IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

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

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com

http://junindar.blogspot.com

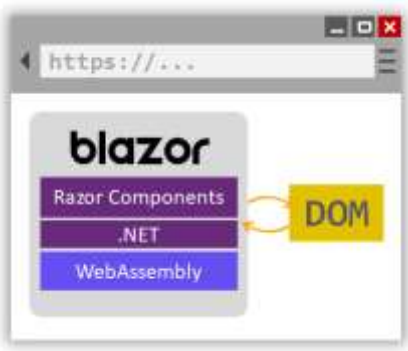
Abstrak

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.

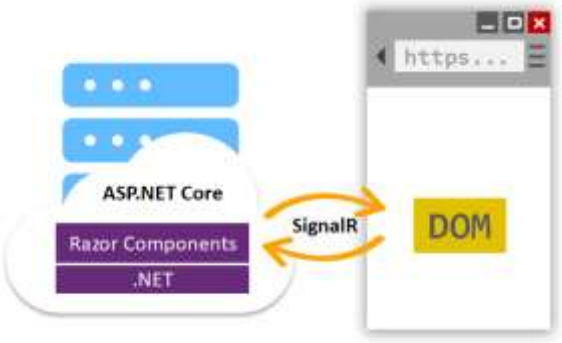
Pendahuluan

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.

Sentiment Analysis menggunakan Azure Cognitive Services

Sentiment Analysis Azure Cognitive Services: Layanan ini dirancang khusus untuk menganalisis sentimen dalam teks. Fungsi utamanya adalah menilai apakah teks mengandung emosi atau opini positif, negatif, atau netral. Analisis sentimen ini

menggunakan algoritma machine learning yang sudah dilatih dengan data tertentu untuk mengidentifikasi dan mengkategorikan emosi dalam kalimat atau dokumen. Layanan ini sangat cocok untuk aplikasi yang membutuhkan pemahaman cepat mengenai perasaan pengguna atau pelanggan, seperti analisis ulasan produk, feedback pelanggan, atau monitoring media sosial.

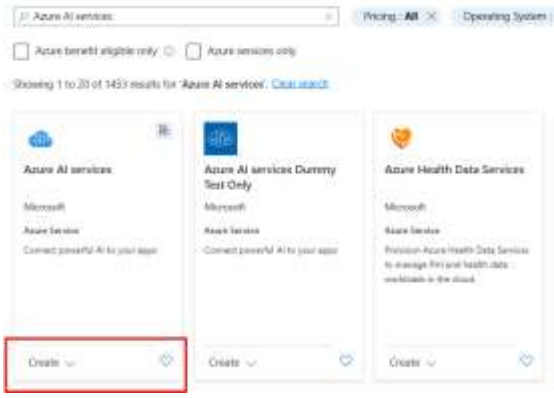
Kemampuan utamanya adalah menilai perasaan atau opini yang terkandung dalam teks secara otomatis, dan memberikan skor sentimen serta label (positif, negatif, netral). Ini lebih terstruktur dan terfokus pada pengklasifikasian sentimen, serta analisis konteks dasar. Meskipun sangat akurat dalam analisis sentimen, layanan ini tidak dapat melakukan tugas-tugas yang lebih kompleks seperti percakapan atau generte teks berbasis konteks yang lebih dalam.

Layanan ini sangat mudah diintegrasikan ke dalam aplikasi yang memerlukan analisis sentimen secara otomatis, misalnya dalam sistem feedback pelanggan atau analisis sosial media. Pengguna cukup mengirimkan teks untuk dianalisis, dan hasilnya berupa label atau skor sentimen. API ini lebih sederhana dan lebih fokus pada satu tugas spesifik, yaitu analisis sentimen.

