SPEECH SDK

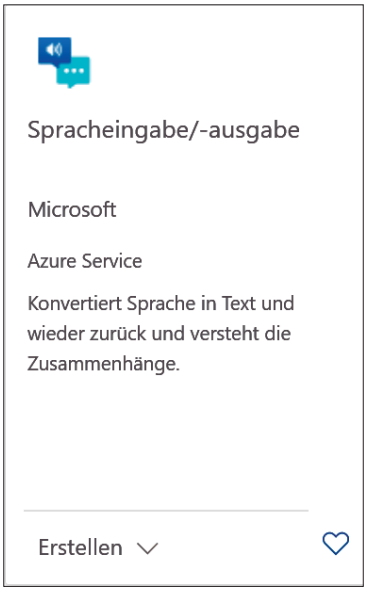
謝謝, my Dolmetscher

Mit dem Azure-Speech-Dienst bauen Sie im Handumdrehen eine sprechende Übersetzungs-App.

Meine Frau stammt aus Taiwan, ich aus dem Ruhrgebiet. Da stoßen Welten aufeinander. Meistens unterhalten wir uns auf Englisch, weil das der kleinste gemeinsame Nenner ist. Manchmal versuche ich es auf Chinesisch. Das ist allerdings riskant. Eine Silbe falsch betont, und aus dem „Lama" wird eine heftige Beleidigung. Das kann schnell den sonntäglichen Zoobesuch ruinieren. Deshalb sehne ich mich schon lange nach einem Dolmetscher, der mir zur Seite steht. Mir schwebt eine Anwendung vor, die ich immer bei mir habe. Das Ziel ist also, eine App zu entwickeln.

Zum Glück gibt es heutzutage künstliche Intelligenz, und es wird immer einfacher, sie einzusetzen. Für viele Anwendungsfälle muss sich niemand mehr tief in die Geheimnisse der Datenwissenschaften einarbeiten, Python-Kenntnisse auffrischen oder den Umgang mit Jupyter-Notebooks lernen. Der neueste Trend: vortrainierte Machine-Learning-Modelle, die man direkt nutzen kann – KI as a Service.

Microsofts Speech SDK [1] geht in diese Richtung. Einen Dienst aktiviert, ein paar Zeilen Code geschrieben, und die Sprachverarbeitung läuft. Damit lassen sich unterschiedliche Aufgaben bewältigen (siehe Tabelle 1). Für den Dolmetscher passt der Anwendungsfall „Sprachübersetzung".



Die Kachel für den Azure Speech-Dienst heißt auf Deutsch „Spracheingabe/-ausgabe" (Bild 1)

Die eigentliche Arbeit erledigt ein vortrainiertes Modell, das in Azure über einen Speech-Dienst [2] verfügbar gemacht wird. Der Speech-Dienst ist Teil der Azure Cognitive Services [3], lässt sich aber auch allein nutzen – für fünf Audiostunden pro Monat sogar kostenlos. Das Preismodell dazu heißt „Free Tier F0". Weitere Details zu den Kosten finden Sie unter [4].

Vorbereitung und Code

Wenn Sie sich ebenfalls einen persönlichen Dolmetscher zulegen wollen, brauchen Sie zuerst ein Konto und ein Abonnement in Azure. Im Azure-Portal legen Sie dann den Speech-Dienst an (Bild 1). Anschließend können Sie auf der Startseite des Dienstes zwei Schlüssel und weitere Verbindungsdaten ablesen (Bild 2). Von den Schlüsseln brauchen Sie immer nur einen; der andere kann bei Bedarf durch einen neuen ersetzt werden, ohne dass im laufenden Betrieb Ausfallzeiten entstehen.

Und schon geht es weiter in Ihrer Entwicklungsumgebung. Erstellen Sie ein .NET-MAUI-Projekt, dann können Sie die fertige App nicht nur unter Windows, sondern auch auf Smartphones nutzen. Danach installieren Sie über NuGet das Speech SDK (Microsoft.CognitiveServices.Speech) und die Bibliothek Plugin.Maui.Audio, die für die Sprachausgabe sorgt. ▶

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

Damit die App Zugriff auf das Mikrofon von Android-Smartphones bekommt, fügen Sie der Datei *Platforms/Android/AndroidManifest.xml* folgende Zeile hinzu:

```
<uses-permission android:name="android.permission
  .RECORD_AUDIO" />
```

