IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Sentiment Analysis dengan Blazor Hybrid dan Azure Open AI – Part II

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com

http://junindar.blogspot.com

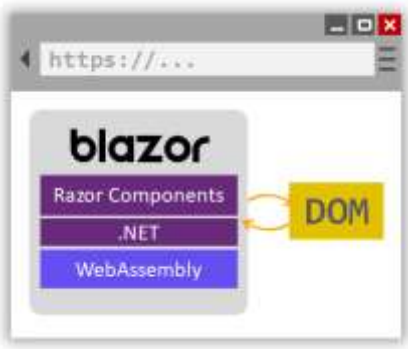
Abstrak

Blazor Hybrid adalah pendekatan inovatif yang menggabungkan framework Blazor dengan .NET MAUI (Multi-platform App UI). Yang memungkinkan kita sebagai *developer* untuk membangun aplikasi *cross platform* menggunakan teknologi web yang sudah dikenal. Dalam aplikasi Blazor Hybrid, komponen Razor berjalan secara native di perangkat dan direnderer ke kontrol Web View yang tertanam melalui local interop. Sehingga komponen ini tidak berjalan di browser dan tidak melibatkan WebAssembly.

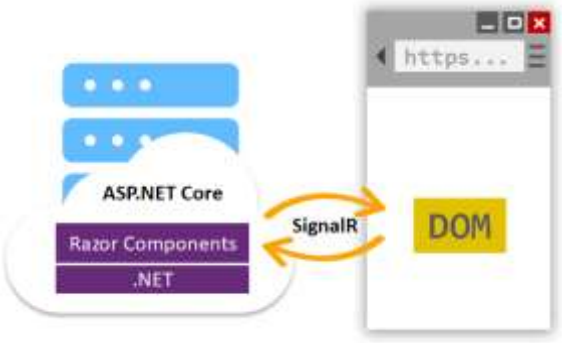
Pendahuluan

Blazor adalah Web Framework yang bersifat Open Source dimana aplikasi Web yang bersifat client-side interactive dapat dikembangkan dengan menggunakan .Net (C#) dan HTML. Pada saat ini C# biasa digunakan untuk melakukan proses back-end dari aplikasi web. Dengan menggunakan fitur baru dari ASP.NET Core yaitu Blazor, kita dapat membangun interactive WEB dengan menggunakan C# dan .NET .

Code .Net berjalan pada Web Assembly, yang artinya kita dapat menjalankan "NET" didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.



Dengan menggunakan Blazor kita dapat memilih apakah menggunakan WebAssemby (Client Side), seperti yang telah dijelaskan di atas atau Blazor yang dijalankan diatas server. Dengan menggunakan cara kedua kita memerlukan SignalR untuk menghubungkan antara client (browser) dan server app. Sebagai contoh jika user melakukan proses klik button pada browser, maka data akan dikirmkan ke Server menggunakan SignalR dan hasilnya akan dikembalikan ke client dengan mengupdate DOM pada client.



Aplikasi Blazor menggunakan runtime .NET yang didasarkan pada Web Assembly atau disingkat WASM. WASM didukung secara native oleh mayoritas browser.

Blazor Hybrid adalah pendekatan inovatif yang menggabungkan framework Blazor dengan .NET MAUI (Multi-platform App UI). Yang memungkinkan kita sebagai *developer* untuk membangun aplikasi *cross platform* menggunakan teknologi web yang sudah dikenal. Dalam aplikasi Blazor Hybrid, komponen Razor berjalan secara native di perangkat dan direnderer ke kontrol Web View yang tertanam melalui local interop. Sehingga komponen ini tidak berjalan di browser dan tidak melibatkan WebAssembly.

Disarankan untuk membaca dan menyeselesaikan latihan pada ebook dari penulis tentang Blazor Hybrid pada tautan berikut. https://tinyurl.com/4t3d9bw6

Chatbot

Chatbot atau chatterbot adalah sebuah layanan obrolan robot/tokoh virtual dengan kecerdasan buatan atau AI (Artificial Intelligent) yang menirukan percakapan manusia melalui pesan suara, obrolan teks ataupun keduanya.

Pada dasarnya bots bekerja dengan cara melihat kata kunci dalam data yang masuk dan membalasnya dengan kata kunci yang paling cocok, atau pola kata-kata yang paling mirip dari basis data tekstual. Artinya, jika pengguna mengirim suatu permintaan maka bots akan membalasnya dengan respon yang spesifik sesuai dengan kata kunci yang dikirim.

Gambar

Sebagai program komputer yang dirancang untuk berinteraksi dengan manusia melalui antarmuka percakapan, seperti pesan teks atau suara. Chat bot menggunakan kecerdasan buatan dan pemrosesan bahasa alami untuk memahami pertanyaan atau perintah yang diberikan oleh pengguna, lalu memberikan respons atau melakukan tindakan yang sesuai berdasarkan pemrograman atau algoritma yang telah ditetapkan.

Chat bot dapat digunakan untuk berbagai tujuan, seperti memberikan informasi, menjawab pertanyaan pengguna, melakukan reservasi, memberikan layanan pelanggan, atau bahkan hanya untuk hiburan. Mereka dapat diimplementasikan dalam berbagai platform, termasuk situs web, aplikasi seluler, layanan pesan instan, dan banyak lagi.

Microsoft Azure

Microsoft Azure adalah platform komputasi awan yang disediakan oleh Microsoft. Ini adalah salah satu penyedia layanan komputasi awan terkemuka di dunia. Azure menyediakan berbagai layanan cloud yang mencakup infrastruktur sebagai layanan

(Infrastructure as a Service/IaaS), platform sebagai layanan (Platform as a Service/PaaS), dan perangkat lunak sebagai layanan (Software as a Service/SaaS).

