IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Speech to Text dan Text to Speech Dengan Blazor Hybrid

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com

http://junindar.blogspot.com

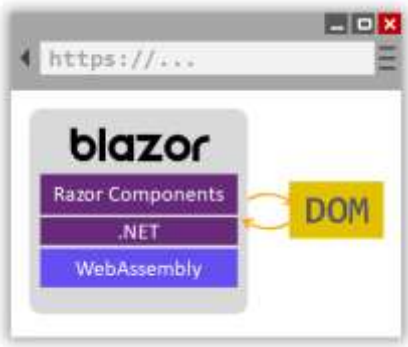
Abstrak

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.

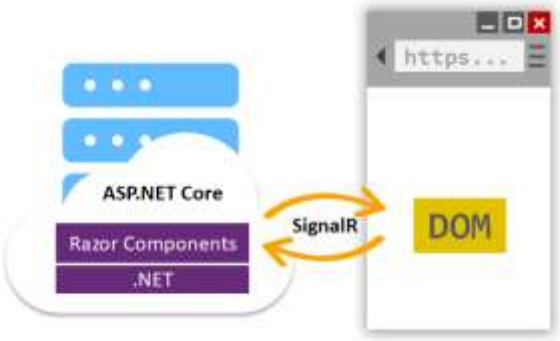
Pendahuluan

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.

Speech to Text dan Text to Speech

Azure Speech to Text adalah layanan kecerdasan buatan dari Microsoft yang berfungsi mengubah suara atau ucapan manusia menjadi teks secara otomatis. Teknologi ini memanfaatkan model AI yang sudah dilatih dengan berbagai bahasa dan aksen, sehingga mampu mengenali percakapan secara cukup akurat, baik dari rekaman audio maupun suara langsung melalui mikrofon. Dalam pengembangan aplikasi modern, layanan ini sering digunakan untuk fitur transkripsi meeting, subtitle otomatis, perintah suara, hingga dokumentasi percakapan tanpa perlu mengetik secara manual.

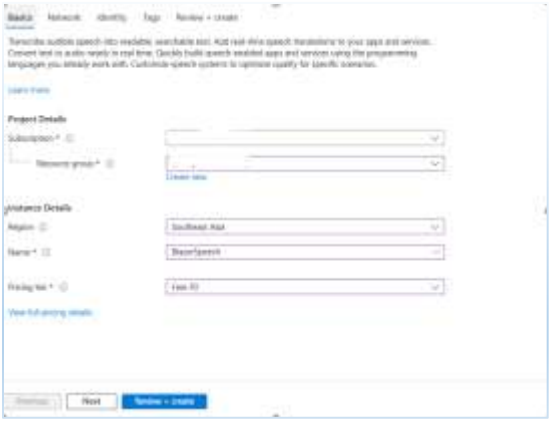
Yang membuat Azure Speech to Text menarik adalah kemudahannya untuk diintegrasikan ke berbagai platform, mulai dari aplikasi web, mobile, hingga desktop. Developer cukup mengirimkan audio ke layanan ini melalui API, lalu hasil teks bisa langsung diproses lebih lanjut, misalnya untuk pencarian, analisis sentimen, atau otomatisasi workflow. Dengan pendekatan berbasis cloud, pengguna juga tidak perlu repot membangun model pengenalan suara sendiri, sehingga proses pengembangan menjadi lebih cepat dan fokus bisa diarahkan ke fitur utama aplikasi.

Sebelum kita mulai mengimplementasikan fitur pada Blazor Hybrid, langkah pertama yang perlu dilakukan adalah menyiapkan resource Speech terlebih dahulu di Azure. Resource ini nantinya akan digunakan sebagai layanan utama untuk menjalankan proses Speech to Text pada aplikasi yang akan kita bangun. Pada bagian berikut, kita akan membuat resource Speech di Azure secara bertahap, mulai dari proses pembuatan hingga konfigurasi dasar yang diperlukan agar layanan dapat digunakan dengan baik di dalam aplikasi.

Setelah selesai dengan langkah-langkah diatas, kita lanjutkan dengan membuat “Speech“.