Für iOS-Geräte müssen Sie stattdessen in der *Info.plist*-Datei ein Schlüssel-Wert-Paar hinzufügen:

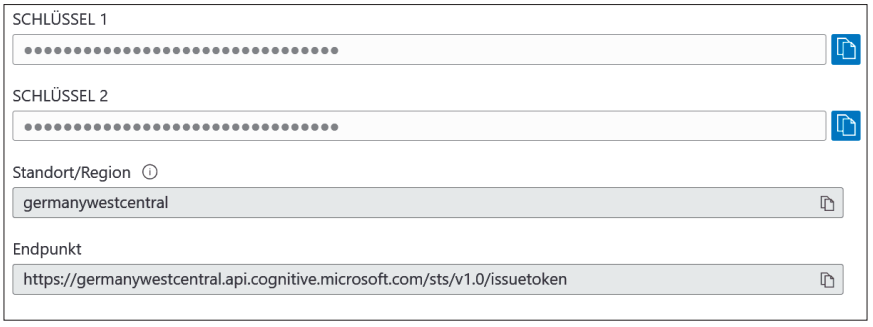
```
<key>NSMicrophoneUsageDescription</key>
<string>Zugriff auf das Mikrofon wird benötigt, um Audio
  aufzunehmen.</string>
```

Die Datei *MainPage.xaml* bekommt einen Titel, ein angepasstes *VerticalStackLayout*-Element und einen Button, vergleiche Listing 1.

Die Programmlogik steckt in der Datei *MainPage.xaml.cs* (Listing 2). Hier setzen Sie zuerst einige Variablen für die Verbindung zum Speech-Dienst und zur Sprachverarbeitung. Welche Sprachen und Stimmen möglich sind, sehen Sie unter [5]. Schauen Sie dazu in die Registerkarten für Spracherkennung, Sprachübersetzung und Sprachsynthese (vergleiche Tabelle 1). Tipp: Emily aus Irland (*en-IE-EmilyNeural*) wirkt beruhigend. Einen Übersetzungsfehler im Zoo würde man ihr leicht verzeihen.

