using System;

using System.Net;

using System.Net.Http;

using System.Net.Http.Formatting;

using System.Net.Http.Headers;

using System.Reflection.Metadata;

using System.Text.Json;

using System.Threading.Tasks;

using Azure;

using Newtonsoft.Json;

using Polly;

using Polly.Retry;

using qsecure\_intelligence.BlobAudio;

using qsecure\_intelligence.SpeechToText;

using Volo.Abp;

using JsonSerializer = System.Text.Json.JsonSerializer;

public class BatchClient :IDisposable

{

private const int MaxNumberOfRetries = 5;

private readonly HttpClient client;

private readonly string speechToTextBasePath;

private static AsyncRetryPolicy<HttpResponseMessage> transientFailureRetryingPolicy = Policy

.Handle<HttpRequestException>()

.Or<TaskCanceledException>()

.OrResult<HttpResponseMessage>(x => !x.IsSuccessStatusCode && (int) x.StatusCode == 429)

.WaitAndRetryAsync(MaxNumberOfRetries, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)), (result, timeSpan, retryCount, context) =>

{

Console.WriteLine($"Request failed with {result.Exception?.ToString() ?? result.Result?.StatusCode.ToString()}. Waiting {timeSpan} before next retry. Retry attempt {retryCount}");

});

private BatchClient(HttpClient Client)

{

client = Client;

}

public static BatchClient CreateApiV3Client(string key, string hostName)

{

var client = new HttpClient();

client.Timeout = TimeSpan.FromMinutes(25);

client.BaseAddress = new UriBuilder(Uri.UriSchemeHttps, hostName).Uri;

//client.BaseAddress = new Uri(hostName);

client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", key);

return new BatchClient(client);

}

private async Task<TResponse> PostAsJsonAsync<TPayLoad, TResponse>(string path, TPayLoad payload)

{

//string json = JsonConvert.SerializeObject(payload, SpeechJsonContractResolver.WriterSettings);

// var JsonOutput = JsonConvert.SerializeObject(new { jsonCreditApplication = payload});

var options = new JsonSerializerOptions { WriteIndented = true };

string json = JsonSerializer.Serialize(payload, options);

StringContent content = new StringContent(json);

content.Headers.ContentType = JsonMediaTypeFormatter.DefaultMediaType;

content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

try

{

var response = await

transientFailureRetryingPolicy

.ExecuteAsync(() =>

client.PostAsync(path, content))

.ConfigureAwait(false);

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

Console.WriteLine(responseContent); // Log response for debugging

if (response.IsSuccessStatusCode)

{

//var options1 = new JsonSerializerOptions { WriteIndented = true };

//var content1 = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

//var result = JsonConvert.DeserializeObject<TResponse>(responseContent, SpeechJsonContractResolver.ReaderSettings);

var result = JsonSerializer.Deserialize<TResponse>(responseContent);

//r result = JsonConvert.DeserializeObject<TResponse>(responseContent);

return result;

// return await response.Content.ReadAsAsync<TResponse>(

// new[]

// {

// new JsonMediaTypeFormatter

// {

// SerializerSettings = SpeechJsonContractResolver.ReaderSettings

// }

// }).ConfigureAwait(false);

//}

}

throw await CreateExceptionAsync(response).ConfigureAwait(false);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

private async Task<TResponse> GetAsync<TResponse>(string path)

{

try

{

var response = await transientFailureRetryingPolicy

.ExecuteAsync(async () => await client.GetAsync(path).ConfigureAwait(false))

.ConfigureAwait(false);

if (response.IsSuccessStatusCode)

{

// var result = await response.Content.ReadAsAsync<TResponse>().ConfigureAwait(false);

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

var result = JsonSerializer.Deserialize<TResponse>(responseContent);

return result;

}

throw await CreateExceptionAsync(response);

}

catch (Exception ex) {

throw new UserFriendlyException(ex.Message);

}

}

private static async Task<FailedHttpClientRequestException> CreateExceptionAsync(HttpResponseMessage response)

{

switch (response.StatusCode)

{

case HttpStatusCode.Forbidden:

return new FailedHttpClientRequestException(response.StatusCode, "No permission to access this resource.");

case HttpStatusCode.Unauthorized:

return new FailedHttpClientRequestException(response.StatusCode, "Not authorized to see the resource.");

case HttpStatusCode.NotFound:

return new FailedHttpClientRequestException(response.StatusCode, "The resource could not be found.");

case HttpStatusCode.UnsupportedMediaType:

return new FailedHttpClientRequestException(response.StatusCode, "The file type isn't supported.");

case HttpStatusCode.BadRequest:

{

var content = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

var shape = new { Message = string.Empty };

var result = JsonConvert.DeserializeAnonymousType(content, shape);

if (result != null && !string.IsNullOrEmpty(result.Message))

{

return new FailedHttpClientRequestException(response.StatusCode, result.Message);

}

return new FailedHttpClientRequestException(response.StatusCode, response.ReasonPhrase);

}

default:

return new FailedHttpClientRequestException(response.StatusCode, response.ReasonPhrase);

}

}

public Task<PaginatedTranscriptions> GetTranscriptionsAsync()

{

try

{

var path = "speechtotext/v3.2/transcriptions/";

return GetAsync<PaginatedTranscriptions>(path);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public Task<PaginatedTranscriptions> GetTranscriptionsAsync(Uri location)

{

return GetAsync<PaginatedTranscriptions>(location.PathAndQuery);

}

public Task<PaginatedFiles> GetTranscriptionFilesAsync(Uri location)

{

try {

if (location == null)

{

throw new ArgumentNullException(nameof(location));

}

return GetAsync<PaginatedFiles>(location.PathAndQuery);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public Task<Transcription> GetTranscriptionAsync(Uri location)

{

try {

if (location == null)

{

throw new ArgumentNullException(nameof(location));

}

return GetAsync<Transcription>(location.PathAndQuery);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public async Task<RecognitionResults> GetTranscriptionResultAsync(Uri location)

{

try {

if (location == null)

{

throw new ArgumentNullException(nameof(location));

}

var response = await transientFailureRetryingPolicy

.ExecuteAsync(async () => await this.client.GetAsync(location).ConfigureAwait(false))

.ConfigureAwait(false);

if (response.IsSuccessStatusCode)

{

var json = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

return JsonConvert.DeserializeObject<RecognitionResults>(json, SpeechJsonContractResolver.ReaderSettings);

}

throw await CreateExceptionAsync(response);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public async Task<Transcription> CreateTranscriptionAsync(Transcription transcription)

{

try {

if (transcription == null)

{

throw new ArgumentNullException(nameof(transcription));

}

var path = "speechtotext/v3.2/transcriptions/";

Transcription newJson = await PostAsJsonAsync<Transcription, Transcription>(path, transcription);

string jsonType = null;

string jsonType2 = null;

return newJson;

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public Task DeleteTranscriptionAsync(Uri location)

{

try {

if (location == null)

{

throw new ArgumentNullException(nameof(location));

}

return

// transientFailureRetryingPolicy

//.ExecuteAsync(() =>

client.DeleteAsync(location.PathAndQuery);

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public void Dispose()

{

client?.Dispose();

}

}

using Azure;

using Azure.Storage.Blobs;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Configuration;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Volo.Abp;

using Volo.Abp.Application.Services;

using Volo.Abp.DependencyInjection;

using System.IO.Pipes;

using Volo.Abp.Data;

using Azure.Storage.Sas;

using System.Net.Http;

using Azure.Storage;

using System.Reflection.Metadata.Ecma335;

using Newtonsoft.Json;

using qsecure\_intelligence.SpeechToText;

using System.Threading.Channels;

namespace qsecure\_intelligence.BlobAudio

{

public class BlobAudioService : ApplicationService, ITransientDependency

{

private IConfiguration \_configuration;

public BlobAudioService(IConfiguration configuration)

{

\_configuration = configuration;

}

public async Task<string> CreateContainer(string cName)

{

try

{

if (cName.Where(char.IsUpper).Any())

{

throw new NotSupportedException($"{cName} should not contain upper case letters");

}

string? connectionString = \_configuration.GetValue<string>("AzureStorageSpeechAI:Connectionstring");

var blobServiceClient = new BlobServiceClient(connectionString);

var container = blobServiceClient.GetBlobContainerClient(cName);

if (!container.Exists())

{

BlobContainerClient containerClient = await blobServiceClient.CreateBlobContainerAsync(cName);

return "Container created successfully !";

}

else

{

throw new UserFriendlyException($"{cName} container already exist in the Storage Account");

}

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public async Task<string> UploadBlob(IFormFile file, string cName)

{

try

{

if (cName.Where(char.IsUpper).Any())

{

throw new NotSupportedException($"{cName} should not contain upper case letters only");

}

string? connectionString = \_configuration.GetValue<string>("AzureStorageSpeechAI:Connectionstring");

string? SubscriptionKey = \_configuration.GetValue<string>("AzureSpeechAI:Key1");

string? Region = \_configuration.GetValue<string>("AzureSpeechAI:Region");

var blobServiceClient = new BlobServiceClient(connectionString);

var container = blobServiceClient.GetBlobContainerClient(cName);

if (container.Exists())

{

if (file == null || file.Length == 0)

{

return "No file uploaded.";

}

using (var stream = file.OpenReadStream())