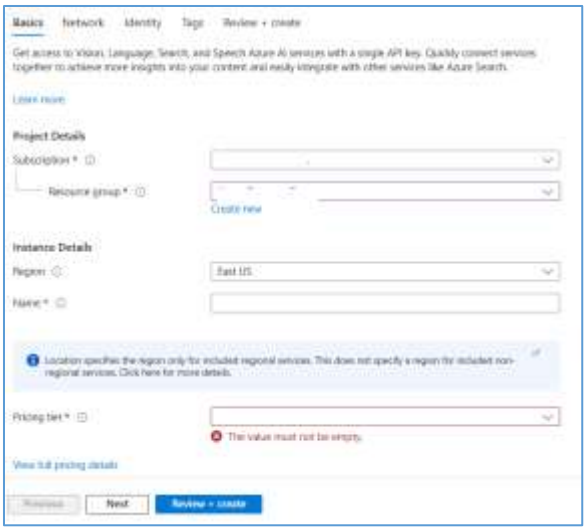
Sebelum kita mulai dengan membuat project pada Blazor Hybrid, pastikan kita telah memiliki Azure Account terlebih dahulu. Jika belum memiliki Azure Account, kita dapat membuatnya terlebih dahulu disini https://azure.microsoft.com/en-us/free/ .

Setelah selesai dengan langkah-langkah diatas, kita lanjutkan dengan membuat “Azure AI services multi-service account“.

- Login pada Azure Portal : https://portal.azure.com/
- Pilih “Create a resource“ dan cari “Azure AI Services“.



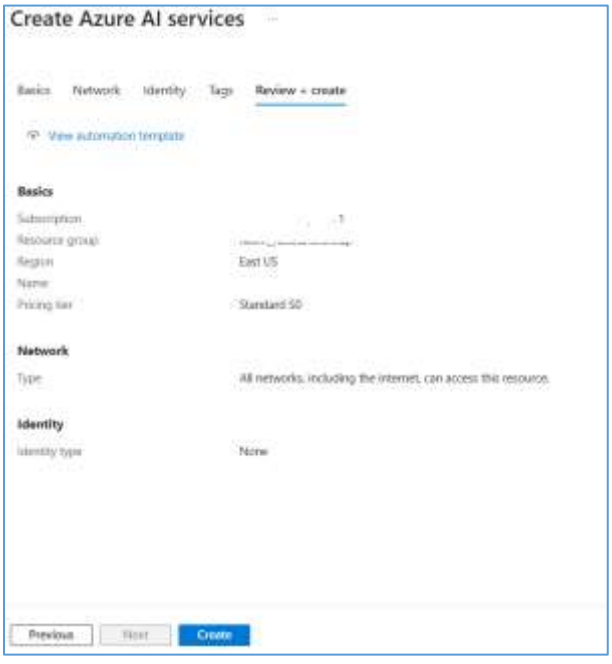
- Lalu klik “Create“ pada Azure AI services yang ditampilkan pada hasil pencarian.



Gambar

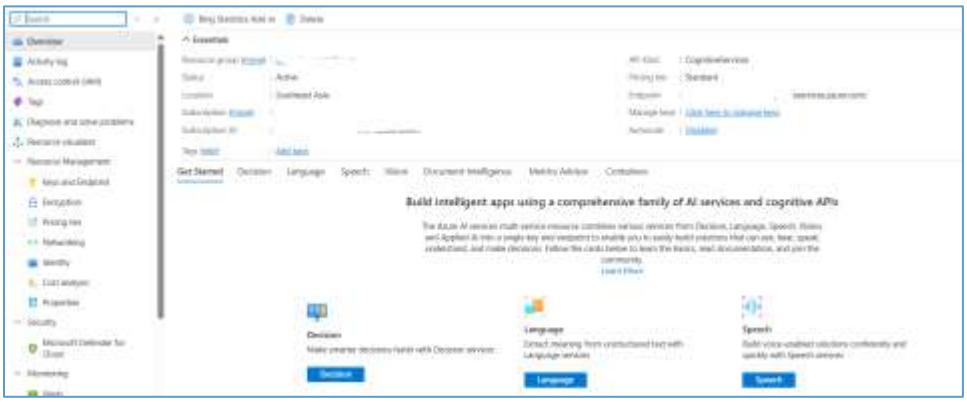
Pada halaman “Create Azure AI services“ tab Basic, masukkan informasi yang diperlukan.