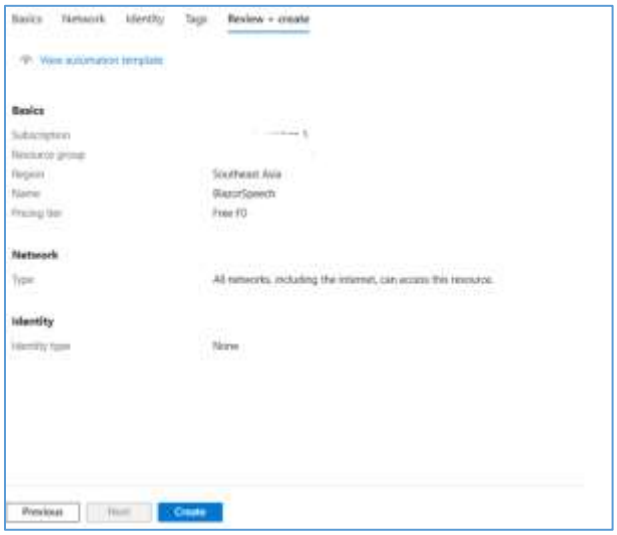
- Login pada Azure Portal : https://portal.azure.com/
- Pilih “Create a resource“ dan cari “Speech“.

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

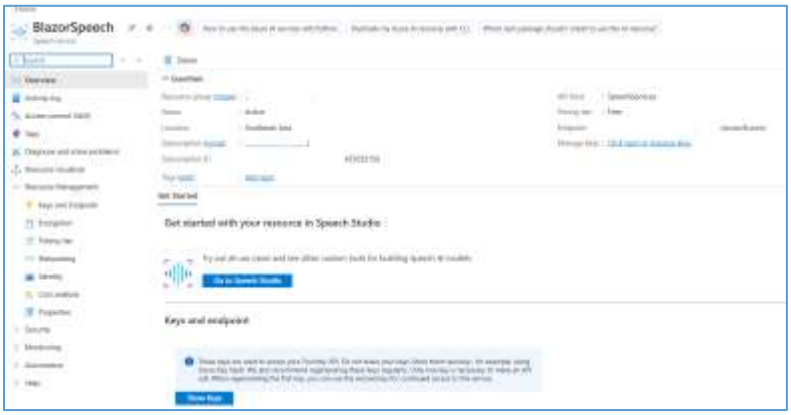


Pada halaman “Create Speech Services“, di tab Basic, masukkan informasi yang diperlukan.

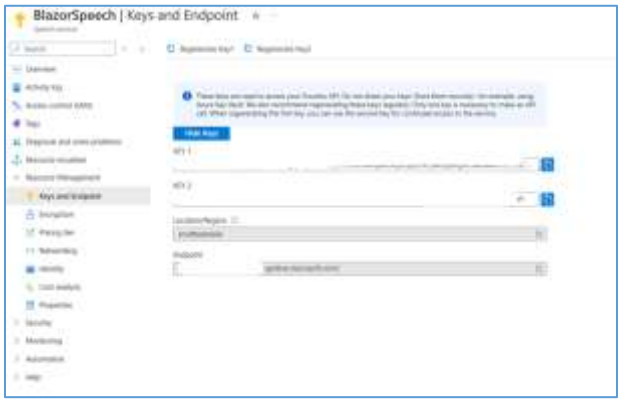
- Lalu pada Tab “Network“ pilih “***All networks, including the internet, can access this resource.***“ Dan klik Next button.
- Dilanjutkan dengan klik “Review+create“ button.



- 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 pembahasan sebelumnya. Pastikan proyek tersebut berjalan dengan baik sebelum melanjutkan, agar kita dapat fokus pada penambahan fitur baru tanpa terganggu oleh kesalahan dari konfigurasi sebelumnya.

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.CognitiveServices.Speech" pada project.

  Nuget Microsoft.CognitiveServices.Speech adalah SDK resmi dari Microsoft yang digunakan untuk memudahkan developer mengintegrasikan layanan Speech AI, seperti Speech to Text, Text to Speech, dan pengenalan suara, ke dalam aplikasi. Dengan library ini, proses komunikasi dengan layanan Azure Speech menjadi jauh lebih sederhana karena developer tidak perlu berinteraksi langsung dengan low level API, sehingga cukup menggunakan class dan method yang sudah disediakan. Library ini juga mendukung berbagai platform seperti .NET, desktop, web, dan mobile, sehingga sangat cocok digunakan pada aplikasi modern yang membutuhkan fitur berbasis suara dengan implementasi yang cepat dan relatif mudah.

