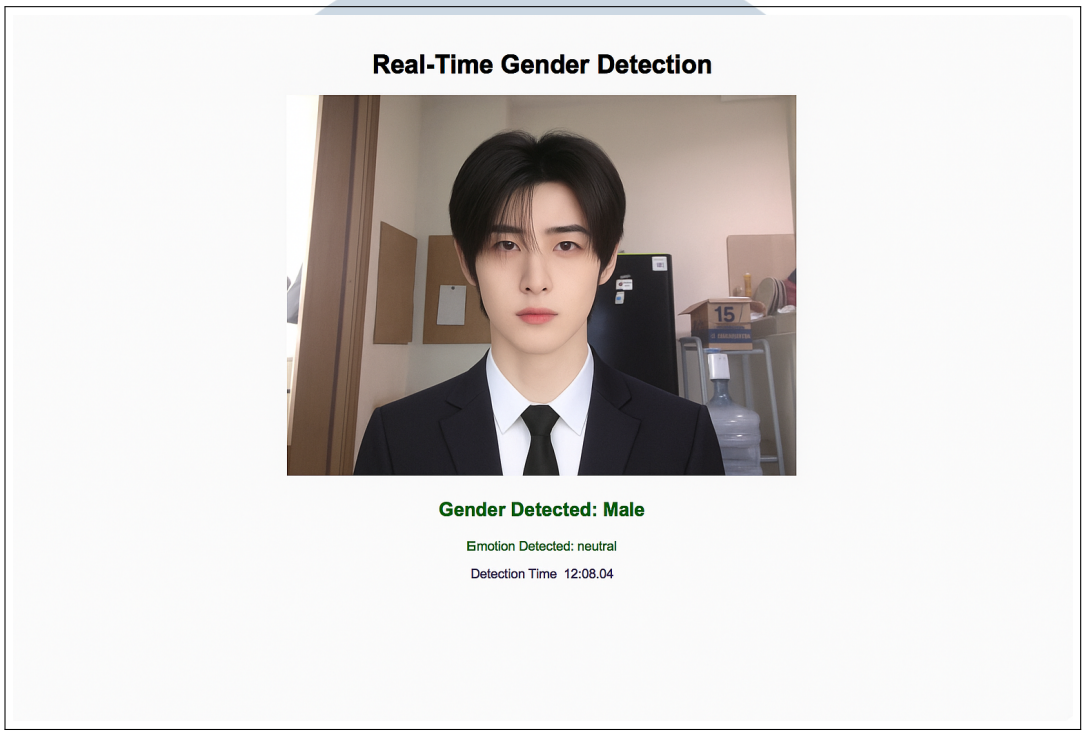
***WebSocket Client* yang Dioptimalkan**

```
const WebSocketGreeting = ({ onGreetingRequest }) => {
    const lastGreetingTime = useRef(0);
    const greetingCooldown = 10000; // 10 detik

    const handleDetection = (data) => {
        const currentTime = Date.now();
        if (currentTime - lastGreetingTime.current <
    greetingCooldown) {
            console.log("Skipping greeting due to cooldown");
            return;
        }

        lastGreetingTime.current = currentTime;
        onGreetingRequest({
            gender: formatGender(data.gender),
            time: data.time,
            emotion: data.emotion,
            tone: formatTone(tone)
        });
    };

    useEffect(() => {
        const socket = io(backendUrl, {
            reconnectionAttempts: 3,
            reconnectionDelay: 1000,
            timeout: 20000
        });

        socket.on("detection", handleDetection);
        return () => socket.disconnect();
    }, [tone]);
};
```

Kode 3.12: Implementasi WebSocket Client

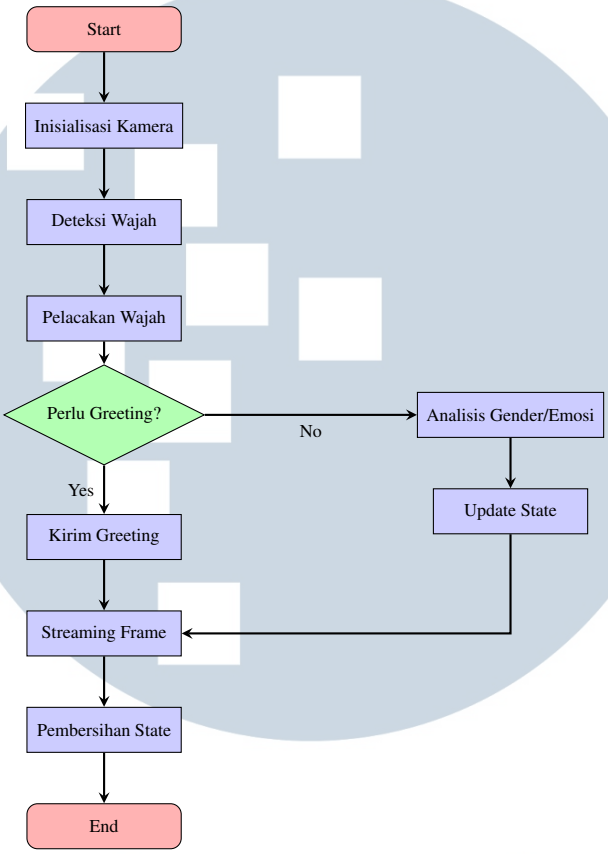
Tampilan *User Interface* (UI)

Berikut ini adalah *screenshoot* dari antarmuka *backend*:



Gambar 3.6. Tampilan UI *backend*

Flowchart



Gambar 3.7. Flowchart Proses Deteksi dan Greeting

Kendala

Dalam implementasi perbaikan fitur *greeting*, terdapat beberapa kendala teknis yang signifikan. Pertama, masalah *synchronization* antara *cooldown* di sisi *server* dan *client* sering menyebabkan inkonsistensi dalam pembatasan *greeting*. Kedua, optimalisasi nilai *threshold Intersection over Union* (IoU) menjadi tantangan tersendiri karena harus menyesuaikan dengan berbagai kondisi pencahayaan yang berbeda-beda di lingkungan pengguna. Ketiga, manajemen koneksi *WebSocket* yang tidak stabil terkadang menyebabkan terputusnya aliran data *real-time* antara *frontend* dan *backend*.

Solusi

Untuk mengatasi berbagai kendala tersebut, perlu menerapkan beberapa solusi teknis. Pertama, dilakukan implementasi sistem *threshold* IoU dinamis yang mampu menyesuaikan diri berdasarkan kualitas gambar dan kondisi pencahayaan. Kedua, ditambahkan mekanisme *heartbeat* untuk mempertahankan koneksi *WebSocket* tetap stabil, dengan interval pengiriman *ping* setiap 15 detik. Ketiga, sistem *logging* yang komprehensif diimplementasikan pada seluruh komponen kritis untuk mempermudah proses *debugging* masalah deteksi wajah. Terakhir, diterapkan kebijakan *cooldown* bertingkat yang terdiri dari *cooldown* global (10 detik) dan *cooldown* per-wajah (30 detik) untuk menyeimbangkan antara responsivitas sistem dan stabilitas interaksi.

Sprint 4: Implementasi Fitur Memori Konteks Menggunakan LangChain

Pada Sprint ke-4, fokus pengembangan diarahkan pada implementasi fitur memori konteks agar sistem CORA dapat mempertahankan pemahaman terhadap percakapan sebelumnya. Dengan adanya fitur ini, pengguna dapat melakukan interaksi berkelanjutan tanpa perlu mengulangi konteks pertanyaan secara eksplisit pada setiap pernyataan.

Teknologi utama yang digunakan adalah *LangChain*, sebuah framework modular untuk membangun aplikasi berbasis bahasa alami yang memungkinkan pengelolaan state percakapan secara efisien. Fitur memori dikembangkan dengan menggunakan pendekatan `ConversationBufferMemory` dari LangChain, yang menyimpan rekam jejak dialog antara pengguna dan sistem. Memori ini kemudian dikirim kembali sebagai bagian dari prompt ke model LLM agar respons yang dihasilkan lebih relevan dan kontekstual.

Implementasi fitur memori ini merupakan fondasi penting untuk menciptakan pengalaman percakapan yang lebih alami dan manusiawi, di mana sistem tidak hanya menjawab berdasarkan pertanyaan terakhir, tetapi juga memahami alur percakapan secara keseluruhan. Selain itu, mekanisme reset atau penghapusan memori juga disiapkan untuk memastikan pengelolaan konteks tetap terkendali dan sesuai kebutuhan.