- Lalu pada Tab “Network“ pilih “***All networks, including the internet, can access this resource.***“ Dan klik Next button.
- Dilanjutkan dengan klik “Review+create“ button.

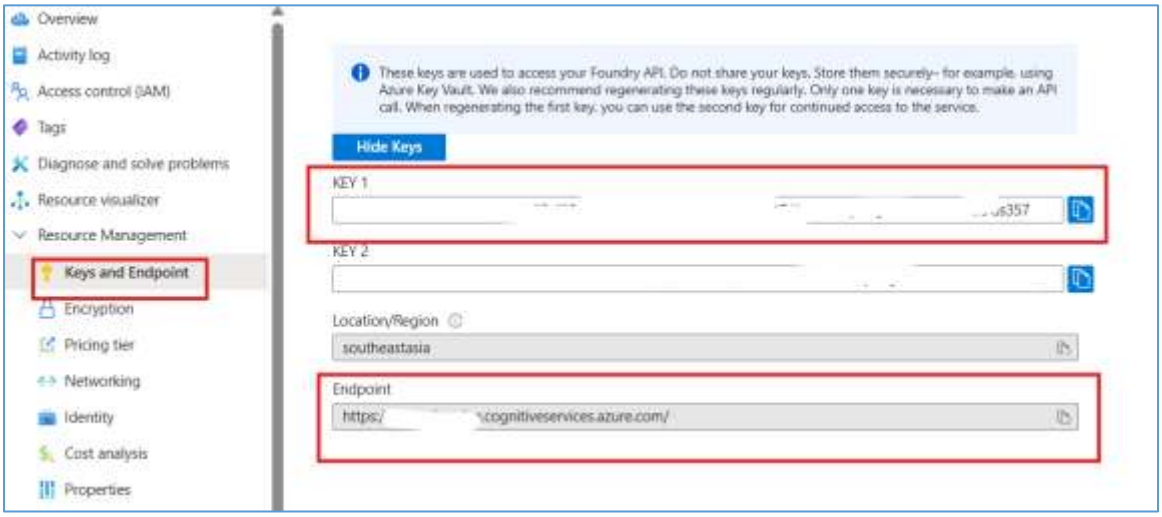


Gambar

- Jika semua informasi sudah benar klik button “Create“. Lalu tunggu hingga ada notifikasi jika resource yang dibuat telah selesai. Hasilnya seperti pada gambar dibawah ini.



Untuk mendapatkan key dan url (endpoint) cognitive services, klik “Keys and Endpoint“ pada “Resource Management“. Key dan url (endpoint) akan digunakan pada Blazor.



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 & https://junindar.blogspot.com/2025/11/sentiment-analysis-dengan-blazor-hybrid_24.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 " Azure.AI.TextAnalytics" pada project.

  NuGet Azure.AI.TextAnalytics adalah library resmi dari Microsoft Azure yang menyediakan antarmuka untuk mengakses layanan Azure Cognitive Services – Text Analytics. Library ini memungkinkan developer dengan mudah menambahkan kemampuan pemrosesan bahasa alami (NLP) ke aplikasi, seperti analisis sentimen, ekstraksi frasa kunci, deteksi bahasa, pengenalan entitas (NER), serta fitur lanjutan seperti analisis opini dan deteksi informasi sensitif. Dengan menggunakan SDK ini, kita dapat memanggil API Text Analytics secara efisien, mengelola kredensial, serta memperoleh hasil yang terstruktur tanpa harus menangani request HTTP secara manual.

- Buat sebuah folder dengan nama "Models" dan tambahkan sebuah class dengan nama " SentimentResult".

```
public record SentimentResultDto
{
     public string Sentiment { get; init; } = "";
     public double PositiveScore { get; init; }
     public double NeutralScore { get; init; }
     public double NegativeScore { get; init; }
     public double Confidence { get; set; }
     public string Explanation { get; init; } = "";
 }
```

- Lalu buat folder "Service" dan tambahkan Interface dan Class, masing-masing bernama " ITextAnalyticsService" dan " TextAnalyticsService". Berikut sintaks detail dari dua file tersebut.