Azure memungkinkan pengguna untuk menyimpan data, menjalankan aplikasi, dan mengelola sumber daya IT di lingkungan cloud, tanpa perlu membangun dan memelihara infrastruktur fisik sendiri. Platform ini menawarkan berbagai fitur dan layanan termasuk komputasi awan, penyimpanan awan, basis data awan, kecerdasan buatan, Internet of Things (IoT), analisis data, keamanan, dan banyak lagi.

Azure OpenAI Services

Azure OpenAI Services adalah sejumlah layanan yang disediakan oleh Microsoft Azure yang memungkinkan pengembang untuk mengintegrasikan teknologi kecerdasan buatan yang dikembangkan oleh OpenAI ke dalam aplikasi mereka. Ini mencakup layanan seperti Azure Cognitive Services, yang menyediakan berbagai fitur AI dan Machine Learning, serta integrasi khusus dengan model-model canggih yang dikembangkan oleh OpenAI untuk tugas-tugas seperti pemrosesan bahasa alami, pengenalan gambar, dan lainnya.

Dengan menggunakan Azure OpenAI Services, pengembang dapat dengan mudah memanfaatkan kemampuan AI dan Machine Learning tanpa perlu mengembangkan model mereka sendiri dari awal. Ini mempercepat proses pengembangan aplikasi yang cerdas dan membantu dalam menciptakan solusi yang lebih canggih untuk berbagai macam masalah bisnis dan teknis.

Azure OpenAI Service menyediakan akses REST API ke model OpenAI yang powerfull termasuk seri model GPT-4, GPT-4 Turbo dengan Vision, GPT-3.5-Turbo. Selain itu, seri model GPT-4 dan GPT-3.5-Turbo baru kini telah tersedia secara umum. Model ini dapat dengan mudah disesuaikan dengan tugas spesifik namun tidak terbatas pada pembuatan konten, ringkasan, pemahaman gambar, pencarian semantik, dan terjemahan natural language ke kode.

Pada artikel ini tidak menjelaskan apa itu blazor hybrid, bagaimana bekerja dengan project maupun cara-cara untuk menambahkan item pada project, karena semuanya telah dijelaskan baik pada ebook maupun artikel-artikel sebelumnya. Pastikan anda telah

menyelesaikan latihan-latihan pada artikel maupun ebook tersebut. Artikel ini akan fokus membuat chatbot dengan menggunakan Azure Open AI pada Blazor Hybrid.

Sebelum kita mulai dengan membuat project pada Blazor Hybrid, pastikan kita telah memiliki Azure Account terlebih dahulu. Jika belum memiliki Azure Account, kita dapat membuatnya terlebih dahulu disini https://azure.microsoft.com/en-us/free/ .

Sentiment Analysis pada OpenAI

Sentiment Analysis pada OpenAI adalah kemampuan AI untuk memahami dan mengevaluasi emosi yang terkandung dalam sebuah teks. Dengan memanfaatkan Large Language Model / LLM seperti GPT, sistem ini mampu menentukan apakah sebuah pernyataan bersentimen positif, negatif, atau netral. Lebih dari itu, model juga dapat menangkap nuansa emosional yang lebih kompleks, seperti frustrasi, kegembiraan, kekecewaan, kekhawatiran, atau bahkan sarkasme. Hal ini membuat Sentiment Analysis sangat berguna untuk memahami opini dan respons manusia dalam bentuk teks.

Berbeda dengan metode tradisional yang hanya mengandalkan keyword atau rule-based approach, Sentiment Analysis berbasis OpenAI bekerja dengan memahami konteks penuh dari sebuah kalimat. GPT mempertimbangkan struktur kalimat, hubungan antar kalimat, intensitas emosi, serta gaya bahasa, sehingga dapat menafsirkan teks secara lebih natural. Pendekatan ini memungkinkan AI mengenali sentiment bahkan dalam teks yang ambigu, informal, atau panjang dengan berbagai macam perasaan yang bercampur.

Dalam praktiknya, Sentiment Analysis OpenAI banyak digunakan untuk berbagai aplikasi, seperti customer review analysis dan social media monitorings.Kemampuannya untuk menafsirkan emosi secara konsisten menjadikan teknologi ini sangat berharga bagi bisnis, peneliti, dan pengembang perangkat lunak. Dengan memanfaatkan Sentiment Analysis, organisasi dapat memahami perasaan pengguna secara lebih mendalam, memperbaiki layanan, serta mengambil keputusan berbasis data yang lebih tepat.

Setelah berhasil membangun aplikasi sentiment analysis dengan OpenAI untuk menganalisis satu kalimat, kita akan mengembangkan aplikasi ini agar bisa menangani lebih dari satu kalimat atau bahkan file besar, seperti Excel atau CSV, untuk analisis dalam jumlah banyak (bulk analysis). Hal pertama yang perlu kita lakukan adalah memungkinkan aplikasi menerima input teks dalam bentuk yang lebih fleksibel, seperti beberapa kalimat sekaligus. Pengguna bisa memasukkan beberapa kalimat dalam satu kali input, misalnya dengan memisahkan kalimat-kalimat tersebut menggunakan tanda titik atau koma. Setelah teks tersebut dimasukkan, kita bisa memecahnya menjadi kalimat-kalimat terpisah dan mengirimkan setiap kalimat ke API OpenAI untuk dianalisis. Dengan cara ini, setiap kalimat mendapatkan analisis sentimen tersendiri.

Selain itu, kita bisa menambahkan fitur untuk mengunggah file seperti CSV atau Excel, yang berisi banyak teks untuk dianalisis sekaligus. Ini sangat berguna bagi pengguna yang ingin menganalisis data dalam jumlah besar tanpa harus mengetikkan setiap kalimat secara manual. Untuk mengimplementasikan ini, kita perlu menambahkan fungsi upload file pada antarmuka pengguna (UI) aplikasi. Setelah file diunggah, aplikasi akan membaca teks dari file tersebut misalnya, membaca setiap baris atau kolom yang berisi teks dan mengirimkan data tersebut ke OpenAI untuk dianalisis. Setiap entri dalam file akan diproses satu per satu, dan hasil analisis sentimen akan dikembalikan dalam format yang mudah dipahami, seperti tabel atau file baru yang sudah dilengkapi dengan hasil sentimen.