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

  Interface

```
public interface IAzureSpeechService
{
  event Action<string> OnRecognizing;
  event Action<string> OnRecognized;
  event Action<string> OnError;

  bool IsListening { get; }
  Task StartContinuousRecognitionAsync();
  Task StopContinuousRecognitionAsync();
  void Dispose();
}
```

Class

```
public class AzureSpeechService :
IAzureSpeechService,
IDisposable
{
  private SpeechRecognizer? _recognizer;
  private readonly string _key = "";
  private readonly string _region = "southeastasia";
  public bool IsListening { get; private set; }
  public event Action<string>? OnRecognizing;
  public event Action<string>? OnRecognized;
  public event Action<string>? OnError;

 public async Task StartContinuousRecognitionAsync()
 {
    if (IsListening) return;

    var config = SpeechConfig.FromSubscription(_key,
    _region);
    config.SpeechRecognitionLanguage = "id-ID";

    _recognizer = new SpeechRecognizer(config);

    _recognizer.Recognizing += (s, e) =>
    OnRecognizing?.Invoke(e.Result.Text);

    _recognizer.Recognized += (s, e) => {
    if (e.Result.Reason ==
    ResultReason.RecognizedSpeech)
    OnRecognized?.Invoke(e.Result.Text);
    };

    _recognizer.Canceled += (s, e) =>
    OnError?.Invoke(e.Reason.ToString());

    await
    _recognizer.StartContinuousRecognitionAsync();
   IsListening = true;
 }
```

Method **StartContinuousRecognitionAsync()** ini berfungsi untuk memulai proses pengenalan suara secara terus-menerus. Karena menggunakan **async Task**, prosesnya berjalan secara **asynchronous**, jadi tidak akan menghambat jalannya program utama saat sedang mendengarkan suara. Di bagian awal dilakukan pengecekan **if (IsListening) return;** yang berguna untuk mencegah proses dijalankan lebih dari satu kali. Jika sistem belum dalam kondisi mendengarkan (**IsListening=False**), maka dibuat konfigurasi menggunakan **SpeechConfig.FromSubscription(_key, _region)** yang berisi kredensial dari speech service. Untuk bahasa kemudian diatur ke "**id-ID**" agar sistem mengenali ucapan dalam Bahasa Indonesia. Setelah itu dibuat objek **SpeechRecognizer** berdasarkan konfigurasi tersebut.

Selanjutnya, beberapa event dipasang untuk menangani berbagai kondisi selama proses pengenalan berlangsung. Event **Recognizing** akan aktif saat suara sedang diproses, biasanya untuk menampilkan teks sementara yang masih bisa berubah. Event **Recognized** akan terpanggil ketika sistem sudah yakin dengan hasil pengenalan suara, dan hanya akan memproses hasil jika statusnya benar-benar **RecognizedSpeech**. Jika terjadi pembatalan atau error, event **Canceled** akan mengirimkan informasi alasan kesalahan melalui **OnError**. Terakhir, method **StartContinuousRecognitionAsync()** dijalankan dengan **await** untuk mulai mendengarkan suara secara berkelanjutan, lalu **IsListening** diubah menjadi true sebagai penanda bahwa sistem sedang aktif mendengarkan.

```
public async Task StopContinuousRecognitionAsync()
 {
     if (_recognizer != null)
     {
        await

_recognizer.StopContinuousRecognitionAsync();
        IsListening = false;
    }
 }

 public void Dispose()
 {
          _recognizer?.Dispose();
 }
}
```

Method **StopContinuousRecognitionAsync()** digunakan untuk menghentikan proses pengenalan suara yang sebelumnya telah dijalankan. Sama seperti method start, method ini juga bersifat asynchronous karena menggunakan async Task. Di dalamnya ada pengecekan **if (_recognizer != null)** untuk memastikan bahwa objek pengenal suara memang sudah dibuat dan siap dihentikan. Jika objek tersebut ada, maka **StopContinuousRecognitionAsync()** dipanggil dengan **await** agar proses penghentian berjalan dengan benar tanpa mengganggu alur program lain. Setelah berhasil dihentikan, properti **IsListening** diubah menjadi false sebagai penanda bahwa sistem sudah tidak lagi dalam kondisi mendengarkan.