{

var fileName = $"\_{file.FileName}";

var blobClient = container.GetBlobClient(fileName);

if (!blobClient.Exists())

{

await blobClient.UploadAsync(stream, overwrite: true);

var blobUrl = blobClient.Uri.AbsoluteUri;

string blobname = blobClient.Name;

string sasUri = GenerateSasUrl(blobClient, cName, blobname,blobUrl);

//using (var client = BatchClient.CreateApiV3Client(SubscriptionKey, $"{Region}.api.cognitive.microsoft.com"))

//{

var result= await TranscribeAsync(sasUri).ConfigureAwait(false);

// }

return sasUri;

}

else

{

return "Blob already exist";

}

}

}

return "Container doesnot Exist";

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public async Task<string> DeleteBlob(string url, string cName)

{

try

{

if (cName.Where(char.IsUpper).Any())

{

throw new NotSupportedException($"{cName} should not contain upper case letters only");

}

string? connectionString = \_configuration.GetValue<string>("AzureStorageSpeechAI:Connectionstring");

var blobServiceClient = new BlobServiceClient(connectionString);

BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(cName);

string[] QueryArray = url.Split('/');

string lastWord = QueryArray[QueryArray.Length - 1];

lastWord = lastWord.Replace("%20", " ");

if (lastWord.Contains("%28") || lastWord.Contains("%29"))

{

lastWord = lastWord.Replace("%29", ")");

lastWord = lastWord.Replace("%28", "(");

}

containerClient.DeleteBlobIfExists(lastWord);

return "Blob Deleted Succesfully";

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

private string GenerateSasUrl(BlobClient blobClient, string cName, string blobname, string url)

{

string? StorageName= \_configuration.GetValue<string>("AzureStorageSpeechAI:StorageAccountName");

string? StorageKey = \_configuration.GetValue<string>("AzureStorageSpeechAI:StorageAccountKey");

try

{

if (!blobClient.CanGenerateSasUri)

{

throw new InvalidOperationException("Blob client cannot generate a SAS URI.");

}

var sasBuilder = new BlobSasBuilder

{

BlobContainerName = cName,

BlobName = blobname,

Resource = "b",

Protocol = SasProtocol.Https,

ExpiresOn = DateTimeOffset.UtcNow.AddHours(1)

};

// Set permissions (read permission to allow downloading)

sasBuilder.SetPermissions(BlobContainerSasPermissions.Read);

var sasToken = sasBuilder.ToSasQueryParameters(new StorageSharedKeyCredential(StorageName,StorageKey)).ToString();

sasToken = sasToken.Replace("%3A", ":");

var sasUrl = url + "?" + sasToken;

return sasUrl;

}

catch (Exception ex) {

throw new UserFriendlyException(ex.Message);

}

}

public async Task<string> DeleteContainer(string cName)

{

try

{

if (cName.Where(char.IsUpper).Any())

{

throw new NotSupportedException($"{cName} should not contain upper case letters");

}

string? connectionString = \_configuration.GetValue<string>("AzureStorageSpeechAI:Connectionstring");

var blobServiceClient = new BlobServiceClient(connectionString);

var container = blobServiceClient.GetBlobContainerClient(cName);

if (!container.Exists())

{

return "Container doesnot Exist, Check the name!";

}

else

{

await blobServiceClient.DeleteBlobContainerAsync(cName);

return "The Conatainer Deleted Succesfully";

}

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

public async Task<Transcription> TranscribeAsync(string sasUri)

{

//Console.WriteLine("Deleting all existing completed transcriptions.");

string? SubscriptionKey = \_configuration.GetValue<string>("AzureSpeechAI:Key1");

string? Region = \_configuration.GetValue<string>("AzureSpeechAI:Region");

using (var client = BatchClient.CreateApiV3Client(SubscriptionKey, $"{Region}.api.cognitive.microsoft.com"))

{

try

{

//// get all transcriptions for the subscription

PaginatedTranscriptions paginatedTranscriptions = null;

string? Locale = "en-US";

Uri RecordingsBlobUri = new Uri(sasUri);

EntityReference model =

new EntityReference { Self = new Uri("https://eastus.api.cognitive.microsoft.com/speechtotext/v3.2/models/base/e418c4a9-9937-4db7-b2c9-8afbff72d950") };

Uri ContainerUri = new Uri("https://rndsoftspeechaudios.blob.core.windows.net/agent-insurance-audios");

string DisplayName = "Rnd Transcription";

//do

//{

// if (paginatedTranscriptions == null)

// {

// paginatedTranscriptions = await client.GetTranscriptionsAsync().ConfigureAwait(false);

