**Install Necessary Packages**

Install the following NuGet packages:

* Microsoft.Data.Sqlite for SQLite support.
* System.Net.Http for making HTTP requests.

PROGRAM CS:

using System;

using System.Data.SQLite;

using System.IO;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace SpeechToTextApp

{

class Program

{

private static readonly string subscriptionKey = "YourSubscriptionKey";

private static readonly string region = "YourServiceRegion";

private static readonly string databasePath = "Data Source=audio.db";

static async Task Main(string[] args)

{

// Retrieve audio file from database

var audioFilePath = RetrieveAudioFromDatabase();

if (audioFilePath != null)

{

// Convert speech to text

var transcript = await ConvertSpeechToText(audioFilePath);

if (!string.IsNullOrEmpty(transcript))

{

// Store transcript in database

StoreTranscriptInDatabase(transcript);

Console.WriteLine("Transcription stored successfully.");

}

}

else

{

Console.WriteLine("No audio file found in the database.");

}

}

private static string RetrieveAudioFromDatabase()

{

using (var connection = new SQLiteConnection(databasePath))

{

connection.Open();

var command = new SQLiteCommand("SELECT FilePath FROM AudioFiles LIMIT 1", connection);

var result = command.ExecuteScalar();

return result?.ToString();

}

}

private static async Task<string> ConvertSpeechToText(string audioFilePath)

{

var uri = $"https://{region}.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US";

using (var httpClient = new HttpClient())

{

httpClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

using (var audioFile = File.OpenRead(audioFilePath))

using (var content = new StreamContent(audioFile))

{

content.Headers.ContentType = new MediaTypeHeaderValue("audio/wav");

var response = await httpClient.PostAsync(uri, content);

response.EnsureSuccessStatusCode();

var result = await response.Content.ReadAsStringAsync();

dynamic json = Newtonsoft.Json.JsonConvert.DeserializeObject(result);

return json.DisplayText;

}

}

}

private static void StoreTranscriptInDatabase(string transcript)

{

using (var connection = new SQLiteConnection(databasePath))

{

connection.Open();

var command = new SQLiteCommand("INSERT INTO Transcripts (Text) VALUES (@Text)", connection);

command.Parameters.AddWithValue("@Text", transcript);

command.ExecuteNonQuery();

}

}

}

}

DB

CREATE TABLE AudioFiles (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

FilePath TEXT NOT NULL

);

CREATE TABLE Transcripts (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Text TEXT NOT NULL

);

To handle transcription of live calls by multiple clients, store audio in Azure, transcribe it, and then return the text to an external database using a REST API, you'll need to build a comprehensive solution. This solution will involve several components:

1. **Azure Communication Services**: To handle live call audio.
2. **Azure Blob Storage**: To store the audio files.
3. **Azure Speech Service**: To transcribe the audio files.
4. **ASP.NET Core Web API**: To handle REST API requests.
5. **External Database**: To store the transcription text.

### Step-by-Step Implementation

### Step 1: Set Up Azure Services

#### Azure Communication Services

1. **Create an Azure Communication Services resource**.
2. Use Azure Communication Services SDK to capture live call audio and save it to Azure Blob Storage.

#### Azure Blob Storage

1. **Create a Blob Storage account** in Azure.
2. Create a container to store the audio files.

#### Azure Speech Service

1. **Create a Speech Service resource** in Azure.

### Step 2: Create ASP.NET Core Web API

Create an ASP.NET Core Web API project to handle the entire workflow.

#### Install Necessary Packages

Use NuGet Package Manager to install required packages:

* Microsoft.Azure.CognitiveServices.Speech for Speech Service.
* Azure.Storage.Blobs for Blob Storage.
* Microsoft.EntityFrameworkCore for Entity Framework (optional, if using EF for DB operations).
* System.Data.SqlClient for SQL Server operations.