Listing 1: Die 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="SpeechTranslatorMaui.MainPage"
    Title="Speech Translator">
  <ScrollView>
    <VerticalStackLayout x:Name="vslMain"
        Spacing="10" Padding="20,20">
      <Button Text="Start" Clicked="Btn1_OnClicked"
        HorizontalOptions="Center" />
    </VerticalStackLayout>
  </ScrollView>
</ContentPage>
```



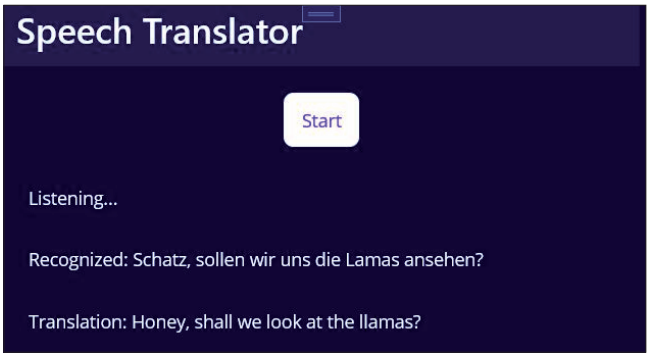
Die Verbindungsdaten des Speech-Dienstes (Bild 2)

Die App soll dolmetschen. Sie erledigt damit drei Schritte auf einmal: Sprache-zu-Text / Übersetzung / Text-zu-Sprache. Ganz wesentlich ist das *SpeechTranslationConfig*-Objekt. Ihm weisen Sie die vorher definierten Werte zu. Achtung: In der Standardeinstellung ist ein Filter für vulgäre Ausdrücke (Profanity-Filter) aktiv. Sie können ihn aber über die *SetProfanity*-Methode abschalten – Sie sind ja erwachsen. Über *AudioConfig* legen Sie das Standardmikrofon als Eingabequelle fest. Nun können Sie ein *TranslationRecognizer*-Objekt erzeugen, das sich um die Kommunikation mit dem Speech-Dienst kümmert.

Die Sprachausgabe erledigen Sie über eine anonyme Methode, die Sie an das *Synthesizing*-Ereignis des *TranslationRecognizer*-Objekts anhängen. Hier empfangen Sie den gesprochenen Text über die *GetAudio*-Methode. Sie können ihn dann mit einem *IAudioPlayer* abspielen.

Listening ...

Fehlt noch die eigentliche Sprachübersetzung. Sie wird im Ereignis-Handler *Btn1_OnClicked* initiiert. Nachdem die Zugriffsrechte auf das Mikrofon sichergestellt sind, wird über



Die App hört und spricht ... (Bild 3)

... auch Chinesisch (Bild 4)

Listing 2: Die Datei MainPage.xaml.cs mit der Programmlogik (Teil 1)

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Translation;
using Plugin.Maui.Audio;

namespace SpeechTranslatorMaui;

public partial class MainPage : ContentPage
{
  private const string SPEECH_SUBSCRIPTION_KEY =
    "(Your key)";
  private const string SPEECH_SERVICE_REGION =
    "(Your service region)";
  private const string FROM_LANGUAGE_LOCALE = "de-DE";
    // Language-Locale code (speech-to-text)
  private const string TO_LANGUAGE =
    "en"; // Language code (speech translation)
  private const string TO_LANGUAGE_LOCALE = "en-US";
    // Language-Locale code (text to speech)
  private const string VOICE_NAME =
    "en-US-JennyNeural"; // Voice (text to speech)
  private static MemoryStream audioStream;
  private static TranslationRecognizer transRecog;
  private static IAudioPlayer audioPlayer;

  public MainPage()
  {
    InitializeComponent();

    // Configure translation
    var speechTransConf = SpeechTranslationConfig
      .FromSubscription(SPEECH_SUBSCRIPTION_KEY,
      SPEECH_SERVICE_REGION);

    speechTransConf.SpeechRecognitionLanguage =
      FROM_LANGUAGE_LOCALE;
    speechTransConf.AddTargetLanguage(TO_LANGUAGE);
    speechTransConf.SpeechSynthesisLanguage =
      TO_LANGUAGE_LOCALE;
    speechTransConf.VoiceName = VOICE_NAME;
    speechTransConf.SpeechSynthesisVoiceName =
      VOICE_NAME;
    speechTransConf.SetProfanity(ProfanityOption.Raw);
      // Disable profanity filter

    // Use default microphone as input device
    using var audioConfig =
      AudioConfig.FromDefaultMicrophoneInput();
    transRecog = new TranslationRecognizer(
      speechTransConf, audioConfig);

    // Speech synthesis
    transRecog.Synthesizing += (_, e) =>
    {
      if (e.Result.Reason ==
          ResultReason.SynthesizingAudio)
      {
        var audio = e.Result.GetAudio();

        if (audio != null && audio.Length > 44)
            // Workaround for corrupt audio
        {
          // Play audio
          audioStream = new MemoryStream(audio);
          audioPlayer = AudioManager.Current
            .CreatePlayer(audioStream);
          audioPlayer.Play();
        }
      }
    };
  }

  private async void Btn1_OnClicked(
    object sender, EventArgs e)
```

die *RecognizeOnceAsync*-Methode auf die Spracheingabe gewartet – und zwar 15 Sekunden lang. Den Wert können Sie aber auch anpassen:

```
speechTranslationConfig.SetProperty(PropertyId
  .SpeechServiceConnection_InitialSilenceTimeoutMs,
  "5000"); // in milliseconds
```

Sobald der *TranslationRecognizer* Sprache erkennt, wartet er auf die nächste Sprechpause. Dann schickt er die Audioaufnahme an den Speech-Dienst. Dort wird sie transkribiert, übersetzt und in Audio zurückverwandelt. Sowohl der transkribierte als auch der übersetzte Text werden über ein *TranslationRecognitionResult*-Objekt zurückgeliefert, die Au- ▶

ConfigureAwait

Es steckt schon im Namen: Die *RecognizeOnceAsync*-Methode wird asynchron ausgeführt. Einige Microsoft-Autoren hängen dieser Methode einen weiteren Aufruf an:

- **ConfigureAwait(false)**. Hier geht es um die Zusammenführung von Threads, mit möglichen Konsequenzen für die Performance und Deadlocks. Das Thema ist komplex und wird viel diskutiert. Im Grunde gilt die Daumenregel: Verwenden Sie *ConfigureAwait(false)* in Code-Bibliotheken, aber nicht in Anwendungen. Aus erster Hand können Sie das unter [8] nachlesen.

