Lab 03. Use Text and Image skills with Azure Search using VS Code

Objective

In this lab you are going to create a console application to call Azure Search and integrate text and image skills from cognitive services to get more information of the indexes and images.

Architecture

Prerequisites

- Complete **Lab 2**.
- Visual Studio Code 2017 or later.

Steps

Work with Azure Search Service

Create projects

- Open target folder.
    - Create a folder where you are going to create your project (Ej. C:\My Project).
    - Open VS Code.
    - Go to File -> Folder.
    - Navigate to the folder created in previous steps.
    - Click on Select Folder.
- Create a solution.
    - Open the terminal.
        - Go to Terminal -> New Terminal.
    - Create the solution.

```
dotnet new sln
```

- Create a console project.

```
dotnet new console -o AzureSearchKnowledgeConsole
```

- Create a library project in the same solution.

```
dotnet new classlib -o AzureSearchKnowledge
```

- Add the Microsoft.Azure.Search version 10.1.0 package to the AzureSearchKnowledge project:
    - Go to the library path.

```
cd AzureSearchKnowledge
```

    - Add package to work with Azure Search Service.

```
dotnet add package Microsoft.Azure.Search -v 10.1.0
```

    - Add the next package to work with the model

```
dotnet add package System.ComponentModel.Annotations -v 4.7.0
```

    - Add the next package to work with Azure

```
dotnet add package Azure.Core -v 1.5.1
```

    - Build the project.

```
dotnet build
```

Create Model class

- Expand the AzureSearchKnowledge project from the Explorer in the left menu.
- Create the new **DocumentModel** class.

    - Right click on the AzureSearchKnowledge project.
    - Select New File and name it DocumentModel.cs.
- Add the following namespaces.

```
using Microsoft.Azure.Search;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
```

- Create the **AzureSearchKnowledge** namespace and the **DocumentModel** class.

```
namespace AzureSearchKnowledge

{
    public class DocumentModel
    {

    }
}
```

- Add the Fields.

NOTE: There are only the fields with the **retrieve** property and including the properties base on the index you created in Azure Search.

```
[Key]
public string metadata_storage_path { get; set; }
public string metadata_storage_name { get; set; }
public string content { get; set; }
[IsFilterable, IsFacetable, IsSearchable]
public IEnumerable<string> people { get; set; }
[IsFilterable, IsFacetable, IsSearchable]
public IEnumerable<string> locations { get; set; }
[IsFilterable, IsFacetable, IsSearchable]
public IEnumerable<string> organizations { get; set; }
[IsFilterable, IsFacetable, IsSearchable]
public IEnumerable<string> keyPhrases { get; set; }
[IsSearchable]
public string translated_text { get; set; }
[IsSearchable]
public string merged_content { get; set; }
[IsSearchable]
public IEnumerable<string> text { get; set; }
[IsSearchable]
public IEnumerable<string> layoutText { get; set; }
[IsSearchable]
public IEnumerable<string> imageTags { get; set; }
[IsSearchable]
public IEnumerable<string> imageCaption { get; set; }
[IsSearchable]
public IEnumerable<string> imageCelebrities { get; set; }
```

- The class should look like the next code:

```
using Microsoft.Azure.Search;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
namespace AzureSearchKnowledge
```

```
{
    public class DocumentModel
    {
        [Key]
        public string metadata_storage_path { get; set; }
        public string metadata_storage_name { get; set; }
        public string content { get; set; }
        [IsFilterable, IsFacetable, IsSearchable]
        public IEnumerable<string> people { get; set; }
        [IsFilterable, IsFacetable, IsSearchable]
        public IEnumerable<string> locations { get; set; }
        [IsFilterable, IsFacetable, IsSearchable]
        public IEnumerable<string> organizations { get; set; }
        [IsFilterable, IsFacetable, IsSearchable]
        public IEnumerable<string> keyPhrases { get; set; }
        [IsSearchable]
        public string translated_text { get; set; }
        [IsSearchable]
        public string merged_content { get; set; }
        [IsSearchable]
        public IEnumerable<string> text { get; set; }
        [IsSearchable]
        public IEnumerable<string> layoutText { get; set; }
        [IsSearchable]
        public IEnumerable<string> imageTags { get; set; }
        [IsSearchable]
        public IEnumerable<string> imageCaption { get; set; }
        [IsSearchable]
        public IEnumerable<string> imageCelebrities { get; set; }
    }
}
```