#### Program.cs

Here's a complete example of how to implement the API:

csharp

Copy code

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Azure.CognitiveServices.Speech;

using Azure.Storage.Blobs;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using System.Data.SqlClient;

namespace LiveCallTranscriptionAPI

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

public class TranscriptionController : ControllerBase

{

private readonly string subscriptionKey = "YourSubscriptionKey";

private readonly string region = "YourServiceRegion";

private readonly string storageConnectionString = "YourStorageConnectionString";

private readonly string containerName = "audiocontainer";

private readonly string externalDbConnectionString = "YourExternalDbConnectionString";

[HttpPost("transcribe")]

public async Task<IActionResult> Transcribe([FromForm] IFormFile audioFile, int callId)

{

if (audioFile == null || callId <= 0)

{

return BadRequest("Invalid input");

}

string audioFilePath = await SaveAudioFileAsync(audioFile);

string blobUri = await UploadAudioToBlobAsync(audioFilePath);

string transcription = await TranscribeAudioAsync(blobUri);

await StoreTranscriptionInDatabaseAsync(transcription, callId);

return Ok(transcription);

}

private async Task<string> SaveAudioFileAsync(IFormFile audioFile)

{

string filePath = Path.Combine(Path.GetTempPath(), audioFile.FileName);

using (var stream = new FileStream(filePath, FileMode.Create))

{

await audioFile.CopyToAsync(stream);

}

return filePath;

}

private async Task<string> UploadAudioToBlobAsync(string audioFilePath)

{

BlobServiceClient blobServiceClient = new BlobServiceClient(storageConnectionString);

BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(containerName);

BlobClient blobClient = containerClient.GetBlobClient(Path.GetFileName(audioFilePath));

await blobClient.UploadAsync(audioFilePath, true);

return blobClient.Uri.ToString();

}

private async Task<string> TranscribeAudioAsync(string blobUri)

{

var config = SpeechConfig.FromSubscription(subscriptionKey, region);

var audioInput = AudioConfig.FromWavFileInput(blobUri);

var recognizer = new SpeechRecognizer(config, audioInput);

var result = await recognizer.RecognizeOnceAsync();

if (result.Reason == ResultReason.RecognizedSpeech)

{

return result.Text;

}

else

{

throw new Exception($"Speech recognition failed. Reason: {result.Reason}");

}

}

private async Task StoreTranscriptionInDatabaseAsync(string transcription, int callId)

{

using (SqlConnection connection = new SqlConnection(externalDbConnectionString))

{

await connection.OpenAsync();

var command = new SqlCommand("INSERT INTO Transcriptions (CallId, Text) VALUES (@CallId, @Text)", connection);

command.Parameters.AddWithValue("@CallId", callId);

command.Parameters.AddWithValue("@Text", transcription);

await command.ExecuteNonQueryAsync();

}

}

}

}

### Database Setup

#### SQL Server Table

sql

Copy code

CREATE TABLE Transcriptions (

Id INT PRIMARY KEY IDENTITY(1,1),

CallId INT NOT NULL,

Text NVARCHAR(MAX) NOT NULL,

Timestamp DATETIME DEFAULT GETDATE()

);

### Step 3: Deploy and Test

1. Deploy the ASP.NET Core Web API to a suitable environment (e.g., Azure App Service).
2. Test the API with live calls to ensure the audio is captured, stored, transcribed, and saved in the external database.

### Notes

1. **Concurrency**: The API is designed to handle concurrent requests. Azure App Service will scale the application to handle multiple calls.
2. **Error Handling**: Add comprehensive error handling and logging for production use.
3. **Authentication**: Ensure secure handling of credentials and use managed identities where possible.
4. **Scalability**: Consider using Azure Functions or Kubernetes for additional scalability if needed.
5. **Performance**: Optimize audio file handling and database operations for better performance.

This setup provides a robust way to handle live call transcription for multiple clients using Azure services and an external database. Adjust and expand the implementation according to your specific requirements and environment.

THE SAME ABOVE ONLY CALL RECEIVING

To capture live call audio using Azure Communication Services (ACS) and save it to Azure Blob Storage, you will need to follow these steps:

1. **Create an Azure Communication Services Resource**.
2. **Set Up Azure Blob Storage**.
3. **Use ACS SDK to Capture Live Call Audio**.
4. **Save Audio to Azure Blob Storage**.