Sementara itu, method **Dispose()** berfungsi untuk membersihkan resource yang digunakan oleh **_recognizer**. Pemanggilan **_recognizer?.Dispose();** menggunakan operator null-conditional (?.), artinya Dispose() hanya akan dijalankan jika _recognizer

tidak bernilai null. Ini penting untuk mencegah error sekaligus memastikan resource seperti koneksi atau memori yang digunakan oleh speech recognizer benar-benar dilepaskan. Dengan adanya method ini, pengelolaan resource menjadi lebih rapi dan aman, terutama saat aplikasi ditutup atau fitur pengenalan suara sudah tidak digunakan lagi.

Lalu buka “MauiProgram.cs” dan tambahkan sintak berikut.

```
builder.Services.AddSingleton<IAzureSpeechService,
AzureSpeechService>();
```

Sintaks berfungsi untuk mendaftarkan AzureSpeechService 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 membuka file Home.razor, lalu ganti sintaks-nya seperti dibawah ini.

```
@page "/"
@using SpeechToText.Service
@inject IAzureSpeechService SpeechService
@implements IDisposable

<div class="container mt-4">
    <div class="card shadow">
        <div class="card-header bg-primary text-white">
            <h3>Speech to Text</h3>
        </div>
        <div class="card-body">
            <p>Status: <span class="badge
            @(SpeechService.IsListening ? "bg-danger" :
           "bg-secondary")">@statusText</span></p>

            <textarea class="form-control" rows="8"
            readonly>@fullTranscript</textarea>
            <p class=
            "text-muted italic">@interimText</p>
            <div class="mt-3 d-flex gap-2">
                <button class="btn btn-success"
                @onclick="Start"
                disabled="@SpeechService.IsListening">
                Start</button>
                <button class="btn btn-danger"
                @onclick="Stop"

disabled="@(!SpeechService.IsListening)">
                 Stop</button>
                <button class="btn btn-light"
@onclick='()
                => fullTranscript = ""'>Clear</button>
            </div>
        </div>
    </div>
</div>
```

Sintaks diatas adalah komponen dari halaman (Blazor) untuk fitur Speech to Text. **@inject IAzureSpeechService SpeechService** digunakan untuk menghubungkan halaman dengan service pengenal suara, sementara **@implements IDisposable** menandakan komponen ini mengelola resource yang perlu dibersihkan.

Tersedia tiga tombol: Start untuk mulai merekam, Stop untuk menghentikan, dan Clear untuk mengosongkan hasil transkrip. Tombol Start dan Stop otomatis aktif/nonaktif sesuai status listening..

```
@code {
    private string fullTranscript = "";
    private string interimText = "";
    private string statusText = "Siap";
    protected override void OnInitialized()
    {
        SpeechService.OnRecognizing += HandleRecognizing;
        SpeechService.OnRecognized += HandleRecognized;
        SpeechService.OnError += HandleError;
    }
    private async void HandleRecognizing(string text)
    {
        await InvokeAsync(() => {
            interimText = text;
            StateHasChanged();
        });
    }
    private async void HandleRecognized(string text)
    {
        await InvokeAsync(() => {
            fullTranscript += " " + text;
            interimText = "";
            StateHasChanged();
        });
    }
    private async void HandleError(string error)
    {
        await InvokeAsync(() => {
            statusText = $"Error: {error}";
            StateHasChanged();
        });
    }
    private async Task Start()
    {
        statusText = "Mendengarkan...";
        await
       SpeechService.StartContinuousRecognitionAsync();
    }

    private async Task Stop()
    {
        statusText = "Berhenti";
        await
       SpeechService.StopContinuousRecognitionAsync();
    }

    public void Dispose()
    {

        SpeechService.OnRecognizing -= HandleRecognizing;
        SpeechService.OnRecognized -= HandleRecognized;
        SpeechService.OnError -= HandleError;
    }
}
```

Bagian **@code** ini berisi logika utama yang mengatur bagaimana halaman berinteraksi dengan layanan Speech to Text. Tiga variabel disiapkan untuk mengelola data yang tampil di layer. **fullTranscript** untuk menyimpan seluruh hasil akhir pengenalan suara, **interimText** untuk menampilkan teks sementara saat suara masih diproses, dan **statusText** untuk menunjukkan kondisi aplikasi, misalnya "**Siap**" atau "**Mendengarkan...**". Pada method **OnInitialized()**, komponen mulai mendaftarkan