Listing 2: Die Datei MainPage.xaml.cs mit der Programmlogik (Teil 2)

```
{
  var btn1 = sender as Button;

  // 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;
  }

  btn1.IsEnabled = false;
  vslMain.Children.Add(new Label { Text =
    $"{Environment.NewLine}Listening..." });

  // Listen and translate
  var result =
    await transRecog.RecognizeOnceAsync();

  // Output translation results
  switch (result.Reason)
  {
    case ResultReason.TranslatedSpeech:
      vslMain.Children.Add(new Label { Text =
        $"{Environment.NewLine}Recognized:
          {result.Text}" });
      vslMain.Children.Add(new Label { Text =
        $"{Environment.NewLine}Translation:
          {result.Translations[TO_LANGUAGE]}" });
      break;

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

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

  btn1.IsEnabled = true;
}

protected override void OnDisappearing()
{
  base.OnDisappearing();
  if (audioStream != null) audioStream.Dispose();
  if (transRecog != null) transRecog.Dispose();
  if (audioPlayer != null) audioPlayer.Dispose();
}
}
```

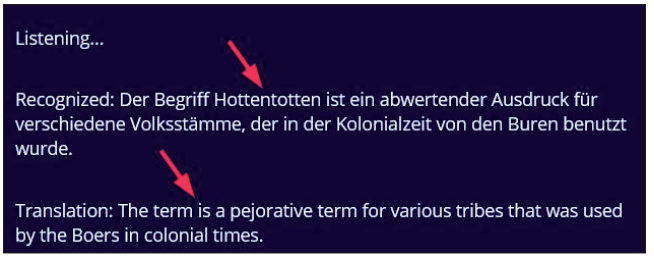
diodaten über ein *TranslationSynthesisResult*-Objekt in dem erwähnten *Synthesizing*-Ereignis.

Klappt das?

Das Ganze funktioniert auf Anhieb erstaunlich gut, selbst bei Hintergrundgeräuschen oder undeutlicher Aussprache (vergleiche Bild 3 und Bild 4). Auch die Reaktionszeit ist gut: Die Übersetzung dauert meistens nur einen kurzen Moment. Ungeduld kommt nicht auf.

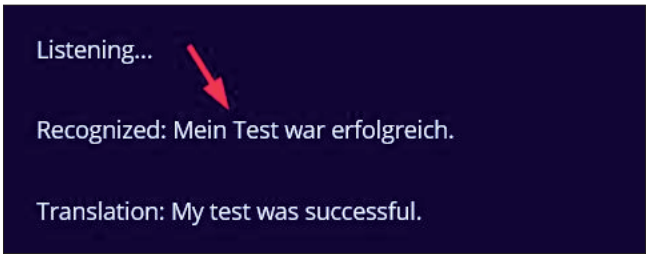
Fehlerfrei ist der Speech-Dienst allerdings nicht.

Beispiel 1: Es gibt Wörter, die zwar erkannt, aber nicht übersetzt werden. Beispiel: „Hottentotten“ (Bild 5). Sehr seltsam. Vielleicht hängt das mit „politisch unkorrekten“ Begriffen zusammen. Der Profanity-Filter kann nicht schuld sein, weil er Buchstaben durch Platzhalter ersetzt, statt Wörter zu löschen. Dumm ist, dass der Audio-Stream defekt ist, wenn der gesprochene Text ausschließlich aus solchen Wörtern be-



Ein Wort wurde verschluckt (Bild 5)

Meintest du „Mein Test“? Hier muss feinjustiert werden (Bild 6)



Der Ausdruck „Mein Test“ wird jetzt richtig erkannt (Bild 7)

steht. Am einfachsten umgehen Sie das Problem, indem Sie sehr kurze Streams verwerfen. In Listing 2 wird alles unter 45 Byte ignoriert.

Beispiel 2: Manche Ausdrücke werden grundsätzlich falsch interpretiert. Aus „Mein Test“ wird so „meintest“ – eine Form des Verbs „meinen“ (Bild 6). Auch dann, wenn eine deutliche Sprechpause zwischen den Wörtern gemacht wird.

