SPEECH SDK

謝謝, my Sprachlehrer

.NET MAUI und der Azure-Speech-Dienst sind eine gute Kombo für eine App, mit der Sie Ihre Aussprache trainieren können.

Als ich noch zur Schule ging, kam im Englischunterricht einmal ein Austauschschüler zu Besuch. Der Lehrer forderte mich auf, ein paar Fragen zu stellen. Der Schüler – und mit ihm die ganze Klasse – sah mich an. Ich schluckte. Meine Zunge war verknotet. Und ich wollte, dass Scotty mich rausbeamt.

Das war in der achten Klasse. Ich war kein schlechter Englischschüler. Ich konnte meinem Alter entsprechend englische Texte schreiben. Ich hatte ja Vokabeln gelernt, Texte gelesen und Grammatik gepaukt. Aber einem Muttersprachler in die Augen sehen und ihn mit peinlich deutschem Akzent ansprechen? Darin hatte ich keine Übung.

Inzwischen lerne ich Chinesisch, und es wird mit zunehmendem Alter nicht leichter. Die Aussprache mancher Phoneme erfordert eine Zungenakrobatik, die ich früher für unmöglich gehalten habe. Und eine professionelle Lehrkraft für die Aussprache steht mir nicht zur Verfügung.

Doch es gibt Hoffnung: das Microsoft Speech SDK [1]. In Verbindung mit dem Azure-Speech-Dienst [2] hat es einiges in puncto Sprachverarbeitung zu bieten (siehe Tabelle 1). Die dotnetpro hat zum Anwendungsfall „Sprachübersetzung“ bereits in einer früheren Ausgabe gezeigt, wie Sie eine einfache, aber praktische Dolmetscher-App erstellen [3]. In diesem Artikel geht es um die Aussprachebewertung. Mit ihr können Sie Sprechlaute einüben und Ihre Aussprache kontinuierlich verbessern.

In eine .NET-MAUI-App verpackt hat man diese Dienstleistung immer in der Tasche. Und das Ganze lässt sich für fünf Audiostunden pro Monat kostenlos nutzen [4].

Tabelle 1: Anwendungsfälle, die das Speech SDK unterstützt

Anwendungsfall (deutsch)	Anwendungsfall (englisch)	Beschreibung
Spracherkennung	Speech-to-text	Sprache-zu-Text (Sprachinterpretation und -transkription)
Sprachsynthese	Text-to-speech	Text-zu-Sprache
Sprachübersetzung	Speech translation	Sprache-zu-Sprache (Übersetzung in eine andere Sprache)
Sprechererkennung	Speaker recognition	Erkennung der sprechenden Person
Aussprachebewertung	Pronunciation assessment	Beurteilung der Aussprache der sprechenden Person
Sprachenerkennung	Language identification	Identifikation einer Sprache
Benutzerdefiniertes Schlüsselwort	Custom keyword	Erkennung von Schlüsselwörtern
Absichtserkennung	Intent recognition	Erkennung der Absicht einer Person

Vorbereitung und Code

Die vorbereitenden Schritte für eine App, die den Speech-Dienst nutzt, wurden schon in [3] beschrieben. In Kurzform:

- Erstellen Sie ein Konto und ein Abonnement in Azure.
- Legen Sie im Azure-Portal den Speech-Dienst an (auf Deutsch: Spracheingabe/-ausgabe).
- Lesen Sie auf der Startseite des Dienstes die Verbindungsdaten ab.
- Erstellen Sie in Visual Studio ein .NET-MAUI-Projekt.
- Installieren Sie Microsoft.CognitiveServices.Speech (Plugin.Maui.Audio wird diesmal nicht benötigt).
- Ergänzen Sie die Datei *Platforms\Android\AndroidManifest.xml* um einen Eintrag für den Mikrofonzugriff (*RECORD_AUDIO*).

Anschließend passen Sie die Datei *MainPage.xaml* so an wie in Listing 1 gezeigt. Die App bekommt hier ein Textfeld für den einzusprechenden Text sowie einen scrollbaren Bereich für die Anzeige der Auswertung. Dazu kommen noch die folgenden vier Schaltflächen:

- *GetText*: Einen zufällig ausgewählten Wikipedia-Text laden
- *Speak*: Text einsprechen
- *Listen*: Text von der App vorlesen lassen
- *Close*: App schließen

Listing 2 zeigt das Hauptgerüst der Datei *MainPage.xaml.cs*. Hier konfigurieren Sie ein *SpeechRecognizer*-Objekt mit den nötigen Verbindungsdaten, einer Timeout-Länge für die Spracherkennung, einer synthetischen Sprechstimme und der zu bewertenden Sprache. In diesem Beispiel ist das US-Englisch.