Proses

1. Inisialisasi Memory Buffer

Pembuatan buffer memori untuk menyimpan histori percakapan menggunakan LangChain.

```
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory()
last_response = None
```

Kode 3.13: Inisialisasi Memory

- Menggunakan `ConversationBufferMemory` dari LangChain
- Variabel global `last_response` untuk menyimpan respons terakhir
- Memory buffer akan menyimpan seluruh histori percakapan dalam sesi

2. Penyimpanan Histori Percakapan

Mekanisme penyimpanan setiap interaksi ke dalam memory buffer.

```
def add_to_history(user_input, bot_response):
    global last_response
    memory.save_context({"input": user_input}, {"output":
    bot_response})
    last_response = bot_response
    log_to_supabase(
        "history",
        "success",
        "Added to history",
        {"user_input": user_input, "bot_response": bot_response}
    )
```

Kode 3.14: Penyimpanan Histori

- Menyimpan pasangan input pengguna dan output sistem
- Update `last_response` dengan respons terbaru
- Logging ke Supabase untuk pelacakan histori
- Format penyimpanan sesuai kebutuhan LangChain

3. Pengambilan Histori Percakapan

Fungsi untuk mengambil histori percakapan yang tersimpan.

```
def get_session_history():
    history = memory.load_memory_variables({})
    log_to_supabase(
        "history",
        "success",
        "Session history loaded",
        {"history": history.get('history', ''), "last_response":
    last_response}
    )
    return {
        'history': history.get('history', ''),
        'last_response': last_response
    }
```

Kode 3.15: Pengambilan Histori

- Mengembalikan histori dalam format yang dapat digunakan langsung
- Termasuk respons terakhir yang terpisah untuk kebutuhan khusus
- Logging ke Supabase untuk audit trail
- Menggunakan `load_memory_variables` dari LangChain

4. Integrasi dengan LLM

Pemanfaatan memori konteks dalam pembuatan respons oleh model bahasa.

```
history_data = get_session_history()
conversation_history = history_data.get('history', '')

llm_handler = LLMHandler()
llm_use_case = LLMUseCase(llm_handler)
chatbot_response = llm_use_case.generate_response(
    text=user_text,
    history=conversation_history,
    mode=response_mode,
    language=language
)

```

```
13 add_to_history(user_text, chatbot_response)
```

Kode 3.16: Integrasi Memory dengan LLM

- Mengambil histori sebelum memproses permintaan baru
- Meneruskan histori ke LLM sebagai bagian dari prompt
- Menyimpan interaksi baru ke histori setelah mendapatkan respons
- Mempertahankan konteks percakapan antar permintaan

Kendala

- Kurangnya pemahaman awal terkait struktur LangChain, khususnya dalam menggabungkan `LLMChain`, `Memory`, dan `PromptTemplate`.
- Terjadinya over-context, yaitu konteks terlalu panjang sehingga respons menjadi tidak fokus atau melenceng.
- Tidak adanya mekanisme sesi pengguna (seperti Redis) menyulitkan penanganan percakapan simultan atau identifikasi user.

Solusi

- Melakukan eksplorasi dokumentasi resmi LangChain dan mencoba berbagai jenis memory, termasuk `ConversationSummaryBufferMemory`, untuk efisiensi konteks.
- Menambahkan batasan panjang konteks yang dikirim ke LLM agar tetap dalam batas token maksimum.
- Menambahkan tombol *"Akhiri Percakapan"* di antarmuka pengguna untuk menghapus memori secara manual sebagai pengganti sistem sesi otomatis.

Sprint 5: Penambahan Fitur Mode Respons "General" dan "Detail"

Pada Sprint ke-5, dilakukan penambahan fitur mode respons pada sistem CORA yang memungkinkan pengguna untuk memilih tingkat kedalaman informasi yang diinginkan, yaitu dalam bentuk mode "General" maupun "Detail". Fitur ini

dirancang guna meningkatkan fleksibilitas dan relevansi jawaban yang diberikan oleh sistem berdasarkan preferensi masing-masing pengguna.

Mode "General" ditujukan untuk menghasilkan jawaban yang bersifat ringkas dan langsung kepada inti dari pertanyaan, sementara mode "Detail" disusun agar memberikan uraian yang lebih lengkap, menyeluruh, dan informatif. Dengan diterapkannya fitur ini, sistem diharapkan mampu menyajikan respons yang lebih adaptif serta sesuai dengan kebutuhan konteks percakapan yang berlangsung.

Implementasi fitur ini merupakan bagian dari upaya peningkatan kualitas interaksi antara pengguna dan sistem, serta mendukung penyajian informasi yang lebih terstruktur dan responsif terhadap dinamika permintaan pengguna.

Proses

Pada Sprint 5, sistem CORA ditingkatkan dengan menambahkan fitur pemilihan tingkat kedalaman informasi melalui mode *General* dan *Detail*. Implementasi ini bertujuan untuk memberikan pengalaman interaksi yang lebih fleksibel dan sesuai dengan kebutuhan pengguna.

Sistem dirancang dengan alur sebagai berikut:

1. Pengguna memilih mode melalui antarmuka
2. Parameter mode dikirim bersamaan dengan pertanyaan
3. Backend memproses permintaan sesuai mode yang dipilih
4. Respons dikembalikan dalam format yang sesuai

1. Frontend - Pemilihan Mode

```
// State untuk menyimpan mode yang dipilih
const [responseMode, setResponseMode] = useState<"general" | "
    detailed">("general");

// Komponen pemilih mode
<div className={classes.modeSelector}>
  <button
    onClick={() => setResponseMode('general')}
    className={`${classes.modeButton} ${
```

```
      responseMode === 'general' ? classes.activeMode : classes.
    inactiveMode
    }`}
    disabled={isThinking || isSpeaking}
  >
    {t('General')}
  </button>
  <button
    onClick={() => setResponseMode('detailed')}
    className={`${classes.modeButton} ${
      responseMode === 'detailed' ? classes.activeMode : classes.
    inactiveMode
    }`}
    disabled={isThinking || isSpeaking}
  >
    {t('Detailed')}
  </button>
</div>
```

Kode 3.17: Implementasi Mode Selector di Frontend

Fitur Utama:

- State management untuk mode respons
- Toggle button dengan visual feedback
- Disable state selama proses berpikir/berbicara

2. Backend - Pemrosesan Berdasarkan Mode

```
def generate_response(self, text, history="", mode="general",
    language="id"):
    language_instruction = {
        "id": "Always answer in Bahasa Indonesia",
        "en": "Always answer in English"
    }.get(language, "Always answer in Bahasa Indonesia")

    system_prompt = (
        "You are an AI receptionist that helps users. "
        f"{language_instruction}\n"
    )

    if mode == "general":
```

```
        system_prompt += (
            "Provide a brief answer (3-5 sentences). "
            "Focus on key information only."
        )
    else:
        system_prompt += (
            "Provide a detailed answer (100 sentences). "
            "Include relevant details but keep it concise."
        )

    user_prompt = f"{history}\nUser: {text}\nAI:"
    return self.llm_handler.send_prompt(
        system_prompt=system_prompt,
        user_prompt=user_prompt
    )
```

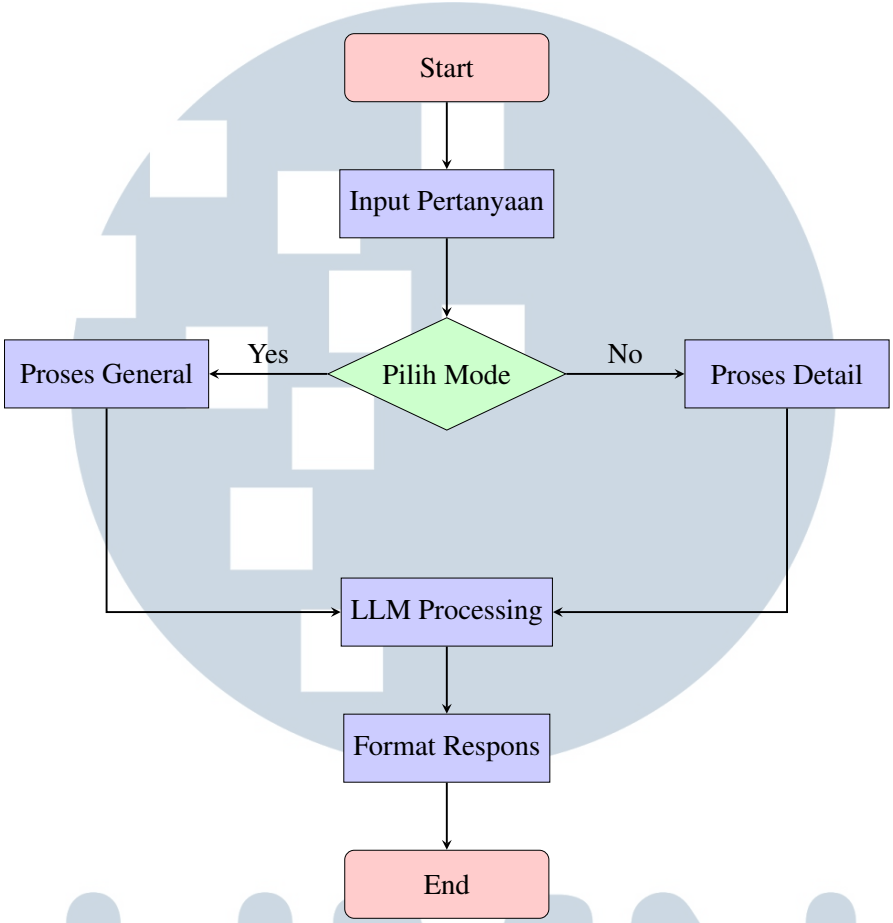
Kode 3.18: Implementasi LLM Use Case

Perbedaan Mode:

Aspek	**General**	**Detail**
Jumlah Kalimat	3–5 kalimat	6–100 kalimat
Kedalaman Informasi	Menyampaikan poin utama saja	Memberikan penjelasan secara mendalam dan terperinci
Contoh Jawaban	*Jam buka: 08.00–17.00*	*Jam buka pukul 08.00–17.00 dengan istirahat makan siang pada pukul 12.00–13.00*

Tabel 3.9. Perbandingan Mode General dan Detail dalam Sistem CORA

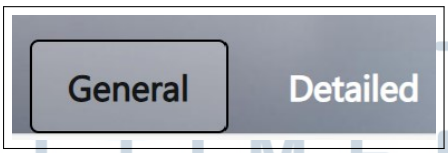
Flowchart



Gambar 3.8. Flowchart Pemrosesan Mode Respons

Tampilan *User Interface* (UI)

Berikut ini adalah *screenshoot* dari antarmuka fitur general dan detail:



Gambar 3.9. Tampilan UI Mode General dan Detail CORA

Kendala

Pengembangan fitur ini menghadapi beberapa tantangan teknis. Pertama, perlu adanya penyesuaian *prompt engineering* untuk menghasilkan respons yang sesuai dengan karakteristik masing-masing mode. Kedua, perbedaan waktu respons antara mode General dan Detail yang signifikan dapat mempengaruhi pengalaman pengguna. Terakhir, menjaga konsistensi kualitas informasi antara kedua mode menjadi tantangan tersendiri.