Here are detailed steps to achieve this:

### Step 1: Create an Azure Communication Services Resource

1. **Navigate to the Azure portal**.
2. **Create a new resource** and search for **Communication Services**.
3. **Follow the prompts** to create the service. Take note of the connection string, which you will need to authenticate.

### Step 2: Set Up Azure Blob Storage

1. **Navigate to the Azure portal**.
2. **Create a new resource** and search for **Storage Account**.
3. **Create a Blob Storage** account and a container to store the audio files. Take note of the connection string for authentication.

### Step 3: Use ACS SDK to Capture Live Call Audio

#### Install the Necessary SDKs

First, install the required NuGet packages:

sh

Copy code

dotnet add package Azure.Communication.Calling

dotnet add package Azure.Storage.Blobs

#### Create a New Project

Create a new .NET Console or ASP.NET Core project:

sh

Copy code

dotnet new console -n CallRecordingApp

cd CallRecordingApp

#### Capture and Save Audio

Here’s a simplified example to demonstrate capturing live call audio and saving it to Azure Blob Storage using the ACS SDK:

csharp

Copy code

using System;

using System.IO;

using System.Threading.Tasks;

using Azure;

using Azure.Communication.Calling;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace CallRecordingApp

{

class Program

{

private static readonly string communicationServicesConnectionString = "YourACSConnectionString";

private static readonly string storageConnectionString = "YourStorageConnectionString";

private static readonly string containerName = "audiocontainer";

static async Task Main(string[] args)

{

var callClient = new CallClient(communicationServicesConnectionString);

var callAgent = await callClient.CreateCallAgentAsync(new CallAgentOptions());

var call = await callAgent.JoinAsync(new GroupCallLocator("YourGroupCallId"), new JoinCallOptions());

call.OnRemoteParticipantsUpdated += async (sender, e) =>

{

foreach (var participant in e.AddedParticipants)

{

participant.OnMediaStreamReceived += async (sender, e) =>

{

var mediaStream = e.MediaStream;

if (mediaStream.MediaStreamType == MediaStreamType.Audio)

{

var audioStream = mediaStream as AudioMediaStream;

await CaptureAndSaveAudioAsync(audioStream);

}

};

}

};

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

private static async Task CaptureAndSaveAudioAsync(AudioMediaStream audioStream)

{

var blobServiceClient = new BlobServiceClient(storageConnectionString);

var containerClient = blobServiceClient.GetBlobContainerClient(containerName);

var blobClient = containerClient.GetBlobClient($"audio\_{Guid.NewGuid()}.wav");

using (var memoryStream = new MemoryStream())

{

// Assuming audioStream has a method to read audio data

await audioStream.ReadToStreamAsync(memoryStream);

memoryStream.Position = 0;

await blobClient.UploadAsync(memoryStream, new BlobHttpHeaders { ContentType = "audio/wav" });

}

Console.WriteLine("Audio saved to Blob Storage.");

}

}

}

### Step 4: Detailed Explanation

#### Azure Communication Services

1. **Create CallClient**: Initialize the CallClient with your ACS connection string.
2. **Create CallAgent**: Create a CallAgent to manage calls.
3. **Join Call**: Join a group call using JoinAsync.
4. **Handle Participants**: Attach event handlers to manage remote participants and their media streams.

#### Azure Blob Storage

1. **BlobServiceClient**: Initialize with your storage connection string.
2. **ContainerClient**: Get a reference to the container where audio files will be stored.
3. **BlobClient**: Create a unique BlobClient for each audio file.
4. **Save Audio**: Capture audio from the media stream and save it to the blob storage.

### Additional Considerations

* **Error Handling**: Implement proper error handling for network issues, storage failures, etc.
* **Authentication**: Use managed identities or secure ways to handle sensitive connection strings.
* **Scaling**: Ensure the application can handle multiple concurrent calls and participants efficiently.

### References

* [Azure Communication Services - SDK Documentation](https://docs.microsoft.com/en-us/azure/communication-services/quickstarts/voice-video-calling/getting-started-with-calling-sdk)
* [Azure Storage Blobs - SDK Documentation](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction)

This setup will help you capture live call audio using Azure Communication Services, save it to Azure Blob Storage, and then you can further process it for transcription or other purposes.