Untuk memproses data dalam jumlah banyak tanpa membebani server atau API OpenAI, kita bisa menggunakan teknik batch processing atau asinkron. Alih-alih mengirimkan permintaan satu per satu, kita bisa mengelompokkan beberapa kalimat atau baris menjadi satu batch dan mengirimkannya sekaligus. Dengan cara ini, kita dapat mengurangi waktu tunggu dan membuat proses lebih efisien, apalagi jika pengguna mengunggah file dengan ribuan baris teks. Setelah proses selesai, aplikasi bisa menampilkan hasil analisis dalam bentuk yang jelas, misalnya dengan menampilkan daftar kalimat bersama sentimen yang terdeteksi, atau memberikan pengguna opsi untuk mengunduh file hasil analisis.

Di sisi antarmuka, kita perlu menyesuaikan desain aplikasi agar lebih user-friendly. Selain tombol "Analyze" untuk menganalisis teks yang dimasukkan langsung, kita bisa menambahkan tombol "Upload File" untuk pengguna yang ingin menganalisis file CSV

atau Excel. Hasil analisis juga bisa ditampilkan dalam format yang mudah dibaca, baik itu berupa tabel interaktif atau file yang bisa diunduh kembali dengan hasil yang sudah diperbarui. Dengan cara ini, aplikasi sentiment analysis yang awalnya hanya untuk satu kalimat kini bisa menangani analisis data dalam jumlah besar secara efisien dan mudah diakses oleh pengguna.

Note : Untuk memudahkan pemahaman terhadap kode atau sintaks yang akan kita gunakan, sangat disarankan untuk membuat proyek baru terlebih dahulu dan menyalin seluruh kode dari artikel sebelumnya. Pastikan proyek tersebut berjalan dengan baik sebelum melanjutkan, agar kita dapat fokus pada penambahan fitur baru tanpa terganggu oleh kesalahan dari konfigurasi sebelumnya.

Pastikan telah membaca dan mengikuti seluruh latihan pada artikel sebelumnya.

https://junindar.blogspot.com/2025/11/sentiment-analysis-dengan-blazor-hybrid.html

Mari kita mulai latihan ini dengan mengikuti langkah-langkah yang akan dijelaskan secara bertahap.

Buat sebuah project Blazor Hybrid dan ikuti langkah-langkah dibawah ini.

- Tambahkan nuget “Microsoft.Extensions.Http“ pada project.
- Buat sebuah folder dengan nama “Models“ dan tambahkan sebuah class dengan nama “ SentimentResult“.

```
public class SentimentResult
{
    public string? Language { get; set; }
    public string Sentiment { get; set; }
    public double Confidence { get; set; }
    public string Explanation { get; set; }
}
```

- Lalu buat folder “Service“ dan tambahkan Interface dan Class, masing-masing bernama “IOpenAiService“ dan “OpenAiService“. Lalu copy sintaks dari latihan sebelumnya. Kita masih menggunakan sintaks-sintaks pada IOpenAiService dan OpenAiService dilatihan sebelumnya.

- Selanjutnya buka file Home.razor, lalu ganti sintaksnya seperti dibawah ini.

```
@page "/"
@using TextAnalyticsMultiple.Models
@using TextAnalyticsMultiple.Service
@using System.Text.RegularExpressions
@inject IOpenAiService OpenAiService
<h3>Bulk Sentiment Analysis</h3>

<textarea @bind="InputText" rows="6" style="width:100%"></textarea>
    <button class="btn btn-primary" @onclick="AnalyzeAll" disabled="@IsProcessing">
        @(IsProcessing ? "Processing..." : "Analyze All")
    </button>

@if (Results.Any())
{
    <table class="table mt-3">
        <thead>
            <tr>
                <th>No</th>
                <th>Text</th>
                <th>Language</th>
                <th>Sentiment</th>
                <th>Confidence</th>
                <th>Explanation</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var r in Results.Select((x, i) => new { x, i }))
            {
                <tr>
                    <td>@(r.i + 1)</td>
                    <td>@r.x.Text</td>
                    <td>@r.x.Result?.Language</td>
                    <td>@r.x.Result?.Sentiment</td>
                    <td>@r.x.Result?.Confidence</td>
                    <td>@r.x.Result?.Explanation</td>
                </tr>
            }
        </tbody>
    </table>
}
```

Potongan sintaks di atas adalah tampilan UI untuk fitur Bulk Sentiment Analysis pada Blazor. Bagian pertama menampilkan sebuah <textarea> yang terhubung dengan variabel **InputText** menggunakan @bind, sehingga setiap teks yang diketik pengguna langsung tersimpan ke variabel tersebut. Lalu ada tombol Analyze All yang menjalankan fungsi **AnalyzeAll** ketika diklik. Tombol ini juga memiliki kondisi disabled="@IsProcessing" sehingga ketika proses sedang berjalan, tombol otomatis tidak bisa ditekan dan teks tombol berubah menjadi "Processing...". Ini membuat pengalaman pengguna lebih nyaman karena mereka tahu bahwa proses sedang berlangsung.