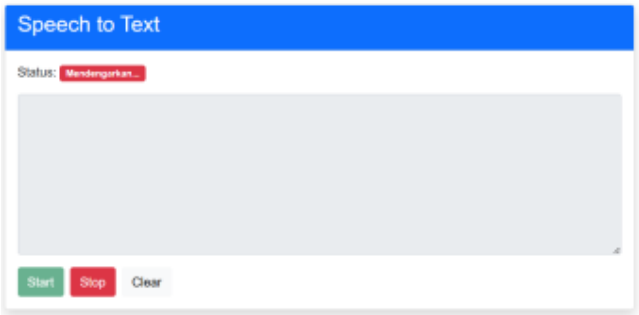
handler ke tiga event dari **SpeechService**, yaitu **OnRecognizing, OnRecognized, dan OnError**. Artinya, setiap kali service mendeteksi suara, menyelesaikan pengenalan, atau mengalami error, komponen ini akan langsung merespons.

Method **HandleRecognizing**, **HandleRecognized**, dan **HandleError** adalah reaksi terhadap event tersebut. Saat suara sedang dikenali, HandleRecognizing akan memperbarui interimText agar teks sementara bisa langsung terlihat oleh pengguna. Ketika hasil sudah final, HandleRecognized akan menambahkan teks tersebut ke fullTranscript lalu mengosongkan teks sementara. Jika terjadi kesalahan, HandleError akan mengubah statusText menjadi pesan error yang sesuai. Semua pembaruan UI dilakukan melalui **InvokeAsync()** dan diikuti **StateHasChanged()** supaya perubahan data langsung tercermin di tampilan tanpa masalah sinkronisasi.

Method **Start()** dan **Stop()** digunakan untuk mengontrol proses pengenalan suara. Saat Start() dipanggil, status diubah menjadi "Mendengarkan..." lalu service dijalankan untuk mulai mengenali suara secara terus-menerus. Sebaliknya, Stop() akan menghentikan proses tersebut dan mengubah status menjadi "Berhenti". Terakhir, method Dispose() berfungsi untuk melepas kembali event handler yang sudah didaftarkan sebelumnya. Ini penting agar tidak terjadi penumpukan event atau memory leak ketika komponen sudah tidak digunakan lagi.

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.

Untuk melakukan pengujian aplikasi ini, klik tombol Start terlebih dahulu, kemudian mulai berbicara menggunakan microphone. Selama Anda berbicara, sistem akan memproses suara dan secara otomatis menampilkan hasilnya dalam bentuk teks pada area textarea, sesuai dengan apa yang diucapkan. Setelah selesai berbicara, klik tombol Stop untuk menghentikan proses pengenalan suara.



Setelah selesai dengan latihan diatas (Text to Speech), maka kita lanjutkan dengan Speech to Text.

Azure Text to Speech adalah layanan dari Microsoft Azure yang memungkinkan kita mengubah teks menjadi suara secara otomatis menggunakan teknologi AI. Dengan layanan ini, teks apa pun, seperti artikel, notifikasi, dialog aplikasi, atau instruksi, bisa dibacakan dalam bentuk suara yang terdengar natural, tidak kaku seperti suara robot. Teknologi ini merupakan bagian dari Azure Cognitive Services (Speech Service) dan mendukung banyak bahasa serta berbagai pilihan karakter suara.

Azure Text to Speech biasanya digunakan pada aplikasi seperti virtual assistant, e-learning, audiobook, sistem navigasi, chatbot, hingga fitur aksesibilitas untuk membantu pengguna dengan keterbatasan penglihatan. Kita juga bisa mengatur gaya bicara, kecepatan, intonasi, bahkan memilih jenis suara (misalnya suara pria atau wanita). Dengan dukungan teknologi neural voice, hasil suaranya terdengar lebih halus, ekspresif, dan mendekati suara manusia asli.