Welche Sprachen und Sprechstimmen sonst noch möglich sind, können Sie unter [5] in den Registerkarten „Aussprachebewertung“ und „Sprachsynthese“ nachschlagen. ▶

Listing 1: Die angepasste Datei MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/
               dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/
               winfx/2009/xaml"
             x:Class="PronunciationAssessor.MainPage"
             Title="Pronunciation Assessor">
  <Grid Margin="20" RowSpacing="20">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <VerticalStackLayout Grid.Row="0" Spacing="20">
      <Editor x:Name="edt1" HeightRequest="100" />
      <FlexLayout Direction="Row" Wrap="Wrap"
          JustifyContent="Center">
        <Button x:Name="btnGetText" Text="Get
            text" Clicked="BtnGetText_Clicked" Margin="4" />
        <Button x:Name="btnSpeak" Text="Speak"
            Clicked="BtnSpeak_Clicked" Margin="4" />
        <Button x:Name="btnListen" Text="Listen"
          Clicked="BtnListen_Clicked" Margin="4" />
        <Button x:Name="btnClose" Text="Close"
          Clicked="BtnClose_Clicked" Margin="4" />
      </FlexLayout>
    </VerticalStackLayout>
    <ScrollView Grid.Row="1" x:Name="scvMain"
        VerticalOptions="Fill">
      <VerticalStackLayout x:Name="stlMain"
        Spacing="10" />
    </ScrollView>
  </Grid>
</ContentPage>
```

Listing 2 zeigt außerdem, wie die App ordnungsgemäß beendet wird. Dabei sollte eine noch laufende Spracherkennung beendet und die Verbindung zum Speech-Dienst getrennt werden. Danach kann das *SpeechRecognizer*-Objekt über die *Dispose*-Methode entsorgt werden. Beachten Sie, dass der Erkennungsvorgang über die *StopContinuousRecognitionAsync*-Methode gestoppt werden muss, obwohl in der App gar keine „fortwährende" (continuous) Erkennung stattfindet (der Vorgang wird nicht etwa über die *StartContinuousRecognitionAsync*-Methode gestartet).

In Listing 3 sehen Sie die Implementierung der vier Schaltflächen-Ereignisse. ▶

Listing 2: Das Hauptgerüst der Datei MainPage.xaml.cs

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.
  PronunciationAssessment;
using System.Text.Json;

namespace PronunciationAssessor;

public partial class MainPage : ContentPage
{
  private static SpeechConfig speechConfig;
  private static PronunciationAssessmentConfig
    pronunciationConfig;
  private static SpeechRecognizer speechRecognizer;

  public MainPage()
  {
    InitializeComponent();

    // Configure
    speechConfig = SpeechConfig.FromSubscription(
      "(your key)", "(your region)");
    speechConfig.SetProperty(PropertyId.
      SpeechServiceConnection_EndSilenceTimeoutMs,
      "2000"); // Timeout in milliseconds
    speechConfig.SpeechSynthesisVoiceName =
      "en-US-JennyNeural"; // Voice (text to speech)

    speechRecognizer = new SpeechRecognizer(
      speechConfig, "en-US"); // Language-Locale code
      for pronunciation assessment
  }

  protected async override void OnDisappearing()
  {
    base.OnDisappearing();

    await speechRecognizer.
      StopContinuousRecognitionAsync();
    await Task.Delay(1000);
    Connection.FromRecognizer(speechRecognizer).
      Close();
    speechRecognizer?.Dispose();
  }
}
```