Setelah proses selesai dan data hasil analisis masuk, blok **@if (Results.Any())** akan menampilkan sebuah tabel. Tabel ini berisi daftar hasil analisis untuk setiap teks yang diproses. Pada bagian **foreach**, setiap hasil (**Results**) ditampilkan

bersama nomor urutnya menggunakan **Select((x, i) => ...)**. Tabel ini kemudian menampilkan detail seperti teks asli, bahasa yang terdeteksi, sentimen, tingkat confidence, dan penjelasan (explanation) untuk tiap item. Dengan struktur ini, setiap hasil analisis ditampilkan secara rapi dan mudah dibaca oleh pengguna, sehingga fitur bulk analysis menjadi jauh lebih informatif dan nyaman digunakan.

```
@code {
    bool IsProcessing;
    List<(string Text, SentimentResult? Result)> Results = new();
    string InputText = string.Empty;

    private static List<string> SplitIntoSentences(string input)
    {
        if (string.IsNullOrWhiteSpace(input)) return new List<string>();


        var parts = Regex.Split(input.Trim(), @"(?<=[\.\?\!,\n])\s+")
            .Select(p => p.Trim())
            .Where(p => !string.IsNullOrEmpty(p))
            .ToList();

        if (parts.Count == 1 && parts[0].Length > 200)
        {
            parts = Regex.Split(parts[0], @"[.,]\s*")
                .Select(p => p.Trim())
                .Where(p => !string.IsNullOrEmpty(p))
                .ToList();
        }

        return parts;
    }

    async Task AnalyzeAll()
    {
        var sentences = SplitIntoSentences(InputText);
        if (sentences.Count == 0) return;
        IsProcessing = true;
        Results.Clear();
        int processed = 0;
        foreach (var t in sentences)
        {
            var result = await OpenAiService.AnalyzeSentimentMultiLanguageAsync(t);
            Results.Add((t, result));
            StateHasChanged();
            await Task.Delay(500);
        }

        IsProcessing = false;
    }
}
```

Bagian @code ini berisi logika utama untuk memproses bulk sentiment analysis. Pertama ada beberapa variabel: **IsProcessing** untuk menandai apakah proses sedang berjalan, **Results** untuk menyimpan daftar teks beserta hasil analisanya, dan **InputText** sebagai tempat menampung input dari textarea. Lalu ada fungsi **SplitIntoSentences** yang bertugas memecah input panjang menjadi beberapa kalimat. Fungsi ini menggunakan Regex untuk memisahkan teks berdasarkan

tanda titik, tanda tanya, tanda seru, koma, atau baris baru. Setelah dipisah, setiap bagian dibersihkan dari spasi berlebih. Ada juga logika tambahan—jika teks hanya satu bagian tetapi sangat panjang (lebih dari 200 karakter), maka sistem mencoba memecahnya lagi berdasarkan tanda titik atau koma. Tujuannya supaya kalimat panjang yang berisi lebih dari satu ide tetap bisa dianalisis secara terpisah.

Fungsi utama **AnalyzeAll()** akan dijalankan ketika tombol "Analyze All" diklik. Di dalamnya, input pengguna dipecah menjadi kalimat menggunakan fungsi **SplitIntoSentences** yang telah kita buat sebelumnya. Jika tidak ada kalimat, proses dihentikan.

Setelah itu, aplikasi mengatur **IsProcessing = true** dan mengosongkan hasil sebelumnya. Lalu aplikasi memproses setiap kalimat satu per satu: setiap kalimat dikirim ke **OpenAiService.AnalyzeSentimentMultiLanguageAsync(t)**, lalu hasilnya disimpan ke dalam list **Results. StateHasChanged()** dipanggil agar UI langsung memperbarui tampilan setelah setiap hasil masuk. Ada juga Task.Delay(500) untuk memberi jeda sehingga UI terasa lebih natural. Setelah semua kalimat selesai diproses, IsProcessing di-set kembali ke false, menandakan proses sudah selesai dan tombol bisa dipakai lagi.

Lalu setelah selesai dengan Home.razor, kita lakukan register interface dan class diatas pada “MauiProgram.cs”, seperti pada gambar dibawah ini.

```
#if DEBUG
        builder.Services.AddBlazorWebViewDeveloperTools();
        builder.Logging.AddDebug();
#endif
        builder.Services.AddHttpClient<IOpenAiService, OpenAiService>();
```

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.

Setelah tahap sebelumnya berhasil diterapkan, kita dapat melanjutkan dengan membuat halaman untuk fitur mengunggah file seperti CSV atau Excel, yang berisi banyak teks untuk dianalisis sekaligus. Untuk mengimplementasikan ini, kita perlu menambahkan fungsi upload file pada antarmuka pengguna (UI) aplikasi. Setelah file diunggah, aplikasi akan membaca teks dari file tersebut—misalnya, membaca setiap baris atau kolom yang berisi teks—dan mengirimkan data tersebut ke OpenAI untuk dianalisis. Setiap entri dalam file akan diproses satu per satu, dan hasil analisis sentimen akan dikembalikan dalam format yang mudah dipahami, seperti tabel atau file baru yang sudah dilengkapi dengan hasil sentimen.

Ikuti Langkah-langkah dibawah ini.

- Buka file Home.razor, ganti sintaksnya seperti dibawah ini.

```
@page "/"
@using TextAnalyticsMultiple.Models
@using TextAnalyticsMultiple.Service
@inject IOpenAiService OpenAiService
<h3>Bulk Sentiment Analysis</h3>
<InputFile OnChange="LoadFile" />
@if (FileName != null)
{
    <p>📄 Loaded file: <b>@FileName</b> (@Texts.Count rows)</p>
    <button class="btn btn-primary" @onclick="AnalyzeAll" disabled="@IsProcessing">
        @(IsProcessing ? "Processing..." : "Analyze All")
    </button>
}

@if (Results.Any())
{
    <table class="table mt-3">
        <thead>
            <tr>
                <th>No</th>
                <th>Text</th>
                <th>Language</th>
                <th>Sentiment</th>
                <th>Confidence</th>
                <th>Explanation</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var r in Results.Select((x, i) => new { x, i }))
            {
                <tr>
                    <td>@(r.i + 1)</td>
                    <td>@r.x.Text</td>
                    <td>@r.x.Result?.Language</td>
                    <td>@r.x.Result?.Sentiment</td>
                    <td>@r.x.Result?.Confidence</td>
                    <td>@r.x.Result?.Explanation</td>
                </tr>
            }
        </tbody>
    </table>
}
```

Bagian kode di atas adalah UI untuk fitur **Bulk Sentiment Analysis** yang memanfaatkan file upload sebagai sumber data. Pengguna bisa memilih file dengan komponen <InputFile>, dan ketika file dipilih, fungsi **LoadFile** akan dijalankan untuk membaca isinya. Jika file berhasil dimuat, aplikasi menampilkan informasi berupa nama file (FileName) dan jumlah baris (**Texts.Count**). Setelah itu, tombol **Analyze All** muncul untuk memulai proses analisis sentimen secara massal. Tombol ini juga memiliki mekanisme state, jika aplikasi sedang memproses (IsProcessing = true), teks tombol berubah menjadi "Processing..." dan tombol tidak bisa ditekan lagi. Hal ini membantu pengguna memahami bahwa sistem sedang bekerja dan mencegah klik berulang.