Solusi

Untuk mengatasi tantangan tersebut, perlu menerapkan beberapa solusi. Sistem *caching* diimplementasikan untuk menyimpan respons umum yang sering diminta. Dilakukan juga optimasi *prompt template* untuk memastikan perbedaan yang jelas antara kedua mode. Selain itu, ditambahkan mekanisme *loading state* yang informatif untuk memberi tahu pengguna saat menunggu respons yang lebih panjang pada mode Detail.

Sprint 6: Penambahan Fitur Mode Bahasa, Pengulangan Jawaban, dan Pengaturan Volume Suara

Pada Sprint ke-6, dilakukan pengembangan beberapa fitur tambahan guna meningkatkan kualitas interaksi pengguna dengan sistem CORA. Fitur-fitur tersebut meliputi pengaturan mode bahasa (Indonesia/Inggris), pengulangan jawaban, serta pengaturan volume output suara.

Fitur mode bahasa memungkinkan pengguna memilih bahasa interaksi sesuai preferensi, sehingga sistem dapat menyampaikan jawaban dalam bahasa Indonesia atau Inggris secara dinamis. Hal ini ditujukan untuk menjangkau lebih banyak pengguna serta meningkatkan aksesibilitas sistem secara umum.

Selanjutnya, fitur pengulangan jawaban ditambahkan agar pengguna dapat meminta sistem untuk mengulang respons sebelumnya, terutama apabila terjadi gangguan dalam pemrosesan audio atau ketidakterpahaman informasi. Hal ini mendukung keterandalan sistem dalam komunikasi berkelanjutan.

Proses

1. Fitur Pengaturan Bahasa

Implementasi fitur bahasa menggunakan react-i18n untuk manajemen terjemahan multi-bahasa.

```
const [language, setLanguage] = useState<'id' | 'en'>('id');
const { t, i18n } = useTranslation();
// Tombol toggle bahasa
<div className={classes.languageToggle}>
  <button onClick={() => {
    setLanguage('id');
    i18n.changeLanguage('id');
  }}>
    IDN
  </button>
  <button onClick={() => {
    setLanguage('en');
    i18n.changeLanguage('en');
  }}>
    ENG
  </button>
</div>
```

Kode 3.19: Toggle Bahasa

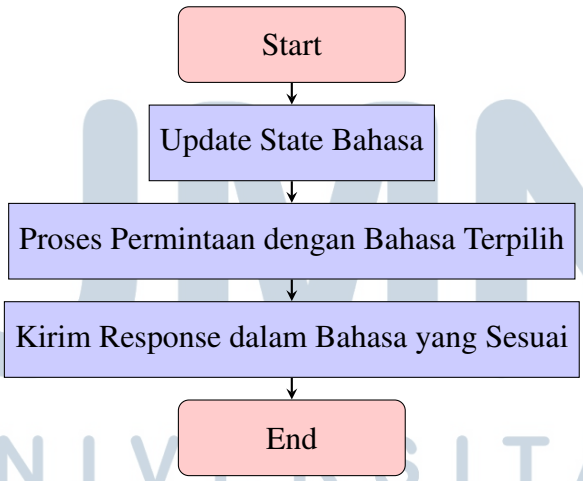
```
def generate_response(self, text, history="", mode="general",
    language="id"):
    language_instruction = {
        "id": "Always answer in Bahasa Indonesia",
        "en": "Always answer in English"
    }.get(language, "Always answer in Bahasa Indonesia")

    system_prompt = f"""
    You are an AI receptionist.
    {language_instruction}
    Provide {'brief' if mode=='general' else 'detailed'} answers.
    """
    return self.llm_handler.send_prompt(
        system_prompt=system_prompt,
        user_prompt=f"{history}\nUser: {text}\nAI:"
    )
```

Kode 3.20: Penanganan Bahasa

Fitur pengaturan bahasa memungkinkan pengguna untuk memilih tampilan antarmuka dan bahasa jawaban AI, yaitu antara Bahasa Indonesia dan Bahasa Inggris. Pada sisi *frontend*, implementasi dilakukan menggunakan pustaka `react-i18next` untuk mendukung manajemen multi-bahasa. State `language` digunakan untuk menyimpan bahasa yang sedang aktif, dan fungsi `i18n.changeLanguage()` digunakan untuk mengubah bahasa aplikasi secara global. Dua buah tombol bertuliskan `IDN` dan `ENG` disediakan untuk pengguna dalam memilih bahasa yang diinginkan.

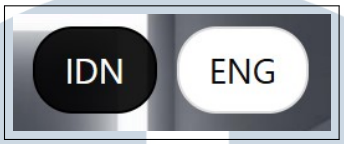
Di sisi *backend*, pilihan bahasa dari pengguna dikirim melalui parameter `language` pada fungsi `generate_response()`. Parameter ini kemudian digunakan untuk membentuk `system_prompt`, yang memberikan instruksi eksplisit kepada model agar menjawab dalam bahasa yang sesuai, seperti "*Always answer in Bahasa Indonesia*" atau "*Always answer in English*". Dengan pendekatan ini, sistem dapat memberikan respons AI yang konsisten dengan pilihan bahasa antarmuka pengguna, sehingga meningkatkan pengalaman interaksi secara keseluruhan.

Flowchart



Gambar 3.10. Flowchart Pengaturan Bahasa

Tampilan *User Interface* (UI)



Gambar 3.11. Tampilan UI Pemilihan Bahasa

2. Fitur Pengulangan Jawaban

Fitur untuk mengulang jawaban terakhir yang diberikan sistem.

```
const [lastQuestion, setLastQuestion] = useState<string | null>(
    null);

const handleRepeat = async () => {
  if (!lastQuestion) return;

  const formData = new FormData();
  formData.append("text", lastQuestion);
  formData.append("isRepeat", "true");

  const response = await fetch("/chat", {
    method: "POST",
    body: formData
  });
  // ... handle response ...
};
```

Kode 3.21: Handle Repeat

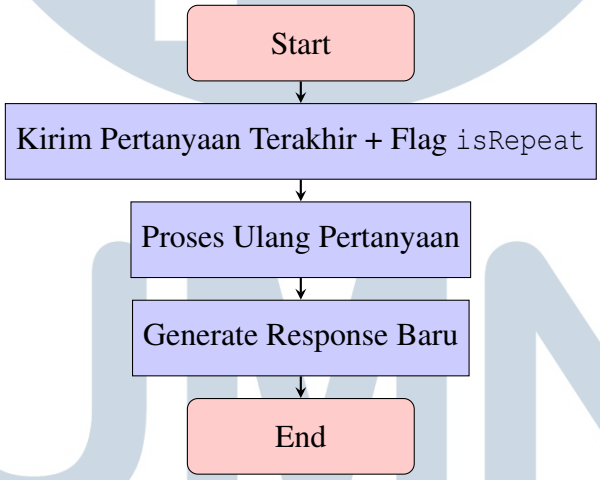
```
@app.route("/chat", methods=["POST"])
def handle_conversation():
    is_repeat = request.form.get("isRepeat", "false").lower() == "
    true"
    user_text = request.form.get("text", "")

    if is_repeat:
        print(f"Processing repeat request for: {user_text}")
        # Gunakan cache atau proses ulang pertanyaan
```

Kode 3.22: Penanganan Repeat

Fitur ini memungkinkan pengguna untuk mengulangi pertanyaan terakhir yang telah diajukan tanpa harus mengetiknya kembali. Di sisi *frontend*, state `lastQuestion` digunakan untuk menyimpan input terakhir dari pengguna. Ketika pengguna menekan tombol ulang (`handleRepeat`), data pertanyaan sebelumnya dikirim kembali ke endpoint `/chat` melalui metode `POST`, disertai dengan penanda `isRepeat` bernilai `"true"`.

Pada sisi *backend*, nilai `isRepeat` diperiksa melalui `request.form`, dan apabila bernilai `true`, sistem akan mengeksekusi logika khusus untuk permintaan ulang. Hal ini dapat digunakan untuk mengakses hasil sebelumnya dari cache atau memproses ulang pertanyaan yang sama. Dengan implementasi ini, sistem menjadi lebih ramah pengguna karena tidak mengharuskan pengguna menginput ulang pertanyaan yang sama secara manual.

Flowchart



Gambar 3.12. Flowchart Pengulangan Jawaban

Tampilan *User Interface* (UI)

Gambar 3.13. Tampilan UI *Repeat Answer*

Kendala

Selama implementasi fitur-fitur pada Sprint 6, tim menghadapi sejumlah tantangan teknis yang cukup kompleks. Pertama, terdapat ketidakstabilan dalam sistem multibahasa, khususnya akibat keterbatasan dukungan *Text-to-Speech* (TTS) untuk Bahasa Indonesia dan inkonsistensi terjemahan pada konten yang bersifat dinamis. Kedua, manajemen *state* pada fitur pengulangan jawaban belum optimal, sehingga menimbulkan tumpang tindih audio serta hilangnya konteks pertanyaan terakhir. Ketiga, pengaturan volume tidak bersifat persistensi antar sesi dan terjadi keterlambatan saat melakukan pembaruan secara real-time. Keempat, sistem penanganan kesalahan (*error handling*) belum mendukung penyesuaian terhadap bahasa aktif, serta cakupan *logging* masih terbatas. Terakhir, interaksi cepat antar fitur menyebabkan konflik sumber daya dan kondisi balapan (*race condition*) pada beberapa skenario penggunaan.