Listing 3: Die Implementierung der vier Schaltflächen-Events (Teil 1)

```
private async void BtnGetText_Clicked(object sender,
    EventArgs e)
{
  var today = DateTime.Today;
  var year = today.Year.ToString("D4");
  var month = today.Month.ToString("D2");
  var day = today.Day.ToString("D2");

  var httpClient = new HttpClient();
  var url = $"https://en.wikipedia.org/api/rest_v1/
    feed/featured/{year}/{month}/{day}";
  var response = await httpClient.GetAsync(url);

  if (response.IsSuccessStatusCode)
  {
    var contentJson =
      await response.Content.ReadAsStringAsync();
    var jsonDoc = JsonDocument.Parse(contentJson);

    var texts = jsonDoc.RootElement.GetProperty(
      "onthisday");
    var numberOfTexts = texts.EnumerateArray().Count();

    // Pick random text
    var random = new Random();
    var randomNumber = random.Next(0, numberOfTexts);
    var text = texts[randomNumber].GetProperty(
      "text").ToString();

    edt1.Text = text;
  }
}

private async void BtnSpeak_Clicked(object sender,
    EventArgs e)
{
  if (string.IsNullOrEmpty(edt1.Text))
  {
    return;
  }

  btnGetText.IsEnabled = false;
  btnSpeak.IsEnabled = false;
  btnListen.IsEnabled = false;
  btnClose.IsEnabled = false;

  // Ensure permissions for microphone access
  var status = await Permissions.
    CheckStatusAsync<Permissions.Microphone>();
  if (status != PermissionStatus.Granted)
  {
    status = await Permissions.
      RequestAsync<Permissions.Microphone>();
    if (status != PermissionStatus.Granted)
    {
      return;
    }
  }

  pronunciationConfig =
    new PronunciationAssessmentConfig(edt1.Text,
    GradingSystem.HundredMark, Granularity.Word, true);
  pronunciationConfig.ApplyTo(speechRecognizer);

  stlMain.Clear();
  stlMain.Children.Add(new Label { Text =
    $"Listening...{Environment.NewLine}" });

  try
  {
    var result =
      await speechRecognizer.RecognizeOnceAsync();

    // Output translation results
    switch (result.Reason)
    {
      case ResultReason.RecognizedSpeech:
        stlMain.Children.Add(new Label { Text =
          $"Recognized: {result.Text}{
          Environment.NewLine}" });

        var pronunciationResult =
          PronunciationAssessmentResult.FromResult(
          result);

        stlMain.Children.Add(new Label { Text =
          $"Overall Pronunciation Score:
          {pronunciationResult.PronunciationScore}
          {Environment.NewLine}" });

        stlMain.Children.Add(new Label { Text =
          $"Accuracy: {pronunciationResult.
          AccuracyScore}" });
        stlMain.Children.Add(new Label { Text =
          $"Completeness:
          {pronunciationResult.CompletenessScore}" });
        stlMain.Children.Add(new Label { Text =
          $"Fluency: {pronunciationResult.FluencyScore}
          {Environment.NewLine}" });

        foreach (var word in pronunciationResult.Words)
        {
          stlMain.Children.Add(new Label { Text =
            $"{word.Word}: {word.AccuracyScore}
            (Error: {word.ErrorType})" });
        }
```

Listing 3: Die Implementierung der vier Schaltflächen-Events (Teil 2)

```
        break;

      case ResultReason.NoMatch:
        stlMain.Children.Add(new Label { Text =
          "SPEECH NOT RECOGNIZED" });
        break;

      case ResultReason.Canceled:
        var cancellation = CancellationDetails.
          FromResult(result);
        stlMain.Children.Add(new Label { Text =
          $"CANCELED: Reason = {cancellation.Reason}" });

        if (cancellation.Reason ==
            CancellationReason.Error)
        {
          stlMain.Children.Add(new Label { Text =
            $"CANCELED: ErrorCode =
            {cancellation.ErrorCode}" });
          stlMain.Children.Add(new Label { Text =
            $"CANCELED: ErrorDetails =
            {cancellation.ErrorDetails}" });
        }
        break;

      default:
        break;
    }
  }
  catch (Exception ex)
  {
    stlMain.Children.Add(new Label { Text = $"ERROR:
      {ex.Message}" });
  }
  finally
  {
    btnGetText.IsEnabled = true;
    btnSpeak.IsEnabled = true;
    btnListen.IsEnabled = true;
    btnClose.IsEnabled = true;
  }
}

private async void BtnListen_Clicked(object sender,
    EventArgs e)
{
  using var synthesizer = new SpeechSynthesizer(
    speechConfig);
  using var result = await synthesizer.SpeakTextAsync(
    edt1.Text ?? string.Empty);

  switch (result.Reason)
  {
    case ResultReason.SynthesizingAudioCompleted:
      break;

    case ResultReason.Canceled:
      var cancellation =
        SpeechSynthesisCancellationDetails.
        FromResult(result);
      stlMain.Children.Add(new Label { Text =
        $"CANCELED: Reason = {cancellation.Reason}" });
      if (cancellation.Reason == CancellationReason.Error)
      {
        stlMain.Children.Add(new Label { Text =
          $"CANCELED: ErrorCode =
          {cancellation.ErrorCode}" });
        stlMain.Children.Add(new Label { Text =
          $"CANCELED: ErrorDetails =
          {cancellation.ErrorDetails}" });
      }
      break;

    default:
      break;
  }
}

private void BtnClose_Clicked(object sender, EventArgs e)
{
  Application.Current.Quit();
}
```