Selanjutnya, jika hasil analisis (**Results**) sudah tersedia, sebuah tabel akan ditampilkan untuk memperlihatkan hasilnya secara lengkap. Tabel ini memiliki kolom nomor urut, teks asli, bahasa yang terdeteksi, sentimen, confidence score, dan penjelasan tambahan. Perulangan menggunakan **Results.Select((x, i) => ...)** memberi kita hasil analisis sekaligus nomor indeks, sehingga lebih mudah menampilkan nomor masing-masing baris. Struktur tabel ini membuat hasil analisis terlihat rapi, terorganisir, dan mudah dipahami oleh pengguna, terutama ketika mereka menganalisis banyak data sekaligus dari file CSV atau Excel.

```
@code {
    string? FileName;
    bool IsProcessing;
    List<(string Text, SentimentResult? Result)> Results = new();
    List<string> Texts = new();

    async Task LoadFile(InputFileChangeEventArgs e)
    {
        var file = e.File;
        FileName = file.Name;
        Texts.Clear();


        using var memory = new MemoryStream();
        await file.OpenReadStream(maxAllowedSize: 10 * 1024 * 1024).CopyToAsync(memory);
        memory.Position = 0;

        if (file.Name.EndsWith(".csv", StringComparison.OrdinalIgnoreCase))
        {
            using var reader = new StreamReader(memory);
            while (!reader.EndOfStream)
            {
                var line = await reader.ReadLineAsync();
                if (!string.IsNullOrWhiteSpace(line))
                    Texts.Add(line.Trim());
            }
        }
        else if (file.Name.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
        {
            using var reader = new StreamReader(memory);
            while (!reader.EndOfStream)
            {
                var line = await reader.ReadLineAsync();
                if (!string.IsNullOrWhiteSpace(line))
                    Texts.Add(line.Trim());
            }
        }
```

```
        else if (file.Name.EndsWith(".xlsx", StringComparison.OrdinalIgnoreCase))
        {

            System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.
            Instance);
             memory.Position = 0;

             using var reader = ExcelDataReader.ExcelReaderFactory.CreateReader(memory);
             do
             {
                 while (reader.Read())
                 {
                     if (reader.GetValue(0) != null)
                     {
                         var text = reader.GetValue(0).ToString();
                         if (!string.IsNullOrWhiteSpace(text)) Texts.Add(text.Trim());
                     }
                 }
             } while (reader.NextResult());
        }

        Results.Clear();
        foreach (var t in Texts)
            Results.Add((t, null));

        StateHasChanged();
    }

}
```

Kode di atas berisi logika untuk membaca file yang di-upload dan mengekstrak teks di dalamnya, baik itu file CSV, TXT, maupun Excel (.xlsx). Ketika pengguna memilih file melalui InputFile, fungsi **LoadFile** dipanggil dan mulai mengambil informasi file seperti Name, lalu membuat **MemoryStream** untuk menyimpan isi file. Isi file dibaca menggunakan **OpenReadStream** dengan batas ukuran 10 MB agar aman. Setelah itu, kode memeriksa ekstensi file — jika file CSV atau TXT, isinya dibaca baris per baris menggunakan **StreamReader**, dan setiap baris yang tidak kosong ditambahkan ke list Texts. Untuk file Excel, digunakan library **ExcelDataReader**, dibantu dengan **EncodingProvider** agar encoding file bisa dikenali. Setiap baris dari kolom pertama Excel dibaca, dan jika ada teksnya, dimasukkan ke list **Texts**.

Setelah proses membaca file selesai, bagian **Results.Clear()** menghapus hasil sebelumnya, dan kemudian hasil awal diisi ulang berdasarkan jumlah teks yang ditemukan. Setiap teks dimasukkan sebagai tuple (text, null) yang menandakan bahwa teks sudah siap tetapi belum dianalisis. Pada akhirnya, StateHasChanged() dipanggil untuk memperbarui tampilan UI agar daftar file yang telah dimuat langsung muncul. Dengan logika ini, aplikasi mampu

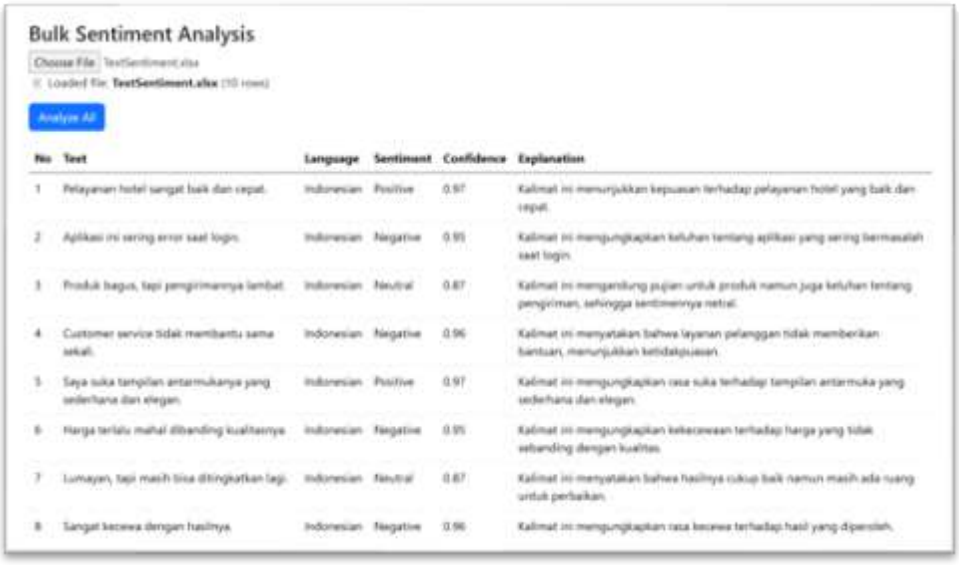
menangani berbagai format file dan menyiapkan data untuk proses bulk sentiment analysis dengan cara yang fleksibel dan efisien.

```
async Task AnalyzeAll()
{
   if (!Texts.Any()) return;
   IsProcessing = true;

   int processed = 0;
   foreach (var t in Texts)
   {
      var result = await OpenAiService.AnalyzeSentimentMultiLanguageAsync(t);
      Results[processed] = (t, result);
      processed++;
      StateHasChanged();
      await Task.Delay(500);
   }
  IsProcessing = false;
}
```