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

- Tambakan sebuah method “SpeakAsync”, pada interface IAzureSpeechService.

```
Task<string?> SpeakAsync(string text, string voiceName);
```

  Lalu buat implementasinya pada class AzureSpeechService seperti dibawah.

```
public async Task<string?> SpeakAsync(string text,
string voiceName)
{
    try
    {
        var config = SpeechConfig.FromSubscription(_key,
        _region);
        config.SpeechSynthesisVoiceName = voiceName;

        using (synthesizer = new
        SpeechSynthesizer(config))
        {
            var result = await
            synthesizer.SpeakTextAsync(text);

            if (result.Reason == ResultReason.Canceled)
            {
                var cancellation =
                SpeechSynthesisCancellationDetails.
                FromResult(result);
                return cancellation.ErrorDetails;
            }
        }

        return null;
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
```

Method **SpeakAsync** ini digunakan untuk mengubah teks menjadi suara menggunakan layanan Azure Text-to-Speech. Method ini bersifat asynchronous dan mengembalikan **Task<string?>**, yang artinya bisa mengembalikan pesan error dalam bentuk string, atau null jika proses berhasil tanpa masalah. Di dalamnya, pertama dibuat konfigurasi menggunakan **SpeechConfig.FromSubscription(_key, _region)** yang berisi kredensial Azure (key dan region). Setelah itu, properti **SpeechSynthesisVoiceName** diatur berdasarkan parameter voiceName, sehingga kita bisa menentukan jenis suara yang ingin digunakan. Objek **SpeechSynthesizer** kemudian dibuat menggunakan konfigurasi tersebut, dan proses pembacaan teks dilakukan melalui **SpeakTextAsync(text)**.

Hasil dari proses tersebut disimpan dalam variabel result. Jika result.Reason menunjukkan status Canceled, artinya terjadi masalah saat sintesis suara berlangsung. Dalam kondisi ini, detail error diambil menggunakan **SpeechSynthesisCancellationDetails.FromResult(result)** lalu dikembalikan sebagai pesan error. Jika tidak ada kendala, method akan mengembalikan null sebagai tanda bahwa proses berjalan sukses. Selain itu, blok try-catch digunakan untuk menangkap

error, misalnya karena koneksi atau konfigurasi yang salah, lalu mengembalikan pesan exception agar bisa ditampilkan atau ditangani oleh pemanggil method ini.

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

```
@page "/Counter"
@using SpeechToText.Service
@inject IAzureSpeechService SpeechService

<PageTitle>Azure Text-to-Speech</PageTitle>

<div class="form-group mb-3">
     <label class="form-label font-weight-
     bold">Masukkan Teks:</label>
     <textarea class="form-control"
      rows="5" @bind="textToSpeak"
      placeholder="Ketik sesuatu di
      sini untuk dibacakan..."></textarea>
            </div>

<div class="mb-3">
   <label class="form-label">Pilih Suara:</label>
   <select class="form-select" @bind="selectedVoice">
   <option value="id-ID-GadisNeural">Gadis (Perempuan –
   Indonesia)</option>
   <option value="id-ID-ArdiNeural">Ardi (Laki-laki –
   Indonesia)</option>
   <option value="en-US-JennyNeural">Jenny (Perempuan –
   English)</option></select>
</div>

<button class="btn btn-primary btn-lg w-100"
     @onclick="SpeakText" disabled="@isProcessing">
      @if (isProcessing)
      {
       <span class="spinner-border spinner-border-
        sm"></span> <span>Memproses...</span>
      }
      else
      {
        <span><i class="oi oi-volume-high"></i>
        Putar Suara</span>
       }
</button>
@if (!string.IsNullOrEmpty(errorMessage))
{
   <div class="alert alert-danger mt-
    3">@errorMessage</div>
 }
```

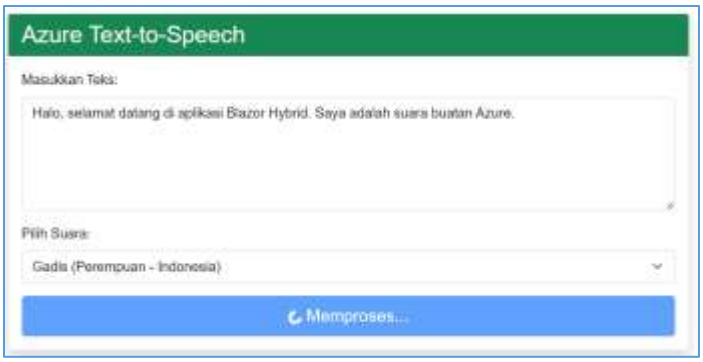
Sintaks ini mengatur alur input teks, pemilihan suara, dan pemutaran audio secara interaktif dan responsif. Pada baris @page "/Counter" menentukan alamat halaman, sedangkan **@inject IAzureSpeechService SpeechService** menghubungkan komponen dengan service Text-to-Speech. textarea yang menggunakan @bind="textToSpeak" akan langsung menyimpan teks yang diketik pengguna, dan dropdown selectedVoice memungkinkan pemilihan jenis suara. Tombol Putar Suara memanggil method SpeakText dan akan nonaktif saat proses berjalan (isProcessing),