  Interface

```
public interface ITextAnalyticsService
{
     Task<SentimentResultDto> AnalyzeSentimentAsync(string text, string language = "id");
}
```

Class

```
public class TextAnalyticsService : ITextAnalyticsService
{
      private readonly TextAnalyticsClient _client;

      public TextAnalyticsService()
      {
          _client = new TextAnalyticsClient(
              new Uri("https://xxx.cognitiveservices.azure.com/"),
              new AzureKeyCredential("xxxxxxxxxxxxxxx")
          );
      }

      public async Task<SentimentResultDto> AnalyzeSentimentAsync(string text,
       string language = "id")
      {
          var options = new AnalyzeSentimentOptions
          {
              IncludeOpinionMining = true
          };

          var response = await _client.AnalyzeSentimentAsync(text, language, options);
          var doc = response.Value;

          var result = new SentimentResultDto
          {
              Sentiment = doc.Sentiment.ToString(),
              PositiveScore = doc.ConfidenceScores.Positive,
              NeutralScore = doc.ConfidenceScores.Neutral,
              NegativeScore = doc.ConfidenceScores.Negative,
              Confidence = Math.Max(doc.ConfidenceScores.Positive,
              Math.Max(doc.ConfidenceScores.Neutral, doc.ConfidenceScores.Negative)),
              Explanation = GenerateExplanation(doc)
          };

          return result;
      }
}
```

Pada sintaks di atas, **TextAnalyticsService** berfungsi sebagai service yang menjadi perantara antara aplikasi blazor dan Azure Text Analytics. Di dalam constructor, terdapat instance **TextAnalyticsClient** dengan memberikan **endpoint** dan **AzureKeyCredential**, agar aplikasi yang kita buat dapat berinteraksi dengan Azure. Dengan cara ini, kita hanya pakai satu instance client yang sudah siap, jadi lebih efisien dan gampang di-manage dibanding bikin klien baru tiap kali akan melakukan analisis.

Di method **AnalyzeSentimentAsync**, aplikasi ngirim teks dan Bahasa yang digunakan ke Azure untuk dianalisis. **AnalyzeSentimentOptions dengan IncludeOpinionMining = true**, digunakan untuk membuat hasil analisis menjadi lebih detail karena Azure bakal memberikan informasi mengenai opini-opini yang mungkin muncul dalam teks. Pada **response.Value** terdapat sentimen keseluruhan ditambah tiga skor confidence: positif, netral, dan negatif. Lalu kita hitung nilai

Confidence dengan memilih skor yang paling tinggi sebagai nilai kepercayaan utama dari analisis tersebut.

Di bagian akhir, semua data yang sudah diproses tadi dimasukan ke dalam **SentimentResultDto**, termasuk nilai sentimen, skor confidence, dan juga **Explanation** yang dibuat melalui method **GenerateExplanation**. Dengan cara ini, service tidak hanya ngambil raw data dari Azure, tapi juga merapikannya jadi output yang siap dipakai oleh UI, API, atau service lain.

```
private string GenerateExplanation(DocumentSentiment doc)
{
  var sb = new System.Text.StringBuilder();

  foreach (var sentence in doc.Sentences)
  {
      foreach (var opinion in sentence.Opinions)
      {
         string aspect = opinion.Target.Text;
         string aspectSentiment = opinion.Target.Sentiment.ToString().ToLower();

         var opinionsText = string.Join(
                     " dan ",
                     opinion.Assessments.Select(a => $"{a.Text}
                     ({a.Sentiment.ToString().ToLower()})"));

          sb.Append(
           $"Bagian mengenai '{aspect}' memiliki sentimen {aspectSentiment}
           karena terdapat opini {opinionsText}. ");
       }
   }

   if (sb.Length == 0)
   {
      sb.Append("Tidak ditemukan aspek atau opini spesifik.
      Sentimen dihitung dari keseluruhan konteks kalimat.");
   }

   return sb.ToString();
}
```