Fungsi **AnalyzeAll()** ini bertugas memproses seluruh teks yang sudah dibaca dari file secara satu per satu. Pertama, fungsi mengecek apakah ada data di **Texts**, kalau kosong, proses langsung berhenti. Setelah itu, **IsProcessing** di-set menjadi true agar UI tahu bahwa analisis sedang berjalan.

Di dalam loop, setiap teks dikirim ke **OpenAiService.AnalyzeSentimentMultiLanguageAsync(t)** untuk dianalisis oleh OpenAI, lalu hasilnya disimpan ke list Results sesuai urutan. Setiap kali satu teks selesai diproses, UI diperbarui dengan **StateHasChanged()**, dan ada jeda **Task.Delay(500)** agar proses terlihat lebih *smooth* dan tidak menekan API terlalu cepat. Setelah semua teks selesai dianalisis, **IsProcessing** dikembalikan ke false sehingga tombol analisis aktif kembali.

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.



Pada latihan diatas, setiap kalimat yang ada pada file akan dikirim satu per satu ke API OpenAI. Artinya, jika file berisi ratusan atau ribuan kalimat, maka akan terjadi ratusan atau ribuan panggilan API secara berurutan. Hal ini tentu mempengaruhi performa aplikasi karena harus menunggu setiap respon dari server OpenAI, dan juga berpotensi meningkatkan biaya penggunaan API secara signifikan. Selain itu, proses yang berulang-ulang ini bisa membuat pengalaman pengguna terasa lambat, apalagi jika menunggu seluruh hasil analisis selesai diproses.

Untuk mengatasi masalah ini, pada latihan selanjutnya kita akan mengubah pendekatan menjadi bulk analysis, yaitu mengirim seluruh kalimat sekaligus dalam satu request ke OpenAI. Caranya adalah dengan mengubah seluruh kalimat yang ada di file menjadi format JSON terlebih dahulu, lalu mengirim JSON tersebut ke API dalam satu kali panggilan. Dengan metode ini, performa aplikasi akan jauh lebih cepat karena hanya ada satu permintaan yang diproses, biaya lebih efisien, dan hasil analisis tetap konsisten karena seluruh data diproses secara bersamaan. Pendekatan ini juga membuat kode lebih rapi dan lebih mudah untuk dikelola.

Ikuti Langkah-langkah dibawah ini.

- Buka file IOpenAiService dan tambahkan sebuah method seperti dibawah ini.

```
Task<List<SentimentResult>> AnalyzeSentimentBatchAsync(List<string> texts);
```

- Lalu lanjutkan dengan membuat implementasi untuk method diatas pada OpenAiService.

```
public async Task<List<SentimentResult>> AnalyzeSentimentBatchAsync(List<string> texts)
{
   var url = $"{_endpoint}openai/deployments/{_deploymentId}/chat/completions?
  api-version={_apiVersion}";

 _httpClient.DefaultRequestHeaders.Clear();
 _httpClient.DefaultRequestHeaders.Add("api-key", _apiKey);
 _httpClient.DefaultRequestHeaders.Accept.Add(new
  MediaTypeWithQualityHeaderValue("application/json"));


 var textsJson = JsonSerializer.Serialize(texts);

 var prompt = $@"
 You are a multilingual sentiment analysis expert.
 Analyze ALL texts in the array below in ONE response.

 Task:
 1. For each text, detect the language automatically
 2. Determine sentiment: Positive, Negative, or Neutral
 3. Estimate confidence score (0 to 1)
 4. Give short explanation in the same language as the text

 Return ONLY a valid JSON array with results for ALL texts, in the same order.

 Example output format:
 [
   {{
     ""Language"": ""Indonesian"",
     ""Sentiment"": ""Negative"",
     ""Confidence"": 0.93,
     ""Explanation"": ""Menunjukkan ketidakpuasan terhadap layanan.""
   }},
   {{
     ""Language"": ""English"",
     ""Sentiment"": ""Positive"",
     ""Confidence"": 0.88,
     ""Explanation"": ""Expresses satisfaction with the product.""
   }}
 ]
 Texts to analyze:
 {textsJson}";
 var requestBody = new
 {
     messages = new[]
     {
     new { role = "system", content = "You are a helpful AI assistant specialized in
     batch sentiment analysis. Always return valid JSON array." },
     new { role = "user", content = prompt }
 },
     temperature = 0.0,
     max_tokens = texts.Count * 150
 };
  var json = JsonSerializer.Serialize(requestBody);
  var content = new StringContent(json, Encoding.UTF8, "application/json");
```

Kode di atas adalah sebuah metode async bernama **AnalyzeSentimentBatchAsync** yang menerima daftar teks, lalu mengirim teks-teks tersebut ke layanan OpenAI. Pertama, metode ini membangun URL endpoint yang berisi alamat dasar, ID deployment, dan versi API. Setelah itu, header HTTP dibersihkan dan diisi ulang dengan api-key serta pengaturan

Accept untuk memastikan request dikirim sebagai JSON. Daftar teks kemudian di-serialize menjadi JSON agar bisa disisipkan ke dalam prompt, di mana prompt tersebut berisi instruksi lengkap untuk model: mendeteksi bahasa, menentukan sentimen, memberi skor kepercayaan, hingga menulis penjelasan singkat dalam bahasa asli teks. Prompt juga meminta model mengembalikan hasil dalam bentuk array JSON yang valid, lengkap dengan contoh format output.