4o

DIRECTLY TO BLOB STORAGE : RESTAPI

To build a .NET Web API application that handles multiple concurrent calls, records audio, stores it in Azure Blob Storage, transcribes it, and then saves the transcription with the associated caller ID in a local database, follow these detailed steps.

### Overview

1. **Set Up Azure Services**: Azure Communication Services (for handling calls), Azure Blob Storage (for storing audio files), and Azure Speech Service (for transcribing audio).
2. **Create ASP.NET Core Web API**: Set up controllers and services to handle audio recording, storage, and transcription.
3. **Configure Local Database**: Set up SQL Server or another database to store transcriptions.

### Step-by-Step Implementation

#### Prerequisites

* Azure Subscription with Communication Services, Blob Storage, and Speech Service.
* SQL Server or another relational database for storing transcriptions.

#### Database Setup

Create a table to store the transcription details:

sql

Copy code

CREATE TABLE Transcriptions (

Id INT PRIMARY KEY IDENTITY(1,1),

CallId NVARCHAR(50) NOT NULL,

Name NVARCHAR(100) NOT NULL,

AudioFilePath NVARCHAR(MAX) NOT NULL,

Transcription NVARCHAR(MAX),

Timestamp DATETIME DEFAULT GETDATE()

);

#### Create ASP.NET Core Web API Project

1. **Create a New ASP.NET Core Web API Project** in Visual Studio.
2. **Install Necessary NuGet Packages**:

sh

Copy code

dotnet add package Azure.Communication.Calling

dotnet add package Azure.Storage.Blobs

dotnet add package Microsoft.CognitiveServices.Speech

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.AspNetCore.Http.Features

1. **Create Required Classes and Services**:

#### Program.cs

csharp

Copy code

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Hosting;

namespace LiveCallTranscriptionAPI

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

#### Startup.cs

csharp

Copy code

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.EntityFrameworkCore;

namespace LiveCallTranscriptionAPI

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddDbContext<TranscriptionContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddScoped<IAudioStorageService, AzureBlobStorageService>();

services.AddScoped<ISpeechService, AzureSpeechService>();

services.AddScoped<ITranscriptionRepository, TranscriptionRepository>();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

#### TranscriptionContext.cs

csharp

Copy code

using Microsoft.EntityFrameworkCore;

namespace LiveCallTranscriptionAPI

{

public class TranscriptionContext : DbContext

{

public TranscriptionContext(DbContextOptions<TranscriptionContext> options)

: base(options)

{

}

public DbSet<Transcription> Transcriptions { get; set; }

}

public class Transcription

{

public int Id { get; set; }

public string CallId { get; set; }

public string Name { get; set; }

public string AudioFilePath { get; set; }

public string TranscriptionText { get; set; }

public DateTime Timestamp { get; set; }

}

}

#### AzureBlobStorageService.cs

csharp

Copy code

using Azure.Storage.Blobs;

using System.IO;

using System.Threading.Tasks;

namespace LiveCallTranscriptionAPI

{

public interface IAudioStorageService

{

Task<string> UploadAudioAsync(Stream audioStream, string fileName);

}

public class AzureBlobStorageService : IAudioStorageService

{

private readonly string \_storageConnectionString = "YourStorageConnectionString";

private readonly string \_containerName = "audiocontainer";

public async Task<string> UploadAudioAsync(Stream audioStream, string fileName)

{

var blobServiceClient = new BlobServiceClient(\_storageConnectionString);

var containerClient = blobServiceClient.GetBlobContainerClient(\_containerName);

var blobClient = containerClient.GetBlobClient(fileName);

await blobClient.UploadAsync(audioStream, true);

return blobClient.Uri.ToString();

}

}

}

#### AzureSpeechService.cs

csharp

Copy code

using Microsoft.CognitiveServices.Speech;

using System;

using System.Threading.Tasks;

namespace LiveCallTranscriptionAPI

{

public interface ISpeechService

{

Task<string> TranscribeAudioAsync(string audioFileUri);