Method **GenerateExplanation** ini bertugas membangun penjelasan yang lebih mudah dipahami dari hasil opinion mining Azure. Dengan StringBuilder, lalu looping setiap kalimat pada **doc.Sentences** dan setiap opinion di dalamnya. Dari setiap opinion, diambil **Target.Text** sebagai aspek yang dibahas, juga **Sentiment**nya, lalu gabung semua assessment yang terkait menggunakan string.Join. Hasilnya dirangkai jadi satu kalimat penjelasan yang menjelaskan aspek apa yang dinilai, sentimennya dan opini-opini kecil apa yang mendukung penilaian tersebut.

Kalau ternyata nggak ada opini atau aspek sama sekali, method ini ngasih teks default supaya hasilnya nggak kosong. Pada akhirnya, method ini mengubah data mentah dari Azure jadi penjelasan yang lebih naratif dan mudah dimengerti.

Lalu buka "MauiProgram.cs" dan tambahkan sintak berikut.

```
builder.Services.AddSingleton<ITextAnalyticsService,TextAnalyticsService>();
```

Sintaks berfunsgi untuk mendaftarkan TextAnalyticsService sebagai service dengan lifetime Singleton di dependency injection. Artinya, hanya akan dibuat satu instance untuk seluruh aplikasi, dan instance itu akan dipakai berulang kali setiap kali service ini dibutuhkan.

Lalu kita lanjutkan dengan membukan file Home.razor, lalu ganti sintaks-nya seperti dibawah ini.

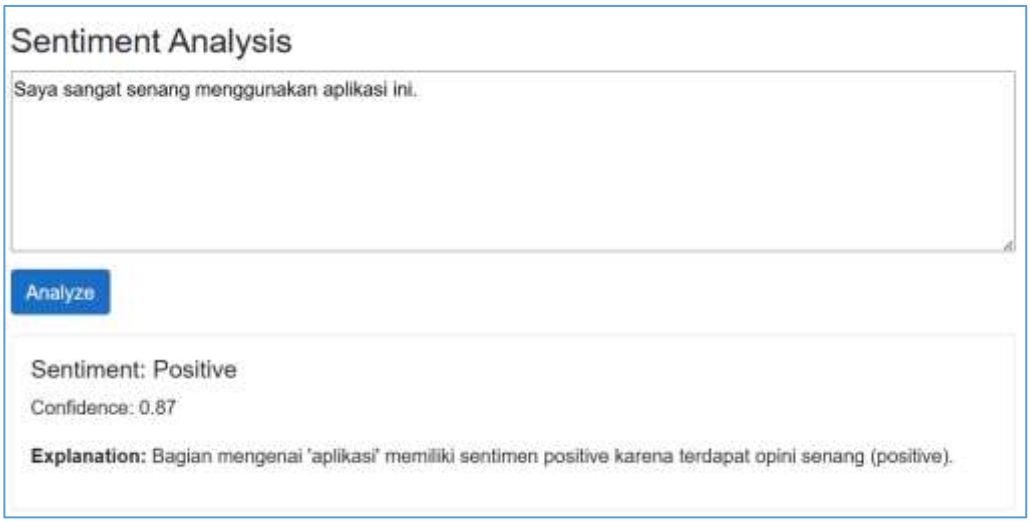
```
@page "/"
@using TextAnalyticsCognitiveServices.Models
@using TextAnalyticsCognitiveServices.Service
@inject ITextAnalyticsService TextService

<h3>Sentiment Analysis</h3>
<textarea @bind="InputText" rows="6" style="width:100%"></textarea>
<div class="mt-2">
    <button class="btn btn-primary" @onclick="Analyze">Analyze</button>
</div>
@if (Result != null)
{
    <div class="card mt-3 p-3">

        <h5>Sentiment: @Result.Sentiment</h5>
        <p>Confidence: @Result.Confidence</p>
        <p><b>Explanation:</b> @Result.Explanation</p>
    </div>
}

@code {
    string InputText { get; set; } = "Saya sangat senang menggunakan aplikasi ini.";
    SentimentResultDto? Result { get; set; }

    async Task Analyze()
    {
        try
        {

            Result = await TextService.AnalyzeSentimentAsync(InputText);
        }
        catch (Exception ex)
        {
            await Shell.Current.DisplayAlert("Error", ex.Message, "OK");
        }

    }

}
```