Setelah prompt siap, kode menyiapkan request body berisi struktur pesan bergaya Chat Completion: ada pesan system untuk mendefinisikan perilaku dasar model, serta pesan user yang memasukkan prompt utama. Parameter seperti temperature diset ke 0 agar hasilnya konsisten dan tidak terlalu kreatif, sementara max_tokens disesuaikan dengan jumlah teks supaya respons tidak terpotong. Semua ini kemudian di-serialize menjadi JSON dan dikemas dalam StringContent dengan encoding UTF-8 sebelum dikirim dalam HTTP request. Jadi alurnya simpel: siapkan endpoint → siapkan header → buat prompt → isi pesan → bungkus jadi JSON → siap dikirim ke API untuk dianalisis.

```
    try
    {
        var response = await _httpClient.PostAsync(url, content);

        if (!response.IsSuccessStatusCode)
        {
            var errorText = await response.Content.ReadAsStringAsync();
            return texts.Select(t => new SentimentResult
            {
                Sentiment = "Error",
                Explanation = $"API Error: {errorText}"
            }).ToList();
        }

        using var responseStream = await response.Content.ReadAsStreamAsync();
        using var jsonDoc = await JsonDocument.ParseAsync(responseStream);

        var textOut = jsonDoc.RootElement
            .GetProperty("choices")[0]
            .GetProperty("message")
            .GetProperty("content")
            .GetString()?
            .Trim();

        if (string.IsNullOrWhiteSpace(textOut))
        {
            return texts.Select(t => new SentimentResult
            {
                Sentiment = "Error",
                Explanation = "Empty response from API"
            }).ToList();
        }


        var results = JsonSerializer.Deserialize<List<SentimentResult>>(textOut,
            new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

        return results ?? texts.Select(t => new SentimentResult
        {
            Sentiment = "Error",
            Explanation = "Failed to parse response"
        }).ToList();
    }
    catch (Exception ex)
    {
        return texts.Select(t => new SentimentResult
        {
            Sentiment = "Error",
            Explanation = $"Exception: {ex.Message}"
        }).ToList();
    }
}
```

Bagian kode ini menangani proses pengiriman request ke API dan membaca respons yang diterima. Setelah memanggil **PostAsync**, kode mengecek apakah status respons sukses atau tidak. Kalau gagal, isi error dari server dibaca dan seluruh teks dikembalikan sebagai daftar SentimentResult dengan status "Error". Jika respons berhasil, data JSON dari API dibaca lewat stream lalu diproses dengan **JsonDocument**. Dari struktur JSON OpenAI, kode mengambil bagian **choices[0].message.content**, yang berisi output utama model. Output

tersebut biasanya berupa array JSON dalam bentuk string, jadi diambil sebagai string dan dibersihkan dari spasi tambahan.

Setelah string JSON berhasil didapat, kode mengecek apakah isinya kosong. Kalau kosong, ia mengembalikan daftar error. Jika tidak, string tersebut dicoba di-deserialize menjadi **List<SentimentResult>** dengan pengaturan case-insensitive supaya nama properti dari model tetap bisa dibaca walaupun beda kapital. Jika proses parsing gagal, juga dikembalikan daftar error yang memberi tahu bahwa parsing gagal. Seluruh blok ini dibungkus dalam try-catch, sehingga jika sewaktu-waktu terjadi exception—misalnya error jaringan, respons tidak valid, atau error parsing—kode tetap aman dan mengembalikan daftar hasil dengan status "Error" dan pesan exception. Ini membuat metode lebih tahan banting dan tetap memberikan output yang terstruktur meskipun API gagal atau terjadi masalah tak terduga.

- Lalu buka file Home.Razor dan ganti sintaks pada method "AnalyzeAll" seperti dibawah.

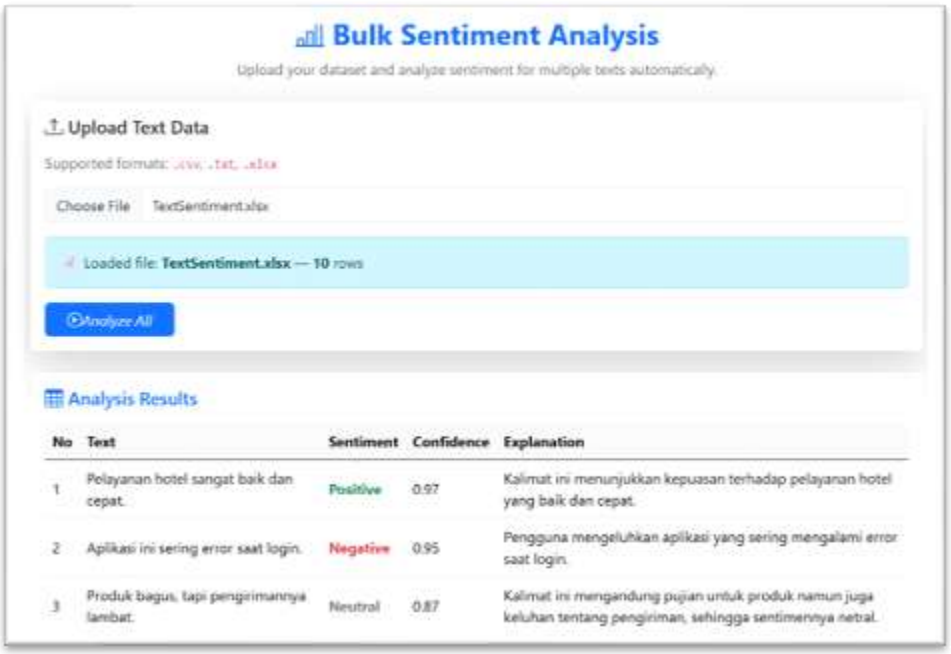
```
async Task AnalyzeAll()
{
    if (!Texts.Any()) return;

    IsProcessing = true;
    Results.Clear();
    StateHasChanged();

    var batchResults = await OpenAiService.AnalyzeSentimentBatchAsync(Texts);

    for (int i = 0; i < Texts.Count; i++)
    {
        var result = i < batchResults.Count ? batchResults[i] : null;
        Results.Add((Texts[i], result));
    }

    IsProcessing = false;
    StateHasChanged();
}
```