Solusi

Untuk mengatasi kendala-kendala tersebut, sejumlah solusi teknis terintegrasi berhasil diimplementasikan. Pertama, dibangun sistem *fallback* untuk TTS dan mekanisme *caching* terjemahan dengan pendekatan *lazy loading* guna memastikan ketersediaan bahasa secara efisien. Kedua, dilakukan optimalisasi manajemen *state* melalui `sessionStorage` serta pembersihan buffer audio (*audio buffer cleanup*) sebelum pengulangan jawaban. Ketiga, diterapkan penyimpanan volume secara lokal menggunakan `localStorage` dan pembaruan paksa (*forced update*) pada instance audio untuk menjamin konsistensi. Keempat, sistem penanganan kesalahan diperluas dengan *error mapping* multibahasa dan integrasi *extended logging* ke platform Supabase. Kelima, diimplementasikan *debouncing* pada interaksi pengguna, pembersihan sumber daya melalui `useEffect`, serta mekanisme `mutex lock` di sisi *backend* untuk mencegah konflik proses. Seluruh solusi ini secara signifikan meningkatkan stabilitas, konsistensi, dan keandalan sistem secara keseluruhan.

Sprint 7: Pengembangan Halaman Admin untuk Manajemen File Berbasis RAG

Pada Sprint ke-7, dilakukan pengembangan halaman admin untuk mengelola dokumen internal yang menjadi sumber pengetahuan tambahan dalam sistem

CORA. Halaman ini memungkinkan admin untuk mengunggah dan menghapus file, serta secara otomatis mengintegrasikan konten dokumen tersebut ke dalam sistem melalui pendekatan Retrieval-Augmented Generation (RAG).

Dalam implementasinya, dokumen yang diunggah akan diproses dengan melakukan ekstraksi konten dan konversi menjadi embedding vector. Proses ini bertujuan untuk menyimpan representasi semantik dari isi dokumen ke dalam basis data vektor (vector database), yang selanjutnya digunakan sebagai referensi dalam menjawab pertanyaan pengguna. Dengan demikian, sistem mampu melakukan pencarian informasi yang relevan dari dokumen berdasarkan konteks pertanyaan yang diberikan.

Pengelolaan file dan data pendukung dilakukan menggunakan Supabase sebagai layanan backend yang menyediakan autentikasi, penyimpanan file, serta pengelolaan metadata. Admin dapat menghapus file yang tidak diperlukan, sehingga menjaga akurasi dan efisiensi dalam proses penelusuran informasi.

Penggunaan RAG dan vector database dalam sprint ini menjadi landasan penting bagi kemampuan CORA dalam menjawab pertanyaan berbasis dokumen secara dinamis, kontekstual, dan lebih informatif.

Proses

1. Autentikasi Admin

Proses dimulai dengan implementasi sistem autentikasi untuk halaman admin menggunakan NextAuth.js.

```
export const authOptions: NextAuthOptions = {
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "text" },
        password: { label: "Password", type: "password" }
      },
      async authorize(credentials) {
        const user = dummyUsers.find(u =>
          u.email === credentials?.email &&
          u.password === credentials?.password
        );
        return user || null;
      }
```

```
    })
  ],
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.role = user.role;
        token.id = user.id;
      }
      return token;
    },
    async session({ session, token }) {
      if (session.user) {
        session.user.role = token.role;
        session.user.id = token.id;
      }
      return session;
    }
  },
  pages: {
    signIn: "/login",
    error: "/login"
  }
};
```

Kode 3.23: Konfigurasi Autentikasi

- Menggunakan `CredentialsProvider` untuk autentikasi berbasis email/password
- Menyimpan informasi role dan ID user dalam token JWT
- Mengarahkan ke halaman login jika terjadi error

2. Upload Dokumen

Proses upload dokumen melibatkan ekstraksi teks dan konversi ke embedding vector.

```
const handleUpload = async () => {
  // 1. Ekstrak teks dari file
  const fileContent = await extractTextFromFile(file);

  // 2. Buat FormData untuk upload
```

```
  const formData = new FormData();
  formData.append('file', file);
  formData.append('content', fileContent);

  // 3. Upload ke server
  const response = await fetch('/api/files', {
    method: 'POST',
    body: formData
  });

  // 4. Handle response
  if (response.ok) {
    onUploadSuccess();
  }
};
```

Kode 3.24: Proses Upload File (FileUpload.tsx)

- Ekstraksi teks dilakukan di client-side untuk mengurangi beban server
- File dan konten teks dikirim ke endpoint API
- Progress bar menampilkan status ekstraksi dan upload

3. Penyimpanan Dokumen di Supabase

Dokumen yang diupload disimpan di Supabase Storage dan metadata-nya di database.

```
// Upload file ke storage
const { error: uploadError } = await supabase.storage
  .from(BUCKET_NAME)
  .upload(filePath, fileBuffer, {
    contentType: file.type
  });

// Simpan metadata ke database
const { data: dbData, error: dbError } = await supabase
  .from('files')
  .insert({
    name: file.name,
    type: fileExt,
    size: file.size,
```

```
    url: urlData.publicUrl,
    content: content,
    embedding: embedding
  });
```

Kode 3.25: API Penyimpanan File (route.ts)

- File disimpan di bucket 'files' di Supabase Storage
- Metadata termasuk nama, ukuran, URL, konten teks, dan embedding vector
- Embedding dihasilkan menggunakan OpenAI API

4. Pencarian Semantik

Implementasi pencarian berbasis embedding vector menggunakan fungsi PostgreSQL.

```
CREATE OR REPLACE FUNCTION search_files(
  query_embedding VECTOR(1536),
  similarity_threshold FLOAT DEFAULT 0.7
) RETURNS TABLE (
  id UUID,
  name TEXT,
  similarity FLOAT
) AS $$
  SELECT id, name, 1 - (embedding <=> query_embedding) AS
    similarity
  FROM files
  WHERE 1 - (embedding <=> query_embedding) > similarity_threshold
  ORDER BY similarity DESC;
$$ LANGUAGE SQL;
```

Kode 3.26: Fungsi Pencarian di Supabase)

- Menggunakan operasi cosine similarity (<=>)
- Hanya mengembalikan hasil dengan similarity di atas threshold
- Hasil diurutkan berdasarkan similarity score

5. Integrasi RAG

Dokumen yang diupload juga diproses untuk sistem Retrieval-Augmented Generation.

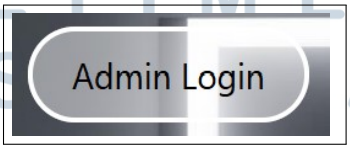
```python
class RAGService:
    def __init__(self):
        self.embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
        self.vector_store = SupabaseVectorStore(
            client=self.client,
            embedding=self.embeddings,
            table_name="documents"
        )

    def query_documents(self, query: str, k: int = 5):
        results = self.vector_store.similarity_search(query, k=k)
        return [
            {"content": doc.page_content, "metadata": doc.metadata}
            for doc in results
        ]
```

Kode 3.27: Implementasi RAG Service (rag_service.py)

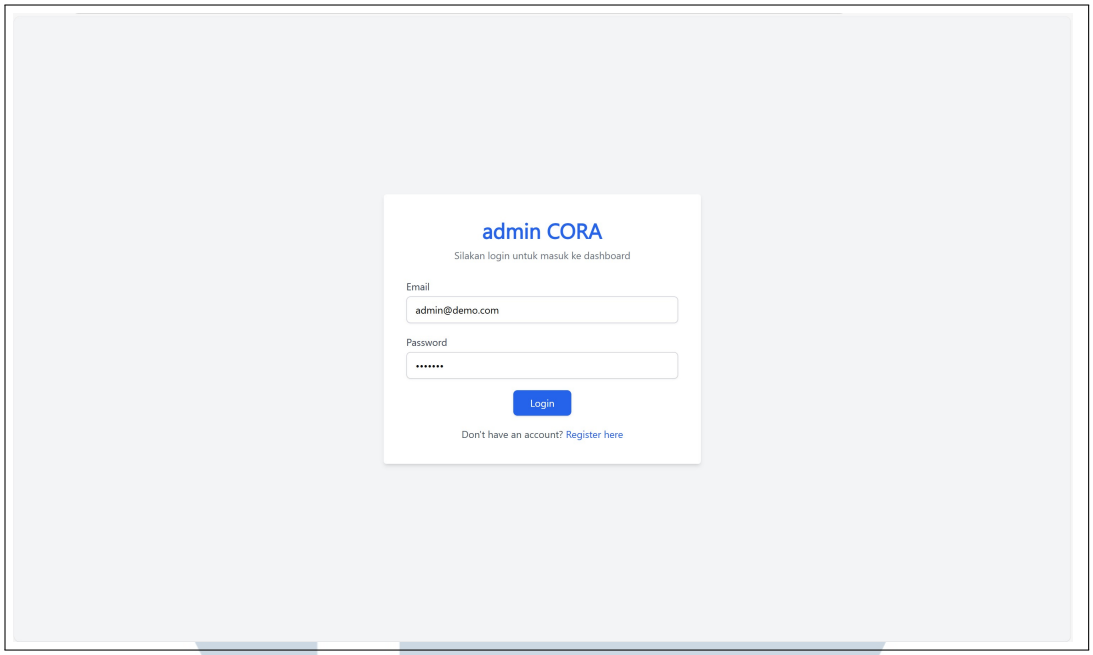
- Menggunakan LangChain untuk manajemen vector store
- OpenAI Embeddings untuk konversi teks ke vector
- SupabaseVectorStore untuk penyimpanan dan query dokumen

Tampilan *User Interface* (UI)