// }

// else

// {

// paginatedTranscriptions = await client.GetTranscriptionsAsync(paginatedTranscriptions.NextLink).ConfigureAwait(false);

// }

// // delete all pre-existing completed transcriptions. If transcriptions are still running or not started, they will not be deleted

// foreach (var transcriptionToDelete in paginatedTranscriptions.Values)

// {

// // delete a transcription

// await client.DeleteTranscriptionAsync(transcriptionToDelete.Self).ConfigureAwait(false);

// Console.WriteLine($"Deleted transcription {transcriptionToDelete.Self}");

// }

//}

//while (paginatedTranscriptions.NextLink != null);

// <transcriptiondefinition>

var newTranscription = new Transcription

{

DisplayName = DisplayName,

Locale = Locale,

ContentUrls = new[] { RecordingsBlobUri },

// ContentContainerUrl=ContainerUri,

//Model = model,

//Properties = new TranscriptionProperties

//{

// TimeToLive = TimeSpan.FromDays(1),

// IsWordLevelTimestampsEnabled = true,

// IsDisplayFormWordLevelTimestampsEnabled = false,

// Channels = new[] { 1 },

// IsDiarizationEnabled = true,

// Diarization = new DiarizationProperties

// {

// Speakers = new DiarizationSpeakersProperties

// {

// MinCount = 1,

// MaxCount = 2

// }

// }

//}

};

newTranscription = await client.CreateTranscriptionAsync(newTranscription).ConfigureAwait(false);

return newTranscription;

//Console.WriteLine($"Created transcription {newTranscription.Self}");

//// </transcriptiondefinition>

//// get the transcription Id from the location URI

//var createdTranscriptions = new List<Uri> { newTranscription.Self };

//Console.WriteLine("Checking status.");

//// get the status of our transcriptions periodically and log results

//int completed = 0, running = 0, notStarted = 0;

//while (completed < 1)

//{

// completed = 0; running = 0; notStarted = 0;

// // get all transcriptions for the user

// paginatedTranscriptions = null;

// do

// {

// // <transcriptionstatus>

// if (paginatedTranscriptions == null)

// {

// paginatedTranscriptions = await client.GetTranscriptionsAsync().ConfigureAwait(false);

// }

// else

// {

// paginatedTranscriptions = await client.GetTranscriptionsAsync(paginatedTranscriptions.NextLink).ConfigureAwait(false);

// }

// // delete all pre-existing completed transcriptions. If transcriptions are still running or not started, they will not be deleted

// foreach (var transcription in paginatedTranscriptions.Values)

// {

// switch (transcription.Status)

// {

// case "Failed":

// case "Succeeded":

// // we check to see if it was one of the transcriptions we created from this client.

// if (!createdTranscriptions.Contains(transcription.Self))

// {

// // not created form here, continue

// continue;

// }

// completed++;

// // if the transcription was successful, check the results

// if (transcription.Status == "Succeeded")

// {

// var paginatedfiles = await client.GetTranscriptionFilesAsync(transcription.Links.Files).ConfigureAwait(false);

// var resultFile = paginatedfiles.Values.FirstOrDefault(f => f.Kind == ArtifactKind.Transcription);

// var result = await client.GetTranscriptionResultAsync(new Uri(resultFile.Links.ContentUrl)).ConfigureAwait(false);

// //Console.WriteLine("Transcription succeeded. Results: ");

// //Console.WriteLine(JsonConvert.SerializeObject(result, SpeechJsonContractResolver.WriterSettings));

// var TextObject = JsonConvert.SerializeObject(result, SpeechJsonContractResolver.WriterSettings);

// return TextObject;

// }

// else

// {

// return ("Transcription failed. Status: {0}", transcription.Properties.Error.Message).ToString();

// }

// case "Running":

// running++;

// break;

// case "NotStarted":

// notStarted++;

// break;

// }

// }

// // for each transcription in the list we check the status

// Console.WriteLine(string.Format("Transcriptions status: {0} completed, {1} running, {2} not started yet", completed, running, notStarted));

// }

// while (paginatedTranscriptions.NextLink != null);

// // </transcriptionstatus>

// // check again after 1 minute

// await Task.Delay(TimeSpan.FromMinutes(1)).ConfigureAwait(false);

//}

////Console.WriteLine("Press any key...");

////Console.ReadKey();

//return "Transcription Ended";

}

catch (Exception ex)

{

throw new UserFriendlyException(ex.Message);

}

}

}

}

}

https://rndsoftspeechaudios.blob.core.windows.net/test-container/\_2.wav?sv=2024-11-04&spr=https&se=2024-09-27T13:24:43Z&sr=b&sp=r&sig=aV6l7b1VvpPAJ43XHB1TDbT8DyVjuqbR1CSjk9q7M9s%3D