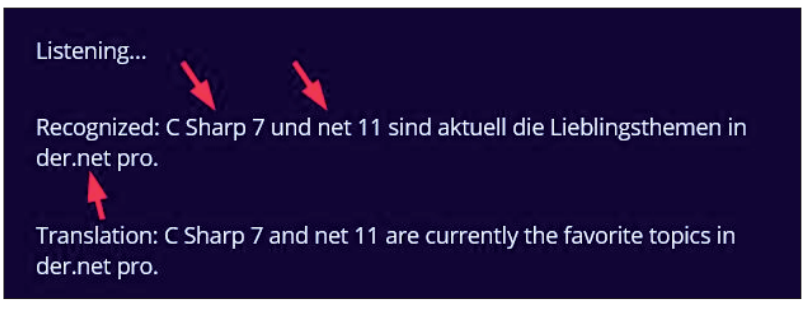
Beispiel 3: Laut Dokumentation soll der *TranslationRecognizer* in der Lage sein, die gesprochene Sprache zu identifizieren. Zu diesem Zweck kann man ihm auch ein *AutoDetectSourceLanguageConfig*-Objekt übergeben:

```
var autoDetectSourceLanguageConf =
  AutoDetectSourceLanguageConfig
  .FromLanguages(new string[] {
  "en-US", "de-DE" });
transRecog = new TranslationRecognizer(speechTransConf,
  autoDetectSourceLanguageConf, audioConfig);
```

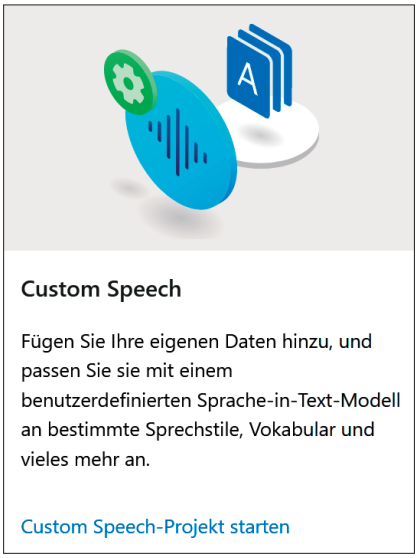
Das funktioniert auch; die Sprache wird korrekt erkannt und übersetzt. Das Problem: Wenn Sie den *TranslationRecognizer* auf diese Weise initialisieren, wird das *Synthesizing*-Ereignis nicht mehr ausgelöst. Kurios. Die Sprachausgabe fällt dann aus.

Hand anlegen

Das Problem aus Beispiel 2 lässt sich leicht beheben: Fügen Sie falsch erkannte Ausdrücke einer Begriffsliste hinzu. Das geht so:



Fachbegriffe erkennt das Standardmodell nicht (Bild 9)



Die Kachel „Custom Speech“ im Speech Studio (Bild 8)

```
var phraseList = PhraseListGrammar
  .FromRecognizer(translationRecognizer);
phraseList.AddPhrase("Mein Test");
```

Danach funktioniert die Erkennung (Bild 7). Diese Lösung versagt jedoch bei Begriffen, die ungewöhnlich ausgesprochen werden. Begriffslisten sollten auch nur maximal 500 Ausdrücke enthalten.

Der Speech-Dienst hat aber noch ein Ass im Ärmel: Custom Speech [6]. Hierüber können Sie das Sprachmodell durch zusätzliches Training an spezielle Bedürfnisse anpassen, zum Beispiel an Fachjargon oder eine individuelle Sprechstimme.

Custom Speech legen Sie nicht in Azure an, sondern im Speech Studio [7]. Hier wählen Sie den Punkt *Custom Speech-Projekt starten* (Bild 8).

Sie landen dann im Bereich *Ihre Daten vorbereiten*, in dem erklärt wird, wie Sie Ihre Trainingsdaten aufbereiten müssen. Was genau zu tun ist, hängt davon ab, was Sie erreichen wollen. Mögliche Optionen sind:

- **Terminologie:** Sie wollen Begriffe hinzufügen, die vom Standardmodell nicht erkannt werden. Das können zum Beispiel Ausdrücke aus einer besonderen Fachdomäne (etwa IT, Wissenschaft oder Recht) sein.
- **Aussprache:** Sie wollen, dass Begriffe besser erkannt werden, die ungewöhnlich ausgesprochen werden (zum Beispiel ausgedachte Produktnamen oder Akronyme).
- **Satzstruktur:** Sie wollen, dass bestimmte Muster in Äußerungen erkannt werden. Das könnte bei einem Chatbot zum Beispiel eine Äußerung wie „Ich habe eine Frage zu Produkt X“ sein.
- **Akustikvarianz:** Sie wollen die Erkennungsgenauigkeit bei Sprechstilen, Akzenten oder Hintergrundgeräuschen erhöhen.