Sintaks pada Home ini adalah halaman Blazor sederhana yang digunakan untuk melakukan sentiment analysis secara interaktif langsung. Di bagian atas, halaman

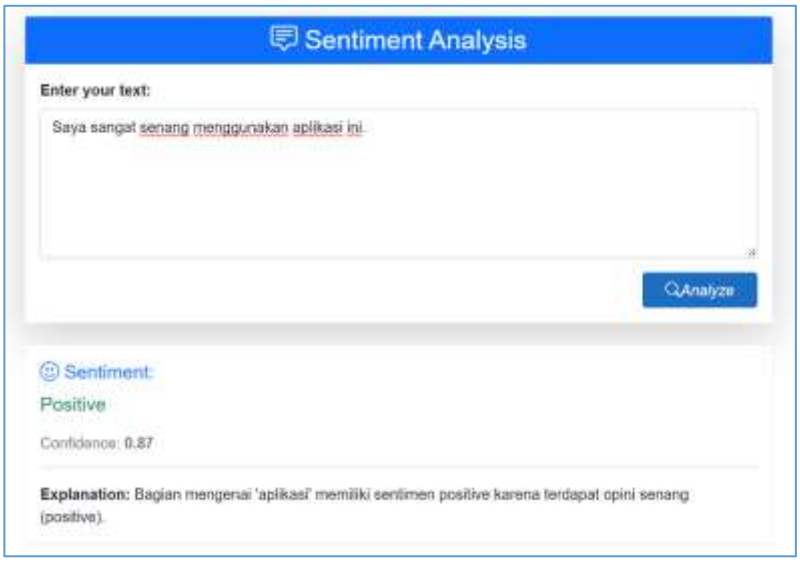
mengimpor namespace untuk model dan service, serta menggunakan dependency injection pada **ItextAnalyticsService** agar metode **AnalyzeSentimentAsync** bisa dipanggil. Pada tampilan UI, terdapat textarea untuk input teks pengguna yang diikat (bind) ke variabel **InputText**, serta sebuah tombol "Analyze" yang akan mengeksekusi method Analyze ketika diklik. Method ini memeriksa apakah InputText tidak kosong, lalu memanggil service OpenAI untuk menganalisis sentimen teks yang dimasukkan, dan menyimpan hasilnya di variabel Result.

Bagian bawah halaman menampilkan hasil analisis jika Result tidak null, dengan informasi seperti Sentiment, Confidence, dan Explanation, semuanya dibungkus dalam card agar mudah dibaca. Dengan struktur ini, pengguna dapat mengetik teks, menekan tombol analisis, dan langsung melihat hasil sentiment analysis secara real-time. Halaman ini memberikan pengalaman interaktif yang sederhana tapi efektif untuk memahami bagaimana AI menilai perasaan atau opini dari teks yang diberikan.

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.



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.



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 ITextAnalyticsService dan tambahkan sebuah method seperti dibawah ini.

```
Task<List<SentimentResultDto>> AnalyzeSentimentBatchAsync(List<string> texts, string language = "id");
```