- *BtnGetText_Clicked* holt sich mittels einer REST-Anfrage einen Text bei der englischsprachigen Wikipedia ab. Dabei wird ein tagesaktueller „On this day"-Beitrag zufällig ausgewählt. Der Text wird in das Textfeld *edt1* geladen.
- *BtnSpeak_Clicked* stellt den Mikrofonzugriff sicher und konfiguriert die Sprachbewertung mithilfe eines *PronunciationAssessmentConfig*-Objekts. Über die *RecognizeOnceAsync*-Methode wird die Spracherkennung gestartet. Danach werden der vom Speech-Dienst verstandene Text und die Bewertung ausgegeben.
- *BtnListen_Clicked* stellt ein *SpeechSynthesizer*-Objekt her und nutzt die *SpeakTextAsync*-Methode, um den Text aus dem Textfeld *edt1* in Audio umwandeln zu lassen und anschließend auszugeben. *BtnClose_Clicked* schließt die App.

Interessant ist noch, dass Sie die Granularität der Sprachbewertung bestimmen können. In Listing 3 wird sie auf *Word* gesetzt. Dadurch erhalten Sie Resultate für die einzelnen Wörter und den gesamten Text. Es gibt noch eine weitere Ebene: *Phoneme*. Das ist die niedrigstmögliche Ebene. Auf

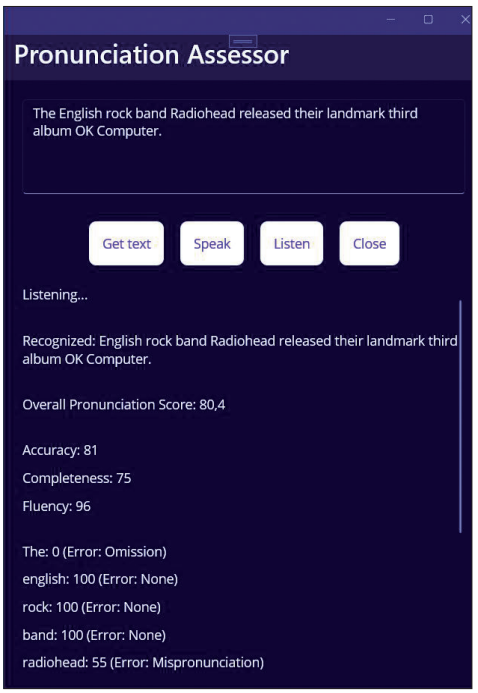
ihr findet zusätzlich eine Bewertung einzelner Sprechlaute statt.

Sprechen, hören, und von vorn

Der Ablauf ist einfach: Laden Sie einen Wikipedia-Inhalt oder tippen Sie Ihren eigenen Text in das Textfeld. Sprechen Sie den Text ein und sehen Sie sich die Auswertung an. Wenn Sie unsicher sind, wie ein Wort ausgesprochen wird, lassen Sie sich den Text von der App vorlesen. Und wiederholen Sie das Prozedere dann nach Belieben.