Angenommen, die App soll den folgenden Satz übersetzen: „C# 7 und .NET 11 sind aktuell die Lieblingsthemen in der dotnetpro“. Das geht mit dem Standardmodell ziemlich daneben (Bild 9).

Der Satz enthält Fachbegriffe und gemischte Sprachen. Mit Anpassungen der Terminolo- ▶

gie und Aussprache bekommen Sie das aber in den Griff.

Training für Fortgeschrittene

Erstellen Sie zwei TXT-Dateien. Die erste (für die Terminologie) heißt *terminology.txt* und bekommt die neuen Fachbegriffe (einen pro Zeile):

```
dotnetpro
.NET
C#
```

Die zweite (für die Aussprache) heißt *pronunciation.txt* und bekommt zusätzlich ein Tabulatorzeichen sowie die phonetische Sequenz, die aus Buchstaben, Silben, Wörtern oder auch einer Kombination aus allen dreien bestehen kann:

```
dotnetpro[tab]dot net pro
.NET[tab]dot net
C#[tab]c sharp
```

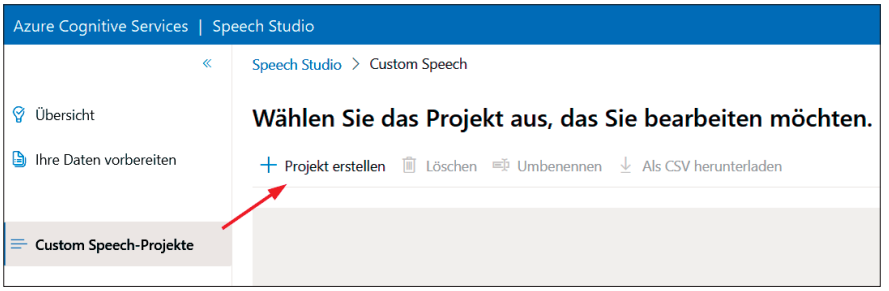
Wie die phonetischen Laute genau funktionieren, steht in dem bereits vorher erwähnten Bereich *Ihre Daten vorbereiten*.

Nun können Sie im Speech Studio unter *Custom Speech-Projekte* ein neues Projekt erstellen (Bild 10). Als Sprache müssen Sie Ihre Eingangssprache angeben, auch wenn die Begriffe aus einer anderen Sprache stammen.

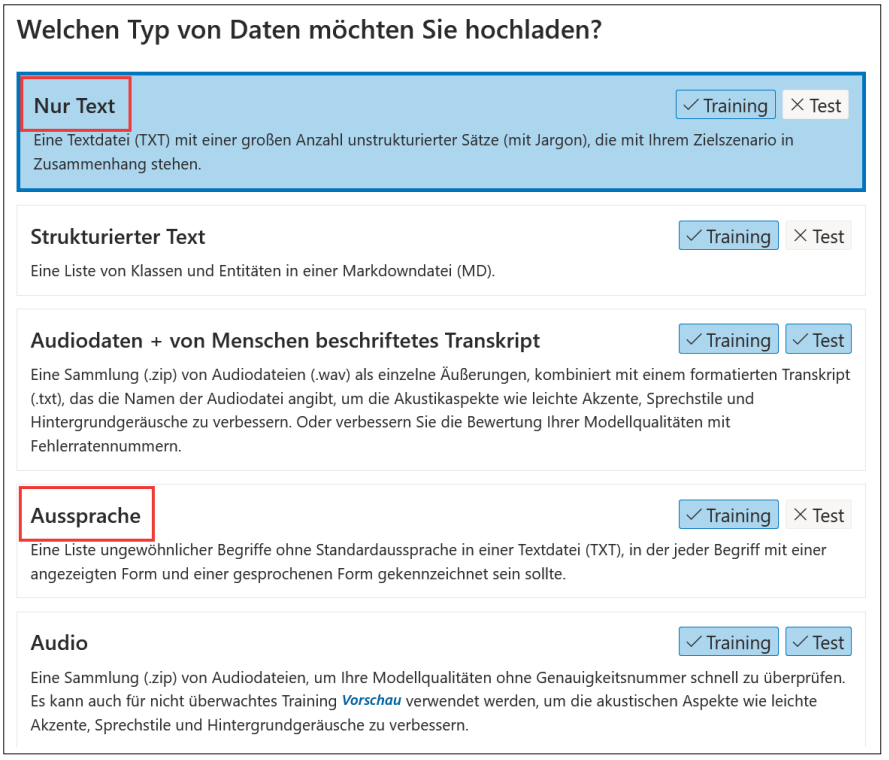
Folgen Sie den Anweisungen und laden Sie an der geeigneten Stelle die beiden TXT-Dateien hoch (*terminology.txt* als „Nur Text“ und *pronunciation.txt* als „Aussprache“, siehe (Bild 11).