Lalu lanjutkan dengan membuat implementasi untuk method diatas pada ITextAnalyticsService.

```
public async Task<List<SentimentResultDto>> AnalyzeSentimentBatchAsync(List<string>
texts, string language = "id")
 {
     var options = new AnalyzeSentimentOptions
     {
         IncludeOpinionMining = true
     };


     var response = await _client.AnalyzeSentimentBatchAsync(texts, language, options);
     var documents = response.Value;

     var results = new List<SentimentResultDto>();

     foreach (var doc in documents)
     {
         var result = new SentimentResultDto
         {
             Sentiment = doc.DocumentSentiment.Sentiment.ToString(),
             PositiveScore = doc.DocumentSentiment.ConfidenceScores.Positive,
             NeutralScore = doc.DocumentSentiment.ConfidenceScores.Neutral,
             NegativeScore = doc.DocumentSentiment.ConfidenceScores.Negative,
             Confidence = Math.Max(doc.DocumentSentiment.ConfidenceScores.Positive,
             Math.Max(doc.DocumentSentiment.ConfidenceScores.Neutral,
             doc.DocumentSentiment.ConfidenceScores.Negative)),
             Explanation = GenerateExplanation(doc.DocumentSentiment)
         };

         results.Add(result);
     }

     return results;
 }
```

Method **AnalyzeSentimentBatchAsync** ini digunakan untuk menganalisis beberapa teks dalam satu kali request menggunakan kemampuan batch dari Azure Text Analytics. Prosesnya dimulai dengan membuat opsi analisis yang mengaktifkan IncludeOpinionMining, sehingga layanan Azure tidak hanya memberikan sentimen keseluruhan tetapi juga insight tambahan tentang opini yang muncul dalam teks. Setelah itu, daftar teks dikirim ke Azure melalui **AnalyzeSentimentBatchAsync**, dan hasilnya berupa koleksi dokumen yang masing-masing sudah dianalisis secara individual.

Setiap dokumen hasil analisis kemudian diproses satu per satu. Informasi yang diambil meliputi sentimen utama, tiga jenis confidence score (positive, neutral, negative), serta nilai confidence tertinggi yang dianggap paling representatif. Data-data tersebut kemudian dirangkai menjadi objek **SentimentResultDto** agar hasilnya lebih terstruktur dan mudah digunakan pada lapisan aplikasi lain.

Selain itu, method ini juga menambahkan penjelasan tambahan melalui **GenerateExplanation**, yang menghasilkan narasi sederhana berdasarkan opinion mining dari Azure. Narasi ini membantu menjelaskan alasan suatu teks dinilai positif, netral, atau negatif. Semua hasil akhirnya dikumpulkan ke dalam list dan dikembalikan sebagai output, sehingga bisa langsung dimanfaatkan untuk UI, laporan, atau kebutuhan analisis lanjutan.

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

```
@page "/Multiple"
@using TextAnalyticsCognitiveServices.Models
@using TextAnalyticsCognitiveServices.Service
@inject ITextAnalyticsService TextService
<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>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?.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

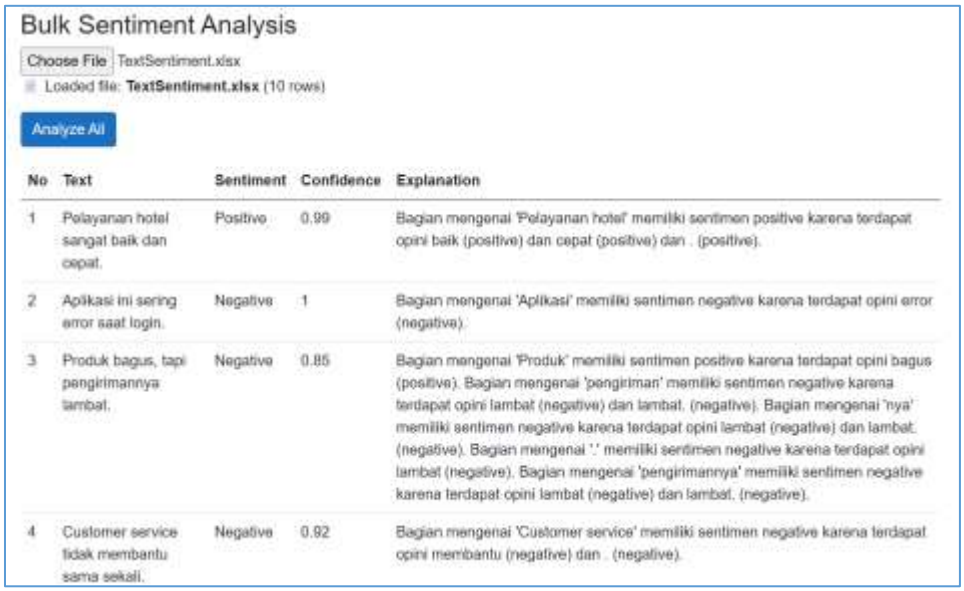
menangani berbagai format file dan menyiapkan data untuk proses bulk sentiment analysis dengan cara yang fleksibel dan efisien.

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

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

     var batchResults = await TextService.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();
}
```