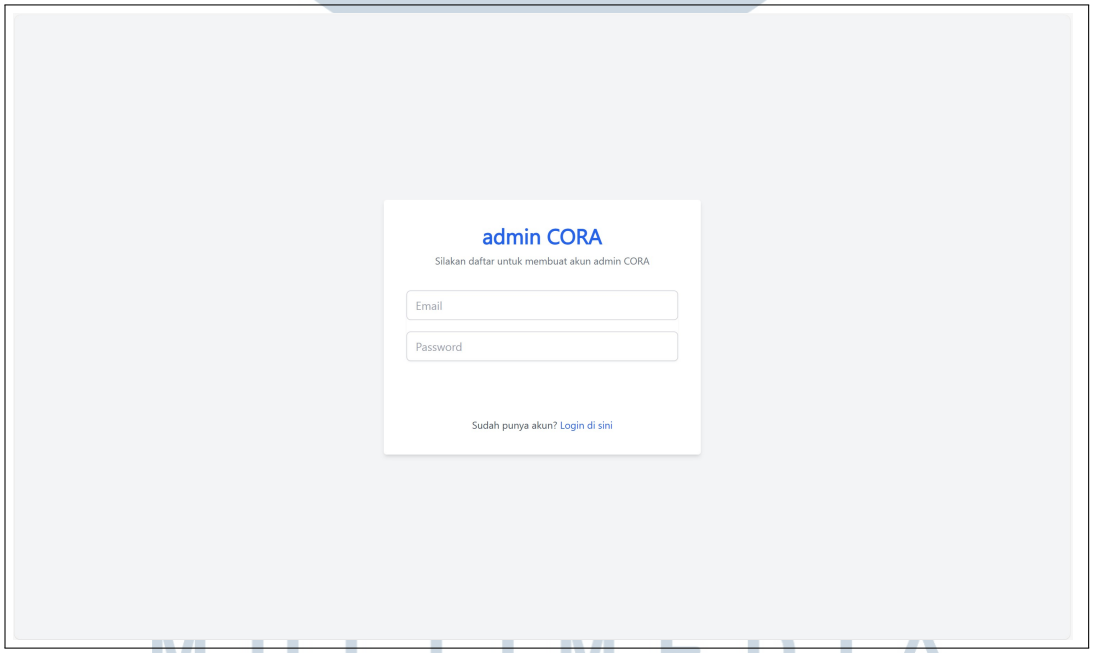
Berikut ini adalah *screenshoot* dari antarmuka *login page*, *register page*, *admin dashboard*, dan supabase:



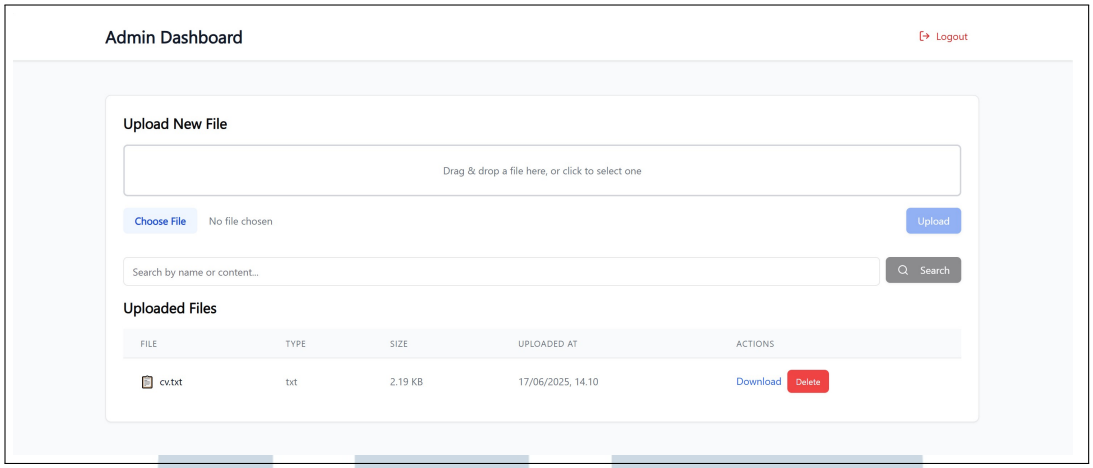
Gambar 3.14. Tampilan UI Tombol Login di *Main Page* CORA



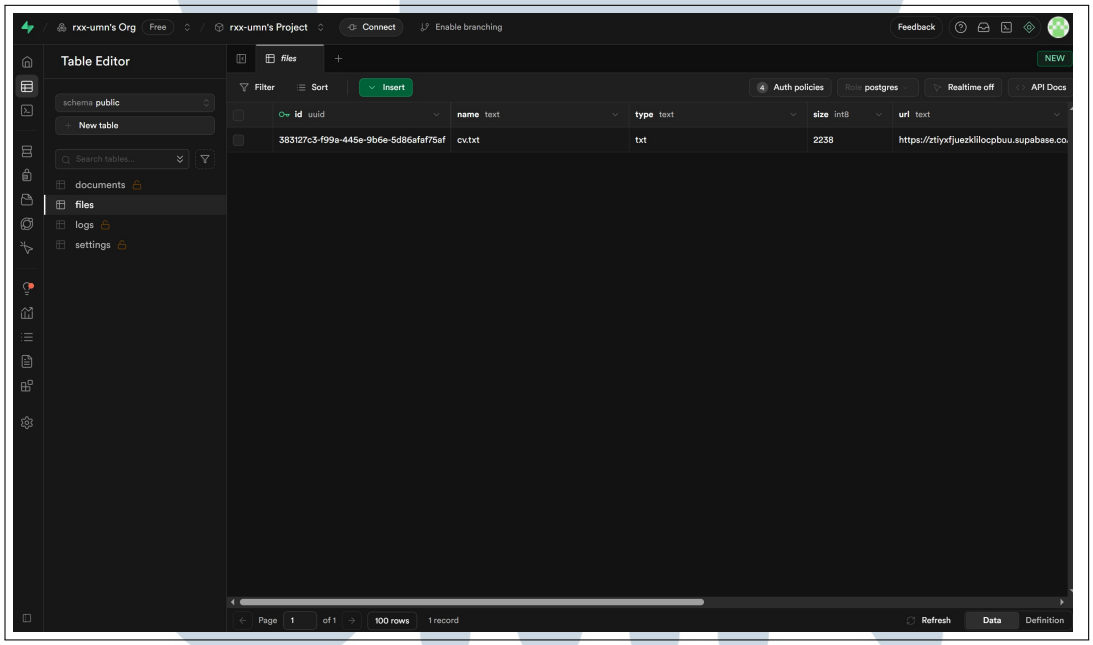
Gambar 3.15. Tampilan UI *Login Page*



Gambar 3.16. Tampilan UI *Register Page*

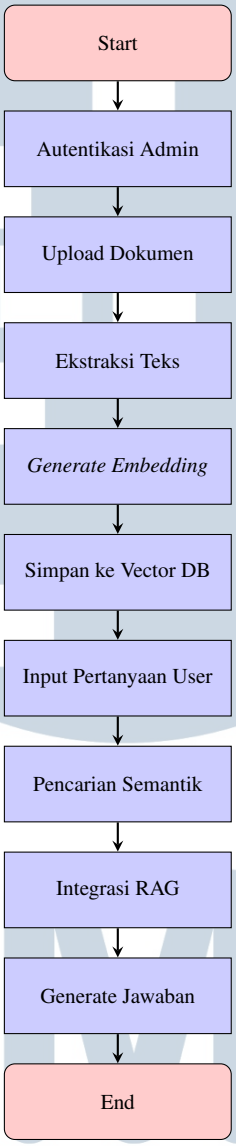


Gambar 3.17. Tampilan UI *Admin Dashboard*



Gambar 3.18. Tampilan Supabase Database

Flowchart



Gambar 3.19. Flowchart Proses Manajemen Dokumen dan Pencarian Berbasis RAG

Kendala

Selama proses pengembangan sistem manajemen dokumen, sejumlah kendala teknis ditemui. Pertama, pada tahap autentikasi admin, implementasi masih menggunakan akun *dummy* karena berdasarkan kebijakan perusahaan, sistem login harus menggunakan Keycloak yang sepenuhnya dikendalikan oleh tim infrastruktur internal, sehingga pengembang tidak memiliki akses langsung terhadap proses autentikasi sebenarnya. Kedua, saat proses ekstraksi dan upload

dokumen, terjadi penurunan performa pada klien ketika menangani file berukuran besar. Ketiga, dalam penyimpanan ke Supabase Storage dan database, ditemukan beberapa kendala seperti duplikasi nama file dan sinkronisasi metadata yang tidak selalu konsisten. Keempat, fungsi pencarian semantik berbasis embedding vector memerlukan penyesuaian parameter *similarity threshold* agar hasil tetap relevan. Terakhir, pada tahap integrasi sistem RAG, ditemukan tantangan dalam manajemen vector store serta pemetaan metadata yang belum sepenuhnya otomatis.

Solusi

Untuk mengatasi kendala tersebut, diterapkan solusi teknis yang terstruktur. Proses autentikasi tetap dikembangkan menggunakan `NextAuth.js` dan `CredentialsProvider` sebagai simulasi, sambil menunggu integrasi resmi dengan Keycloak dari pihak perusahaan. Untuk menangani file besar, dilakukan optimalisasi proses ekstraksi di sisi klien dan validasi ukuran file sebelum upload. Pada penyimpanan di Supabase, digunakan penamaan file yang unik dan mekanisme penanganan error saat penyimpanan metadata. Dalam pencarian semantik, parameter *threshold* dikalibrasi melalui uji coba, serta dilakukan indexing untuk meningkatkan efisiensi. Pada integrasi RAG, digunakan `LangChain` dan `SupabaseVectorStore` dengan penyesuaian struktur metadata dan optimasi kueri. Implementasi solusi ini berhasil meningkatkan stabilitas sistem dan efisiensi pengelolaan dokumen internal.