Hangeln Sie sich anschließend durch die Menüpunkte (Bild 12). Die Schritte sind weitgehend selbsterklärend. Dabei wird ein neues Modell trainiert und an einem Endpunkt bereitgestellt. Tests brauchen Sie hier nicht auszuführen.

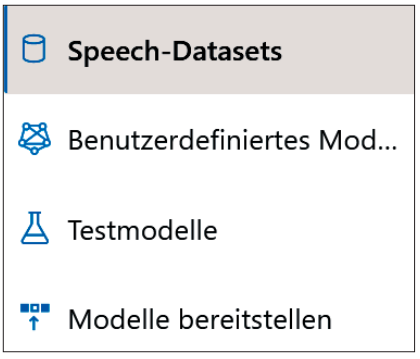
Zum Schluss lesen Sie dann noch auf der Seite des fertig trainierten Modells die Endpoint-ID ab. Um das neue Modell



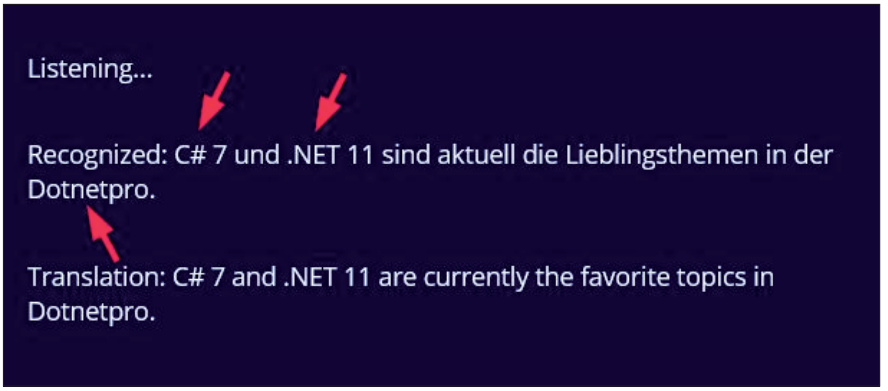
Ein neues Projekt in Custom Speech erstellen (Bild 10)



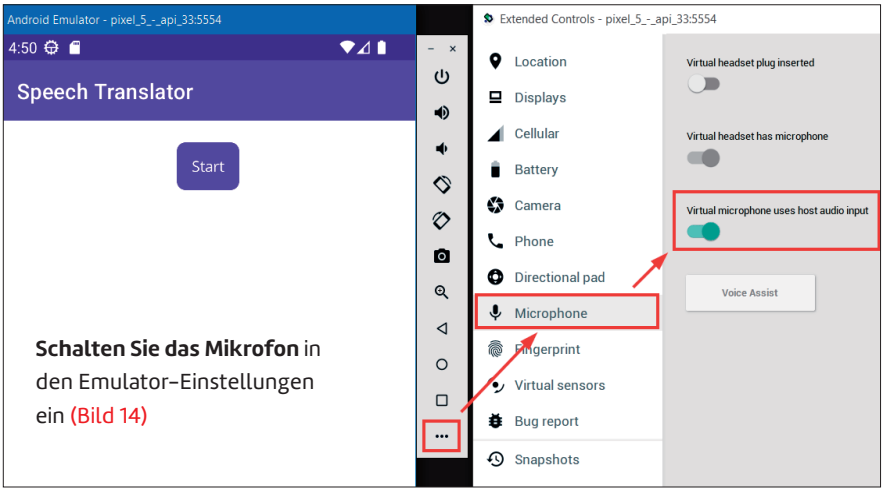
Die Upload-Optionen „Nur Text“ und „Aussprache“ (Bild 11)