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 **TextService.AnalyzeSentimentBatchAsync(t)** untuk dianalisis oleh Azure AI Service, lalu hasilnya disimpan ke list Results sesuai urutan. Setiap kali satu teks selesai diproses, UI diperbarui dengan **StateHasChanged**. Setelah semua teks selesai dianalisis, **IsProcessing** dikembalikan ke false sehingga tombol analisis aktif kembali.

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.

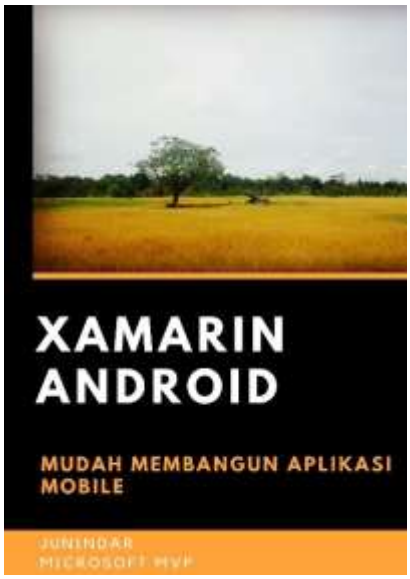


No	Text	Sentiment	Confidence	Explanation
1	Pelayanan hotel sangat baik dan cepat.	Positive	0.99	Bagian mengenai 'Pelayanan hotel' memiliki sentimen positive karena terdapat opini baik (positive) dan cepat (positive) dan . (positive).
2	Aplikasi ini sering error saat login.	Negative	1	Bagian mengenai 'Aplikasi' memiliki sentimen negative karena terdapat opini error (negative).
3	Produk bagus, tapi pengirimannya lambat.	Negative	0.85	Bagian mengenai 'Produk' memiliki sentimen positive karena terdapat opini bagus (positive). Bagian mengenai 'pengiriman' memiliki sentimen negative karena terdapat opini lambat (negative) dan lambat. (negative). Bagian mengenai 'nya' memiliki sentimen negative karena terdapat opini lambat (negative) dan lambat. (negative). Bagian mengenai '.' memiliki sentimen negative karena terdapat opini lambat (negative). Bagian mengenai 'pengirimannya' memiliki sentimen negative karena terdapat opini lambat (negative) dan lambat. (negative).
4	Customer service tidak membantu sama sekali.	Negative	0.92	Bagian mengenai 'Customer service' memiliki sentimen negative karena terdapat opini membantu (negative) dan . (negative).

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/2026/01/sentiment-analysis-dengan-blazor-hybrid.html

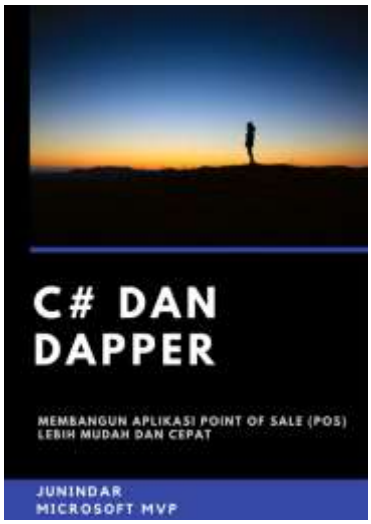
Referensi



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".**