sekaligus menampilkan indikator “Memproses...”. Jika terjadi error, pesan akan muncul dalam alert merah.

```
@code {
    private string textToSpeak = "Halo, selamat datang di
    aplikasi Blazor Hybrid. Saya adalah suara buatan
    Azure.";
    private string selectedVoice = "id-ID-GadisNeural";
    private bool isProcessing = false;
    private string errorMessage = "";


    private async Task SpeakText()
    {
        if (string.IsNullOrWhiteSpace(textToSpeak))
         return;

        isProcessing = true;
        errorMessage = "";

        try
        {
            var error = await
            SpeechService.SpeakAsync(textToSpeak,
            selectedVoice);

            if (!string.IsNullOrEmpty(error))
            {
                errorMessage = $"Error: {error}";
            }
        }
        finally
        {
            isProcessing = false;
        }
    }

}
```

Bagian @code ini merupakan back-end yang berisi logika untuk menjalankan fitur Text-to-Speech. Variabel textToSpeak menyimpan teks yang akan dibacakan dan sudah diberi nilai awal sebagai contoh, sedangkan **selectedVoice** menentukan jenis suara yang digunakan. Variabel **isProcessing** berfungsi sebagai penanda apakah proses sedang berjalan (biasanya untuk mengatur tombol agar nonaktif), dan errorMessage digunakan untuk menampilkan pesan kesalahan jika terjadi masalah. Method **SpeakText()** akan dipanggil saat tombol ditekan. Pertama ia mengecek apakah teks kosong, lalu mengaktifkan status proses dan mengosongkan pesan error. Setelah itu, method memanggil SpeechService.SpeakAsync() untuk mengubah teks menjadi suara. Jika service mengembalikan pesan error, maka pesan tersebut ditampilkan. Terakhir, pada blok finally, isProcessing dikembalikan ke false agar status kembali normal setelah proses selesai, baik berhasil maupun gagal.

Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah.




```
@code {
    private string fullTranscript = "";
    private string interimText = "";
    private string statusText = "Siap";
    protected override void OnInitialized()
    {
        SpeechService.OnRecognizing += HandleRecognizing;
        SpeechService.OnRecognized += HandleRecognized;
        SpeechService.OnError += HandleError;
    }
    private async void HandleRecognizing(string text)
    {
        await InvokeAsync(() => {
            interimText = text;
            StateHasChanged();
        });
    }
    private async void HandleRecognized(string text)
    {
        await InvokeAsync(() => {
            fullTranscript += " " + text;
            interimText = "";
            StateHasChanged();
        });
    }
    private async void HandleError(string error)
    {
        await InvokeAsync(() => {
            statusText = $"Error: {error}";
            StateHasChanged();
        });
    }
    private async Task Start()
    {
        statusText = "Mendengarkan...";
        await
       SpeechService.StartContinuousRecognitionAsync();
    }
```

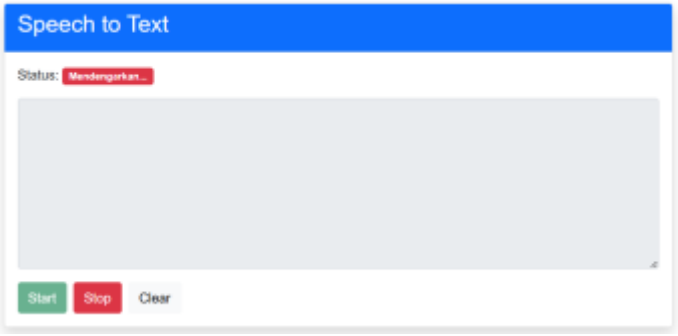
```
    private async Task Stop()
    {
        statusText = "Berhenti";
        await
       SpeechService.StopContinuousRecognitionAsync();
    }

    public void Dispose()
    {

        SpeechService.OnRecognizing -= HandleRecognizing;
        SpeechService.OnRecognized -= HandleRecognized;
        SpeechService.OnError -= HandleError;
    }
}
```