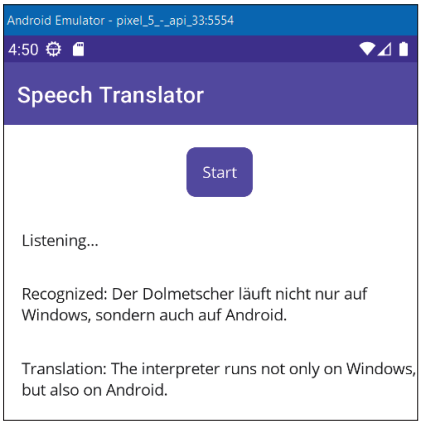
Die Menüpunkte des Custom-Speech-Projekts (Bild 12)



Nach dem Training sehen die Fachbegriffe besser aus (Bild 13)



Schalten Sie das Mikrofon in den Emulator-Einstellungen ein (Bild 14)



Fertig – die App läuft im Emulator (Bild 15)

einzusetzen, fügen Sie die Endpunkt-ID wie folgt dem Code hinzu:

```
speechTransConf.EndpointId =
  "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
```

Und tatsächlich, die problematischen Ausdrücke werden jetzt erkannt (Bild 13).

Es gibt hier noch eine wichtige Einschränkung: Im Free-Tier-Preismodell dürfen die TXT-Dateien für die Aussprache nur 1 KB groß sein. Das sind 40 bis 50 Ausdrücke. Sie können aber beliebig viele davon hochladen.

Mobil

Nun muss die App noch aufs Smartphone gebracht werden. Für diesen Artikel soll es reichen, sie in einem Emulator zu starten. Das ist bei .NET-MAUI-Projekten geradezu trivial.

Legen Sie in Visual Studio über *Tools* | *Android* | *Android Device Manager* ein virtuelles Android-Gerät an und wählen Sie es anschließend als Laufzeitumgebung aus. Die App startet dann im Emulator. Schalten Sie für die Spracheingabe noch das virtuelle Mikrofon ein (Bild 14), und Sie können loslegen (Bild 15).

Fazit

Ich bin beeindruckt. Die Sprachübersetzung klappt in den meisten Fällen hervorragend. Probleme mit Fachbegriffen lassen sich mit Begriffslisten und zusätzlichem Training lösen.

Und dank .NET MAUI kann ich die App sowohl auf meinem PC als auch meinem Smartphone nutzen, ohne dass ich plattformspezifischen Code schreiben muss. Deshalb nutze ich die App jetzt im Alltag beim Chinesischlernen. Versehentliche Beleidigungen sind passé. Danke, thank you, 謝謝, my Dolmetscher.

Natürlich können Sie den Dienst auch in professionelle Softwareprodukte einbauen. Und – gegen Bezahlung – das Volumen an Audiostunden erhöhen.

Denken Sie dabei aber stets daran, dass alles Gesprochene in die Cloud wandert. Sie sind wegen des Azure-Abonnements auch persönlich identifizierbar. Das ist nur vertretbar, wenn Sie sich mit Microsofts Datenschutzerklärungen anfreunden können. ■

[1] Speech SDK, www.dotnetpro.de/SL2310Translator1
[2] Dokumentation für den Speech-Dienst, www.dotnetpro.de/SL2310Translator2
[3] Dokumentation zu Azure Cognitive Services, www.dotnetpro.de/SL2310Translator3
[4] Azure-Sprachdienste – Preise, www.dotnetpro.de/SL2310Translator4
[5] Azure – Sprach- und Stimmunterstützung für den Speech-Dienst, www.dotnetpro.de/SL2310Translator5
[6] Azure – Was ist Custom Speech?, www.dotnetpro.de/SL2310Translator6
[7] Speech Studio, https://speech.microsoft.com/portal
[8] ConfigureAwait FAQ, www.dotnetpro.de/SL2310Translator7

Speech Studio

Das Speech Studio ist eine Microsoft-Website mit umfangreichen Informationen und Tools, die den Einstieg in die Sprachverarbeitung erleichtern sollen. Hier werden die wichtigsten Anwendungsfälle erklärt. Sie können auch Funktionen ausprobieren, ohne Programmcode schreiben zu müssen. Außerdem gibt es Sprachproben der synthetischen Sprechstimmen.

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 A2310Translator