Sprint 8: Pembaruan Antarmuka Mikrofon dan Penambahan Fitur Penghentian Respons Suara

Pada Sprint ke-8, dilakukan pembaruan antarmuka pengguna (UI) yang berfokus pada elemen interaksi suara, khususnya ikon mikrofon dan kontrol terhadap output audio. Pembaruan ini ditujukan untuk meningkatkan kenyamanan dan kejelasan dalam proses komunikasi antara pengguna dan sistem CORA.

Ikon mikrofon diperbarui agar lebih intuitif dan responsif, mencerminkan status sistem secara real-time, seperti saat sedang mendengarkan, memproses, atau tidak aktif. Desain ulang ikon ini juga disesuaikan dengan prinsip user-centered design guna memastikan keterbacaan dan kemudahan penggunaan, terutama bagi pengguna baru.

Selain itu, ditambahkan fitur untuk menghentikan respons suara yang

sedang diputar. Fitur ini memungkinkan pengguna untuk segera menghentikan audio output jika informasi yang diberikan sudah cukup dipahami atau jika terjadi kesalahan dalam respons. Dengan adanya kontrol ini, pengguna memiliki fleksibilitas lebih tinggi dalam mengelola interaksi, sekaligus meminimalisir gangguan yang tidak diinginkan.

Pembaruan pada sprint ini berkontribusi terhadap peningkatan pengalaman pengguna secara keseluruhan, dengan memberikan kontrol yang lebih baik terhadap interaksi berbasis suara dan penyempurnaan visual pada elemen antarmuka.

Proses

1. Pembaruan Ikon Mikrofon

Proses pembaruan ikon mikrofon dilakukan untuk meningkatkan kejelasan status sistem kepada pengguna.

```
<button
  onClick={toggleMicrophone}
  className={isListening ? classes.micButtonStop : classes.
   micButtonStart}
  disabled={isThinking || isSpeaking}
>
  {isListening ? (
    <img src="/stopp.gif" alt="StopMic" className={classes.micIcon
   } />
  ) : (
    <img src="/micc.gif" alt="Mic" className={classes.micIcon} />
  )}
</button>
```

Kode 3.28: Implementasi Ikon Mikrofon (page.tsx)

- Menggunakan dua state berbeda untuk ikon: aktif (`stopp.gif`) dan tidak aktif (`micc.gif`)
- Ikon berubah secara real-time berdasarkan status `isListening`
- Disable button saat sistem sedang memproses (`isThinking`) atau berbicara (`isSpeaking`)

2. Fungsi Penghentian Audio

Implementasi fitur untuk menghentikan respons suara yang sedang diputar.

```
const stopAll = async () => {
  await logToSupabase(
    'system',
    'info',
    'All processes stopped by user',
    {
      isSpeaking,
      isListening,
      isThinking,
      queueLength: audioQueue.current.length
    }
  );

  // Hentikan audio yang sedang diputar
  if (currentAudioRef.current) {
    currentAudioRef.current.pause();
    currentAudioRef.current = null;
  }

  // Hentikan recording jika aktif
  if (isListening) {
    stopListening();
  }

  // Kosongkan antrian audio
  audioQueue.current = [];
  setChatMessages([]);
  setIsSpeaking(false);
  setIsThinking(false);
};
```

Kode 3.29: Fungsi Stop Audio (page.tsx)

- Menghentikan audio yang sedang diputar dengan `pause()`
- Membersihkan antrian audio dan chat messages
- Log aktivitas penghentian ke Supabase untuk pelacakan
- Reset semua state terkait audio

3. Backend Stop Endpoint

Implementasi endpoint untuk menghentikan proses di backend.

```python
@app.route("/stop", methods=["POST"])
def stop_conversation():
    global stop_flag, message_queue

    with stop_lock:
        stop_flag = True
        # Bersihkan antrian pesan
        queue_size_before = message_queue.qsize()
        while not message_queue.empty():
            try:
                message_queue.get_nowait()
            except queue.Empty:
                break
        message_queue = queue.Queue()  # Buat antrian baru

    log_to_supabase("conversation-stop", "success",
                    "Conversation stopped by user", {
        "queue_size_before": queue_size_before
    })
    return jsonify({
        "status": "success",
        "message": "Stop signal received"
    })
```

Kode 3.30: Endpoint Stop (conversation_service.py)

- Menggunakan thread-safe lock untuk mengatur flag stop
- Membersihkan antrian pesan yang sedang diproses
- Log aktivitas penghentian ke Supabase
- Mengembalikan konfirmasi ke frontend

4. Stream Response dengan Kontrol Stop

Modifikasi stream response untuk merespons permintaan stop.

```python
def generate():
    while True:
```

```
        with stop_lock:
            if stop_flag:
                yield "event: stopped\ndata: STOPPED_BY_USER\n\n"
                break

        if not message_queue.empty():
            chatbot_response = message_queue.get()
            response_parts = split_text(chatbot_response)

            for part in response_parts:
                yield f"event: message\ndata: {part}\n\n"
                speech_base64 = tts_use_case.generate_speech(part)
                yield f"event: audio\ndata: {speech_base64}\n\n"

            yield "event: done\ndata: RESPONSE_COMPLETE\n\n"
        else:
            sleep(0.5)

return Response(stream_with_context(generate()),
              content_type="text/event-stream")
```

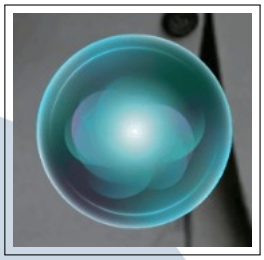
Kode 3.31: Stream Response dengan Stop (conversation_service.py)

- Memeriksa flag stop secara berkala
- Mengirimkan event khusus jika dihentikan oleh pengguna
- Tetap mempertahankan streaming untuk bagian yang sudah diproses
- Menggunakan content type text/event-stream untuk komunikasi real-time

Tampilan *User Interface* (UI)

Berikut ini adalah *screenshoot* dari antarmuka pembaruan ikon mikrofon dan stop respons:

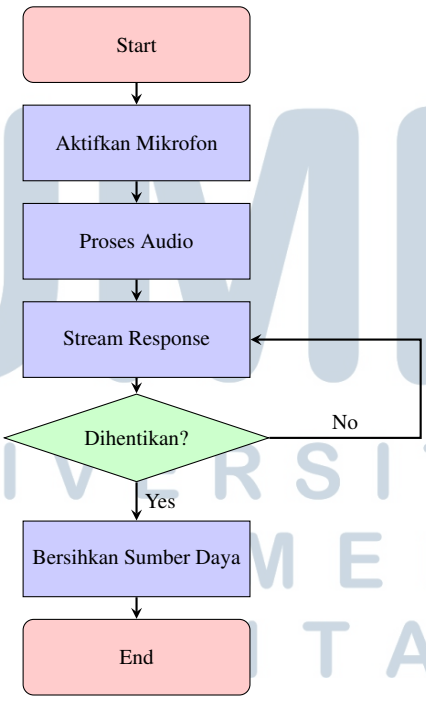
Gambar 3.20. Tampilan UI mikrofon sebelum pembaruan



Gambar 3.21. Tampilan UI mikrofon setelah pembaruan

Gambar 3.22. Tampilan UI Stop Respons

Flowchart



Gambar 3.23. Flowchart Proses Interaksi Suara dengan Kontrol Stop

Kendala

Selama proses pembaruan antarmuka pengguna pada Sprint ke-8, tim menghadapi beberapa tantangan teknis dan desain. Pertama, ikon mikrofon sebelumnya tidak memberikan indikasi status yang cukup jelas, sehingga membingungkan pengguna dalam mengetahui apakah sistem sedang mendengarkan, memproses, atau dalam kondisi tidak aktif. Selain itu, tidak adanya kontrol untuk menghentikan respons suara membuat pengguna merasa kurang memiliki kendali terhadap proses interaksi, terutama ketika terjadi kesalahan output atau informasi yang diberikan sudah cukup.

Dari sisi teknis, tantangan muncul dalam memastikan bahwa perubahan status mikrofon dapat ditampilkan secara real-time tanpa mengganggu proses lain, seperti perekaman suara atau pemutaran audio. Sinkronisasi antara frontend dan backend juga menjadi kendala tersendiri, khususnya saat pengguna menekan tombol stop—sistem harus merespons cepat untuk menghentikan seluruh proses, baik di sisi tampilan, audio, maupun antrian data.

Solusi

Untuk mengatasi kendala tersebut, dilakukan pembaruan ikon mikrofon menggunakan pendekatan berbasis state. Dengan memanfaatkan variabel `isListening`, `isThinking`, dan `isSpeaking`, sistem dapat menampilkan ikon yang sesuai secara dinamis dan real-time. Hal ini meningkatkan transparansi status sistem di mata pengguna.

Fitur penghentian audio juga ditambahkan untuk memberikan pengguna kendali penuh atas interaksi suara. Fungsi `stopAll()` diimplementasikan pada frontend untuk menghentikan audio, rekaman suara, serta mengosongkan antrian audio dan pesan. Di sisi backend, endpoint `/stop` disediakan dengan mekanisme `stop_flag` dan `stop_lock` untuk menghentikan pemrosesan secara aman dan serempak.

Selain itu, proses `stream response` dimodifikasi agar dapat mendeteksi permintaan stop dari pengguna dan mengirimkan event khusus sebagai notifikasi penghentian. Dengan integrasi ini, sistem CORA menjadi lebih responsif, fleksibel, dan ramah pengguna, terutama dalam skenario interaksi berbasis suara.