Bagian **@code** ini berisi logika utama yang mengatur bagaimana halaman berinteraksi dengan layanan Speech to Text. Tiga variabel disiapkan untuk mengelola data yang tampil di layer. **fullTranscript** untuk menyimpan seluruh hasil akhir pengenalan suara, **interimText** untuk menampilkan teks sementara saat suara masih diproses, dan **statusText** untuk menunjukkan kondisi aplikasi, misalnya "**Siap**" atau "**Mendengarkan...".** Pada method **OnInitialized()**, komponen mulai mendaftarkan handler ke tiga event dari **SpeechService**, yaitu **OnRecognizing, OnRecognized, dan OnError**. Artinya, setiap kali service mendeteksi suara, menyelesaikan pengenalan, atau mengalami error, komponen ini akan langsung merespons.

Method **HandleRecognizing**, **HandleRecognized**, dan **HandleError** adalah reaksi terhadap event tersebut. Saat suara sedang dikenali, HandleRecognizing akan memperbarui interimText agar teks sementara bisa langsung terlihat oleh pengguna. Ketika hasil sudah final, HandleRecognized akan menambahkan teks tersebut ke fullTranscript lalu mengosongkan teks sementara. Jika terjadi kesalahan, HandleError akan mengubah statusText menjadi pesan error yang sesuai. Semua pembaruan UI dilakukan melalui **InvokeAsync()** dan diikuti **StateHasChanged()** supaya perubahan data langsung tercermin di tampilan tanpa masalah sinkronisasi.

Method **Start()** dan **Stop()** digunakan untuk mengontrol proses pengenalan suara. Saat Start() dipanggil, status diubah menjadi "Mendengarkan..." lalu service dijalankan untuk mulai mengenali suara secara terus-menerus. Sebaliknya, Stop() akan menghentikan proses tersebut dan mengubah status menjadi "Berhenti". Terakhir, method Dispose() berfungsi untuk melepas kembali event handler yang sudah didaftarkan sebelumnya. Ini penting agar tidak terjadi penumpukan event atau memory leak ketika komponen sudah tidak digunakan lagi.

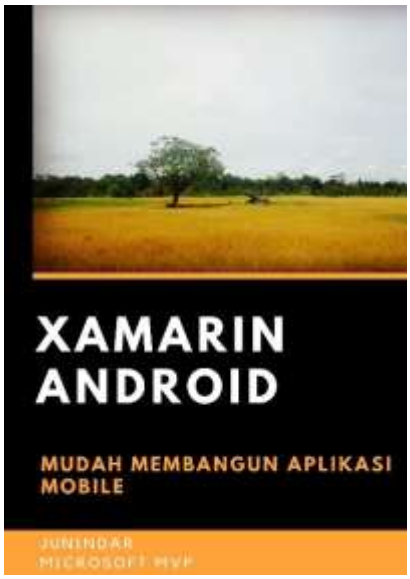
Selanjutnya jalankan program untuk mendapatkan hasil seperti pada gambar dibawah. Untuk melakukan pengujian aplikasi ini, klik tombol Start terlebih dahulu, kemudian mulai berbicara menggunakan microphone. Selama Anda berbicara, sistem akan memproses suara dan secara otomatis menampilkan hasilnya dalam bentuk teks pada area textarea, sesuai dengan apa yang diucapkan. Setelah selesai berbicara, klik tombol Stop untuk menghentikan proses pengenalan suara.



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/2026/09/speech-to-text-dan-text-to-speech.html

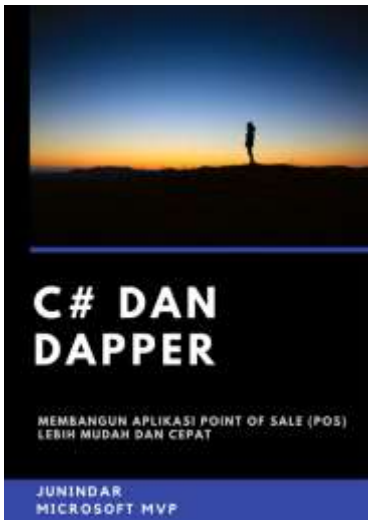
Referensi



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".**