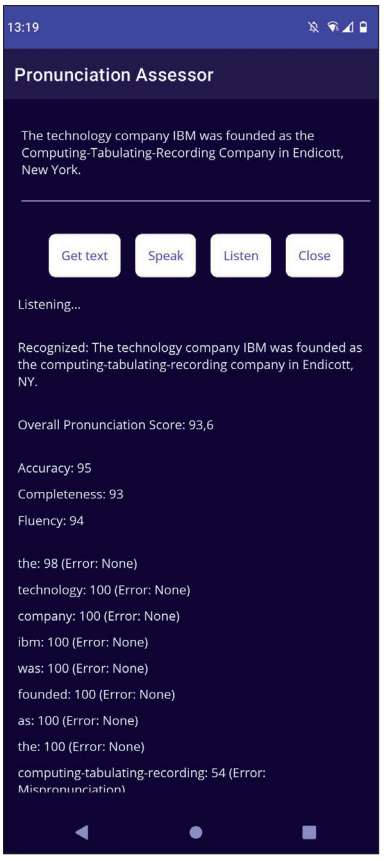
Die Bewertung erfolgt über die Messgrößen „Genauigkeit“ (Accuracy), „Vollständigkeit“ (Completeness) und „Redefluss“ (Fluency). Dazu gibt es als gewichteten Gesamtwert die „Aussprache“ (Pronunciation).

Einzelne Wörter haben darüber hinaus einen „Fehlertyp“ (Error Type). Dieser teilt mit, ob ein Wort ausgelassen (Omission), eingefügt (Insertion) oder falsch ausgesprochen wurde (Mispronunciation). In Bild 1 sehen Sie ein typisches Resultat einer solchen Bewertung.



Nachdem der Text eingesprochen wurde, teilt die App ihre Bewertung mit (Bild 1)

Die Ergebnisse sind fast immer plausibel. Das können Sie überprüfen, indem Sie beim Sprechen absichtlich Fehler machen, zu langsam sprechen oder Wörter auslassen – die Messgrößen verändern sich entsprechend. Lediglich bei der Vorlesefunktion der App müssen Sie aufpassen, denn die Aussprache liegt gelegentlich daneben. Das ist besonders bei Namen und gemischten Sprachen ein Problem. Zum Beispiel spricht die App den Namen „Connie“ wie „Connje“ aus. Wenn man ein solches Wort selbst einspricht, lässt die App auch schon mal die falsche Aussprache durchgehen. Immerhin: Auch die richtige Aussprache wird als korrekt bewertet.



Die Oberfläche der Android-App ist nahezu identisch mit jener der Windows-App (Bild 2)

Ab in die Hosentasche

Wie schon in [3] soll die App noch auf ein Android-Smartphone gebracht werden – für diesen Artikel nicht auf einen Emulator, sondern direkt auf ein physisches Gerät.

Das geht am einfachsten, indem Sie am Gerät die Entwickleroptionen freischalten, den Debug-Modus aktivieren und das Smartphone per USB-Kabel mit dem Entwickler-PC verbinden [6]. Anschließend kann es in Visual Studio als Laufzeitumgebung ausgewählt werden. Das Resultat ähnelt dem einer Windows-App, wie in Bild 2 zu sehen.

Fazit

Ich bin auch dieses Mal vom Speech-Dienst beeindruckt. Die Bewertung der Aussprache klappt in aller Regel sehr gut. Man merkt, dass Microsoft mit seinen KI-Modellen aktuell an der Spitze der Liga mitspielt.

Für eine professionelle Lösung lässt sich die App sicher weiter aufpeppen, schöner machen und eleganter mit MVVM-Pattern umsetzen. Die hier gezeigte Lösung könnte dafür die Grundlage bilden.

Ich habe die Hoffnung, zukünftig peinliche Sprechpausen durch eine verknotete Zunge zu vermeiden.

Danke, thank you, 謝謝, my Sprachlehrer. ■

[1] Speech SDK, www.dotnetpro.de/SL2312Aussprache1
[2] Dokumentation für den Speech-Dienst, www.dotnetpro.de/SL2312Aussprache2
[3] 謝謝, my Dolmetscher, dotnetpro 10/2023, Seite 113ff., www.dotnetpro.de/A2310Translator
[4] Sprachdienste – Preise, www.dotnetpro.de/SL2312Aussprache3
[5] Sprach- und Stimmunterstützung für den Speech-Dienst, www.dotnetpro.de/SL2312Aussprache4
[6] Einrichten eines Android-Geräts für das Debuggen, www.dotnetpro.de/SL2312Aussprache5

Martin Gossen
ist IT-Berater bei der IKS GmbH in Hilden. Er erstellt seit 15 Jahren Softwarelösungen auf Basis von C#, .NET und Microsoft SQL Server. Sie erreichen ihn unter **m.gossen@iks-gmbh.com**.

dnpCode A2312Aussprache