}

public class AzureSpeechService : ISpeechService

{

private readonly string \_subscriptionKey = "YourSpeechServiceSubscriptionKey";

private readonly string \_region = "YourSpeechServiceRegion";

public async Task<string> TranscribeAudioAsync(string audioFileUri)

{

var config = SpeechConfig.FromSubscription(\_subscriptionKey, \_region);

var audioConfig = AudioConfig.FromWavFileInput(audioFileUri);

var recognizer = new SpeechRecognizer(config, audioConfig);

var result = await recognizer.RecognizeOnceAsync();

if (result.Reason == ResultReason.RecognizedSpeech)

{

return result.Text;

}

else

{

throw new Exception($"Speech recognition failed. Reason: {result.Reason}");

}

}

}

}

#### TranscriptionRepository.cs

csharp

Copy code

using System.Threading.Tasks;

namespace LiveCallTranscriptionAPI

{

public interface ITranscriptionRepository

{

Task SaveTranscriptionAsync(Transcription transcription);

}

public class TranscriptionRepository : ITranscriptionRepository

{

private readonly TranscriptionContext \_context;

public TranscriptionRepository(TranscriptionContext context)

{

\_context = context;

}

public async Task SaveTranscriptionAsync(Transcription transcription)

{

\_context.Transcriptions.Add(transcription);

await \_context.SaveChangesAsync();

}

}

}

#### TranscriptionController.cs

csharp

Copy code

using Microsoft.AspNetCore.Mvc;

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

namespace LiveCallTranscriptionAPI.Controllers

{

[ApiController]

[Route("[controller]")]

public class TranscriptionController : ControllerBase

{

private readonly IAudioStorageService \_audioStorageService;

private readonly ISpeechService \_speechService;

private readonly ITranscriptionRepository \_transcriptionRepository;

public TranscriptionController(IAudioStorageService audioStorageService, ISpeechService speechService, ITranscriptionRepository transcriptionRepository)

{

\_audioStorageService = audioStorageService;

\_speechService = speechService;

\_transcriptionRepository = transcriptionRepository;

}

[HttpPost("transcribe")]

public async Task<IActionResult> Transcribe([FromForm] IFormFile audioFile, [FromForm] string callId, [FromForm] string name)

{

if (audioFile == null || string.IsNullOrEmpty(callId) || string.IsNullOrEmpty(name))

{

return BadRequest("Invalid input");

}

// Save audio file to local storage

string localFilePath = Path.GetTempFileName();

using (var fileStream = new FileStream(localFilePath, FileMode.Create))

{

await audioFile.CopyToAsync(fileStream);

}

// Save audio file to Azure Blob Storage

string blobUri = await \_audioStorageService.UploadAudioAsync(new FileStream(localFilePath, FileMode.Open), Path.GetFileName(localFilePath));

// Transcribe audio file using Azure Speech Service

string transcriptionText = await \_speechService.TranscribeAudioAsync(blobUri);

// Store transcription in database

var transcription = new Transcription

{

CallId = callId,

Name = name,

AudioFilePath = blobUri,

TranscriptionText = transcriptionText

};

await \_transcriptionRepository.SaveTranscriptionAsync(transcription);

return Ok(transcription);

}

}

}

#### appsettings.json

Update appsettings.json with your database connection string:

json

Copy code

{

"ConnectionStrings": {

"DefaultConnection": "YourDatabaseConnectionString"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

### Running the Application

1. **Run the Migration**: Ensure your database schema is up to date.

sh

Copy code

dotnet ef migrations add InitialCreate

dotnet ef database update

1. **Run the Application**:

sh

Copy code

dotnet run

1. **Test the API**: Use a tool like Postman to send a POST request to /transcribe with audioFile, callId, and name as form-data fields.

### Additional Considerations

* **Concurrency**: The design supports concurrent requests. Ensure Azure Blob Storage and Azure Speech Service handle concurrent operations effectively.
* **Error Handling**: Add comprehensive error handling and logging to manage failures in audio processing, storage, and transcription.
* **Security**: Implement proper authentication and authorization mechanisms to secure your API and sensitive data.