Sprint 9: Penanganan Gangguan dan Ketahanan Sistem

Pada Sprint ke-9 atau yang terakhir, fokus pengembangan diarahkan pada peningkatan ketahanan sistem terhadap gangguan yang mungkin terjadi selama proses interaksi, seperti koneksi yang tiba-tiba terputus, keterlambatan respons, atau kegagalan pemanggilan API. Perancangan skenario penanganan gangguan dilakukan untuk memastikan bahwa sistem dapat memberikan umpan balik yang sesuai kepada pengguna dan tetap menjaga pengalaman interaksi yang baik meskipun terjadi kendala teknis. Beberapa mekanisme yang dirancang meliputi penanganan error secara terstruktur, notifikasi kegagalan yang ramah pengguna, serta *fallback strategy* untuk melanjutkan proses setelah gangguan terdeteksi. Dengan implementasi ini, sistem CORA menjadi lebih andal dalam kondisi jaringan atau sistem yang tidak ideal, serta mampu memberikan kejelasan status proses kepada pengguna.

Proses

1. Penanganan Error Terstruktur

Implementasi mekanisme penanganan error yang terpusat untuk semua komponen interaksi suara.

```
const showErrorFeedback = async (errorType: 'network' | 'openai' |
    'processing' | 'audio') => {
  await logToSupabase('error-feedback', 'warning', `Showing error
    feedback`, {errorType});

  // Hentikan semua suara sebelumnya
  if ('speechSynthesis' in window) {
    window.speechSynthesis.cancel();
  }

  const messages = {
    network: 'Maaf, saya tidak bisa terhubung ke jaringan...',
    openai: 'Maaf, sedang ada masalah dengan layanan AI...',
    processing: 'Sedang terjadi kendala teknis...',
    audio: 'Ada masalah dengan audio...'
  };

  const message = messages[errorType];
  setErrorFeedback({ message, visible: true });
```

```
18    setLastErrorType(errorType);
19
20    await speakWithBrowser(message, volume);
21
22    // Auto-hide setelah 2 detik
23    setTimeout(() => {
24      setErrorFeedback(null);
25      setLastErrorType(null);
26    }, 2000);
27  };
```

Kode 3.32: Fungsi Error Feedback (page.tsx)

- Kategorisasi error berdasarkan jenis (jaringan, AI, proses, audio)
- Feedback visual melalui komponen Bubble
- Feedback audio melalui TTS browser
- Logging error ke Supabase untuk analisis
- Auto-reset state setelah periode tertentu

2. Fallback Text-to-Speech

Implementasi fallback TTS browser ketika API utama gagal.

```
1  export const speakWithBrowser = (text: string, volume: number) =>
       {
2    if (!('speechSynthesis' in window)) return Promise.resolve(false
       );
3
4    return new Promise((resolve) => {
5      const utterance = new SpeechSynthesisUtterance(text);
6      utterance.volume = volume / 100;
7      utterance.lang = 'id-ID';
8
9      utterance.onend = () => resolve(true);
10     utterance.onerror = (e) => {
11       console.error('TTS Error:', e);
12       resolve(false);
13     };
14
15     window.speechSynthesis.speak(utterance);
```

```
  });
};
```

Kode 3.33: Browser TTS Fallback (browserTTS.ts)

- Deteksi dukungan Web Speech API
- Konfigurasi bahasa Indonesia sebagai default
- Penanganan event `onend` dan `onerror`
- Pengaturan volume sesuai preferensi pengguna
- Return Promise untuk penanganan async

3. Rekoneksi EventSource

Mekanisme otomatis untuk reconnect ketika koneksi terputus.

```
useEffect(() => {
  const initEventSource = () => {
    const eventSource = new EventSource(`/stream_response?voice=${
    voice}`);

    eventSource.onerror = () => {
      console.warn("       Connection error, reconnecting...");
      eventSource.close();
      setTimeout(initEventSource, 3000); // Reconnect after 3s
    };

    return eventSource;
  };

  const es = initEventSource();
  return () => es.close();
}, [voice]);
```

Kode 3.34: EventSource Reconnection (page.tsx)

- Pembuatan koneksi EventSource baru saat terputus
- Delay reconnect 3 detik untuk menghindari spam
- Cleanup saat komponen unmount
- Parameter voice tetap terjaga saat reconnect

4. Penanganan Offline

Deteksi status koneksi dan penanganan yang sesuai.

```
useEffect(() => {
  const handleOnline = () => {
    if (lastErrorType === 'network') {
      window.speechSynthesis.cancel();
      setErrorFeedback(null);
      setLastErrorType(null);

      // Reconnect EventSource
      if (eventSourceRef.current) {
        eventSourceRef.current.close();
        initEventSource();
      }
    }
  };

  window.addEventListener('online', handleOnline);
  return () => window.removeEventListener('online', handleOnline);
}, [lastErrorType]);
```

Kode 3.35: Online/Offline Handler (page.tsx)

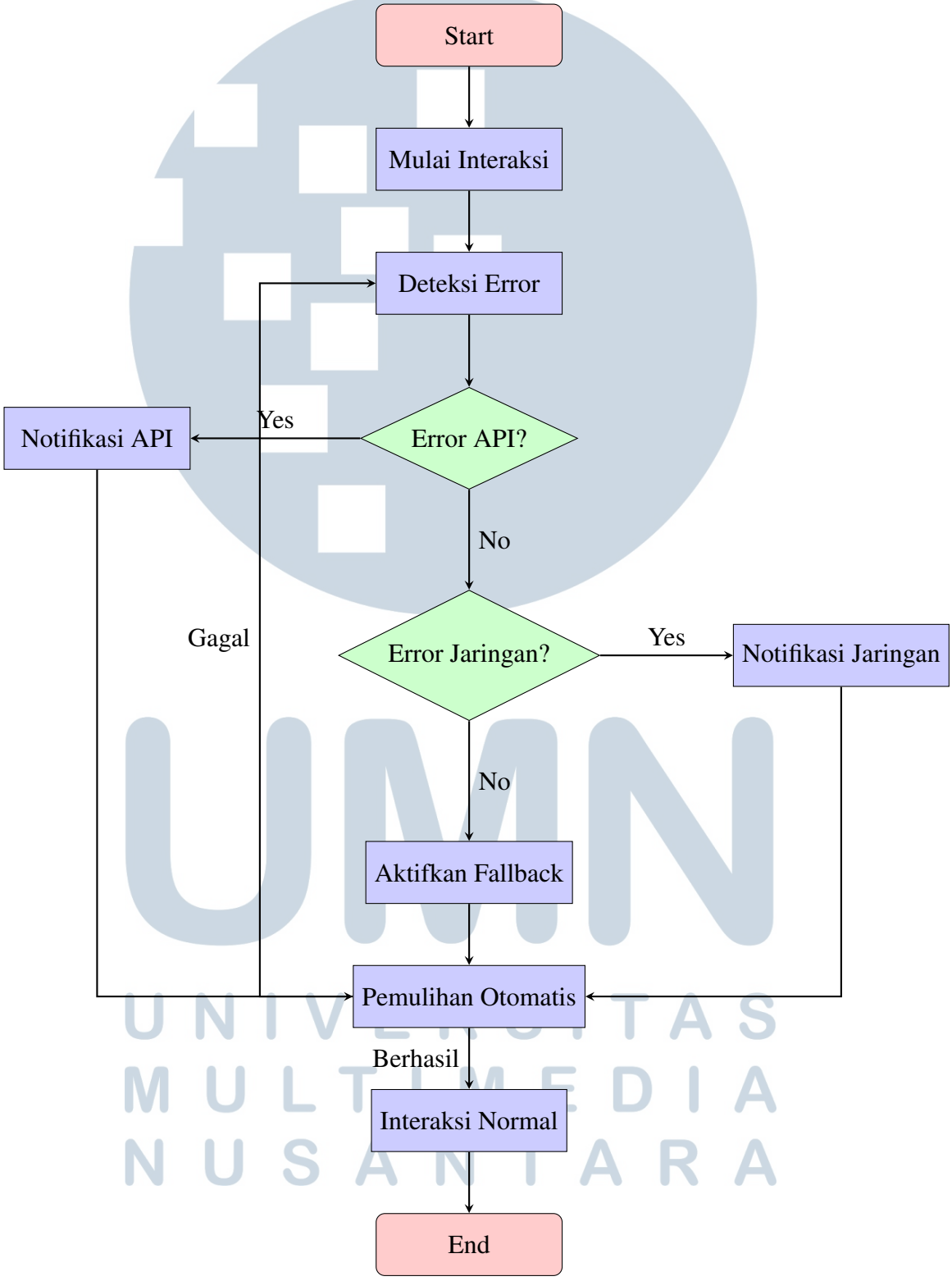
- Listen event `online` browser
- Reset state error jaringan ketika koneksi pulih
- Reconnect otomatis semua layanan
- Bersihkan antrian TTS yang tertunda

Tampilan *User Interface* (UI)

Berikut ini adalah *screenshoot* dari antarmuka ketika terjadi salah satu error:

Gambar 3.24. Tampilan UI Ketika Terjadi Error pada Jaringan

Flowchart



Gambar 3.25. Flowchart Penanganan Gangguan Sistem

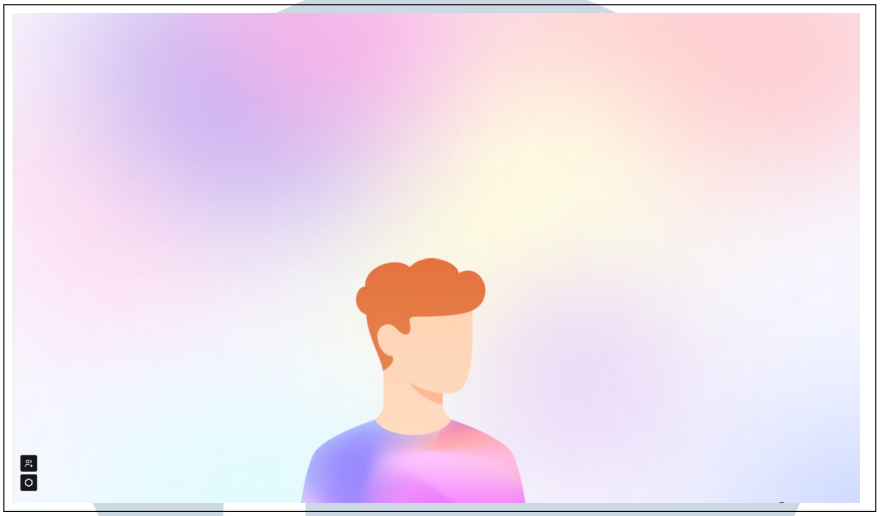
Kendala

Pada Sprint ke-9, terdapat sejumlah kendala teknis yang berkaitan dengan keandalan sistem dalam menangani kondisi gangguan. Permasalahan yang muncul meliputi koneksi jaringan yang tidak stabil, keterlambatan dalam proses respons dari layanan kecerdasan buatan, serta kegagalan dalam pemanggilan API pihak ketiga. Gangguan-gangguan tersebut berdampak pada terganggunya kelancaran interaksi antara pengguna dan sistem, serta berpotensi menurunkan tingkat kepuasan pengguna. Selain itu, pada tahap awal pengembangan, belum tersedia skema penanganan kesalahan (*error handling*) yang terstruktur, serta belum diterapkannya strategi cadangan (*fallback*) ketika terjadi kegagalan dalam pemrosesan.