Method **AnalyzeAll** ini berfungsi mengatur seluruh alur proses analisis sentimen dalam aplikasi. Pertama, mengecek apakah daftar **Texts** memiliki isi, jika kosong, fungsi langsung berhenti. Jika ada data, indikator proses diaktifkan dengan **IsProcessing = true**, daftar **Results** dibersihkan, dan **StateHasChanged()** dipanggil untuk memperbarui UI. Setelah itu, method memanggil **AnalyzeSentimentBatchAsync** agar semua teks dianalisis secara batch oleh API. Begitu hasil diterima, kode melakukan loop untuk

mencocokkan setiap teks dengan hasil analisisny dengan tetap menjaga keamanan indeks jika jumlah hasil tidak sesuai. Setiap pasangan (text, result) dimasukkan ke dalam **Results** agar siap ditampilkan. Setelah semuanya selesai, indikator proses dimatikan dan UI diperbarui kembali.

Selanjutnya coba jalankan program dan bedakan performance setelah menggunakan method **AnalyzeSentimentBatchAsync.**

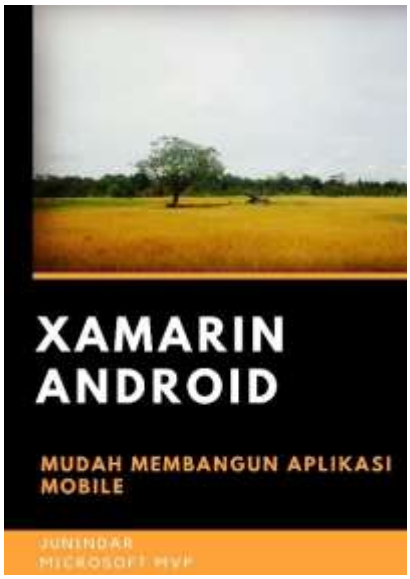
Jika ingin mendapatkan hasil seperti gambar dibawah, agar kelihatan lebih profesional. Sintaksnya dapat dilihat pada project lampiran. Yang banyak berubah adalah dari sisi UI-nya saja. Sedangkan back-end atau C#-nya hanya beberapa bagian saja yang berubah.



Penutup

Sedangkan untuk memudahkan dalam memahami isi artikel, maka penulis juga menyertakan dengan full source code project latihan ini, dan dapat di download disini https://junindar.blogspot.com/2025/11/sentiment-analysis-dengan-blazor-hybrid_24.html

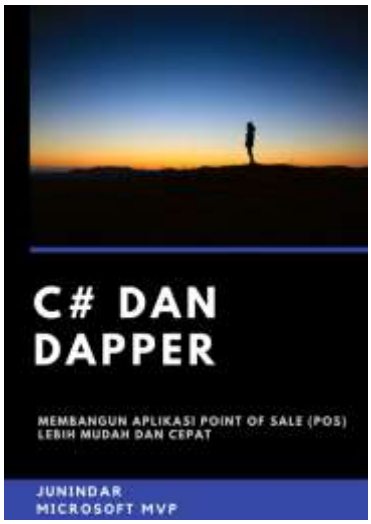
Referensi



https://play.google.com/store/books/details?id=G4tFDgAAQBAJ

https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

https://play.google.com/store/books/details/Junindar_Xamarin_Forms?id=6Wg-DwAAQBAJ



https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ

https://play.google.com/store/books/details/Junindar_ASP_NET_MVC_Membangun_Aplikasi_Web_Lebih?id=XLlyDwAAQBAJ

https://play.google.com/store/books/details/Junindar_ASP_NET_CORE_MVC?id=xEe5DwAAQBAJ

Biografi Penulis.

Junindar Lahir di Tanjung Pinang, 21 Juni 1982. Menyelesaikan Program S1 pada jurusan Teknik Inscreenatika di Sekolah Tinggi Sains dan Teknologi Indonesia (ST-INTEN-Bandung). Junindar mendapatkan Award Microsoft MVP VB pertanggal 1 oktober 2009 hingga saat ini. Senang mengutak-atik computer yang berkaitan dengan bahasa pemrograman. Keahlian, sedikit mengerti beberapa bahasa pemrograman seperti : VB.Net, C#, SharePoint, ASP.NET, VBA. Reporting: Crystal Report dan Report Builder. Database: MS Access, MY SQL dan SQL Server. Simulation / Modeling Packages: Visio Enterprise, Rational Rose dan Power Designer. Dan senang bermain gitar, karena untuk bisa menjadi pemain gitar dan seorang programmer sama-sama membutuhkan seni. Pada saat ini bekerja di salah satu Perusahaan Consulting dan Project Management di Malaysia sebagai Senior Consultant. Memiliki beberapa sertifikasi dari Microsoft yaitu Microsoft Certified Professional Developer (MCPD – SharePoint 2010), MOS (Microsoft Office Specialist) dan MCT (Microsoft Certified Trainer) Mempunyai moto hidup: "**Jauh lebih baik menjadi Orang Bodoh yang giat belajar, dari pada orang Pintar yang tidak pernah mengimplementasikan ilmunya".**