Solusi

Sebagai solusi terhadap kendala tersebut, dilakukan perancangan dan implementasi mekanisme penanganan gangguan yang komprehensif. Sistem dikembangkan untuk dapat mengidentifikasi berbagai jenis kesalahan, seperti kesalahan jaringan, kegagalan layanan API, maupun kendala pada proses audio. Setiap jenis kesalahan diberikan umpan balik secara spesifik dalam bentuk notifikasi visual dan suara, menggunakan TTS (*Text-to-Speech*) berbasis peramban sebagai alternatif jika layanan utama gagal. Selain itu, diterapkan fitur *auto-reconnect* pada koneksi EventSource untuk memulihkan komunikasi secara otomatis ketika koneksi terputus. Sistem juga mampu mendeteksi status *offline* dan melakukan pemulihan (*recovery*) saat koneksi kembali tersedia. Dengan penerapan solusi-solusi ini, sistem CORA menjadi lebih andal dalam menghadapi kondisi jaringan yang tidak ideal dan tetap mampu menjaga kualitas pengalaman interaksi bagi pengguna.

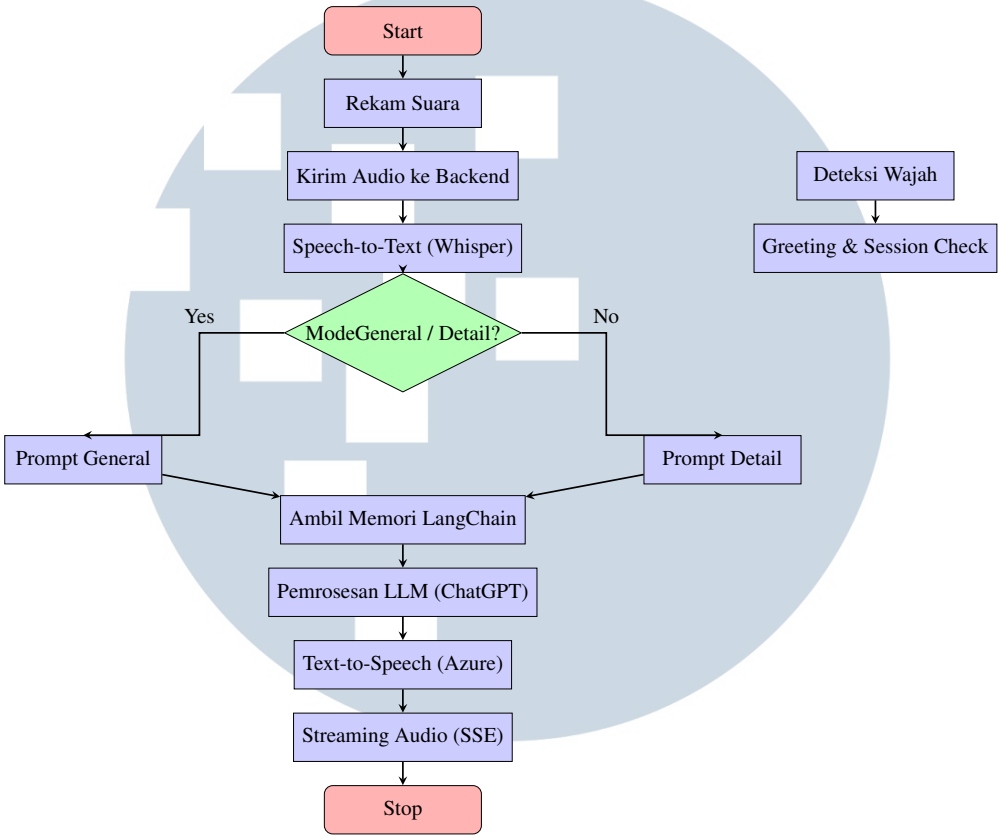
3.4.3 Tampilan CORA Sebelum dan Sesudah Pengembangan



Gambar 3.26. Sebelum Pengembangan

Gambar 3.27. Setelah Pengembangan

3.4.4 Flowchart Keseluruhan Sistem CORA



Gambar 3.28. Flowchart Keseluruhan Sistem CORA

3.5 Tahapan Pengujian Fitur Sistem

Pengujian terhadap fitur-fitur pada sistem CORA dilakukan dengan pendekatan **Black Box Testing**, yang berfokus pada pengujian input dan output dari sistem tanpa memperhatikan struktur internal kode. Proses pengujian dilakukan secara manual menggunakan browser untuk antarmuka frontend dan Postman untuk endpoint backend. Tujuan utama pengujian ini adalah untuk memastikan bahwa setiap fitur bekerja sesuai dengan skenario yang dirancang dan menghasilkan output yang benar sesuai spesifikasi.

Berikut ini adalah beberapa skenario pengujian yang dilakukan terhadap fitur utama sistem.

3.5.1 Pengujian Fitur Login Admin

Tabel 3.10. Test Case: Fitur Login Admin

No	Input	Langkah Uji	Output yang Diharapkan	Hasil
1	Email dan password valid	Submit form login	Redirect ke dashboard admin	Pass
2	Password salah	Submit form login	Muncul pesan error "Login gagal"	Pass
3	Field kosong	Klik tombol login	Validasi muncul di field kosong	Pass

3.5.2 Pengujian Fitur Tanya-Jawab Suara

Tabel 3.11. Test Case: Interaksi Suara

No	Input	Langkah Uji	Output yang Diharapkan	Hasil
1	Suara: "Jam kerja hari ini?"	Tekan mic, ajukan pertanyaan	Respons suara dan teks muncul dengan jawaban benar	Pass
2	Suara tidak jelas / noise	Tekan mic	Sistem menampilkan error atau minta ulangi pertanyaan	Pass

3.5.3 Pengujian Fitur Mode Respons (General / Detail)

Tabel 3.12. Test Case: Mode Jawaban

No	Mode	Pertanyaan	Respons yang Diharapkan	Hasil
1	General	"Apa itu CORA?"	Jawaban singkat (3-5 kalimat)	Pass
2	Detail	"Apa itu CORA?"	Jawaban panjang (6-100 kalimat)	Pass

3.5.4 Pengujian Fitur Repeat Answer

Tabel 3.13. Test Case: Pengulangan Jawaban

No	Input	Langkah Uji	Output yang Diharapkan	Hasil
1	Klik tombol repeat setelah pertanyaan pertama	Ajukan pertanyaan, lalu klik tombol repeat	Jawaban yang sama diulang kembali dalam bentuk suara dan teks	Pass
2	Klik tombol repeat tanpa ada pertanyaan sebelumnya	Klik tombol repeat saat belum ada interaksi	Tidak ada aksi, sistem diam atau tampilkan pesan peringatan	Pass

3.5.5 Pengujian Fitur Pemilihan Bahasa

Tabel 3.14. Test Case: Pemilihan Bahasa

No	Input	Langkah Uji	Output yang Diharapkan	Hasil
1	Pilih bahasa “IDN”	Ajukan pertanyaan setelah pilih bahasa	Jawaban ditampilkan dan diucapkan dalam Bahasa Indonesia	Pass
2	Pilih bahasa “ENG”	Ajukan pertanyaan setelah pilih bahasa	Jawaban ditampilkan dan diucapkan dalam Bahasa Inggris	Pass

3.5.6 Pengujian Fitur Upload File dan Jawaban Dokumen

Tabel 3.15. Test Case: Upload dan Pencarian Dokumen RAG

No	Input	Langkah Uji	Output yang Diharapkan	Hasil
1	Upload file PDF valid	Masuk ke dashboard admin, unggah file PDF	File muncul di daftar, ada notifikasi berhasil	Pass
2	Upload file dengan format tidak didukung	Coba unggah file .exe / .mp3	Sistem menolak dan tampilkan pesan error	Pass
3	Ajukan pertanyaan yang jawabannya ada dalam file	Setelah upload, tanyakan konten dalam file tersebut	Sistem merespons dengan jawaban dari isi file	Pass

3.5.7 Evaluasi Hasil Pengujian

Dari hasil pengujian yang dilakukan terhadap fitur-fitur utama sistem CORA, seluruh skenario berjalan sesuai dengan ekspektasi. Tidak ditemukan error mayor selama proses pengujian. Beberapa error minor seperti keterlambatan respons atau kesalahan pemrosesan suara telah ditangani dengan sistem notifikasi error yang sesuai. Dengan demikian, fitur dapat dinyatakan **siap digunakan** dan memenuhi standar fungsional.