Create Azure Search class

- Expand the AzureSearchKnowledge project from the Explorer in the left menu.
- Change the Class1.cs name to AzureCognitiveSearch.cs
- Modify the class **AzureCognitiveSearch.cs**.
    - Change the class modifier to **public** and the name to AzureCognitiveSearch.

```
public class AzureCognitiveSearch
```

    - Create namespace on top of the class and close it at the end of the docuemnt

```
namespace AzureSearchKnowledge
{
    public class AzureCognitiveSearch
    {
    }
}
```

    - Add the namespaces on the top.

```
using Microsoft.Azure.Search;
using Microsoft.Azure.Search.Models;
using System.Collections.Generic;
using Azure;
```

- Add constants and variables to connect to the services after opening the class. Change the values between the [], removing those too.

```
const string myIndex = "[Index Name]";
const string searchServiceName = "[Azure Search Service Name]";
const string searchApiKey = "[Search API Key]";
const string StorageContainerAddress = "https://[Storage Account
Name].blob.core.windows.net/[Container Name]";
const string StorageAccountName = "[Storage Account Name]";
const string cognitiveServiceKey = "[Cognitive Service Key]";
const string StorageAccountKey = "[Storage Account Key]";
private static readonly AzureKeyCredential credentials = new AzureKeyCredentia
l(cognitiveServiceKey);
private static readonly Uri endpoint= new Uri("[Cognitive Service Endpoint]");
```

- After the variables add the method to search documents.

```
public IEnumerable<SearchResult<DocumentModel>> SearchingQueries(string query)
{
    ISearchIndexClient indexClientForQueries = new SearchIndexClient(searchSer
viceName, myIndex, new SearchCredentials(searchApiKey));

    SearchParameters parameters = new SearchParameters();
    DocumentSearchResult<DocumentModel> results = indexClientForQueries.Docume
nts.Search<DocumentModel>(query, parameters);

    return results.Results;
}
```

- Save the document.
- Go back to the Terminal and build the project.

```
dotnet build
```

- The class should look like the next code:

```
using System;
using Microsoft.Azure.Search;
using Microsoft.Azure.Search.Models;
using System.Collections.Generic;
using Azure;

namespace AzureSearchKnowledge
{
    public class AzureCognitiveSearch
    {
        const string myIndex = "[Index Name]";
        const string searchServiceName = "[Azure Search Service Name]";
        const string searchApiKey = "[Search API Key]";
        const string StorageContainerAddress = "https://[Storage Account
Name].blob.core.windows.net/[Container Name]";
        const string StorageAccountName = "[Storage Account Name]";
        const string cognitiveServiceKey = "[Cognitive Service Key]";
        const string StorageAccountKey = "[Storage Account Key]";
        private static readonly AzureKeyCredential credentials = new AzureKeyCredential(cog
nitiveServiceKey);
        private static readonly Uri endpoint= new Uri("[Cognitive Service Endpoint]");

        public IEnumerable<SearchResult<DocumentModel>> SearchingQueries(string query)
        {
            ISearchIndexClient indexClientForQueries = new SearchIndexClient(searchServiceN
ame, myIndex, new SearchCredentials(searchApiKey));

            SearchParameters parameters = new SearchParameters();
            DocumentSearchResult<DocumentModel> results = indexClientForQueries.Documents.S
earch<DocumentModel>(query, parameters);

            return results.Results;
        }
    }
}
```

Call Search service from console project

- Expand the AzureSearchKnowledgeConsole project from the Explorer in the left menu.
- Open the class program.cs.
  - Add namespace from the console project.

```
using AzureSearchKnowledge;
using System.Collections.Generic;
using Microsoft.Azure.Search.Models;
```

  - Remove everything within the Main method.
  - Create an object of the AzureCognitiveSearch class within the Main method.

```
AzureCognitiveSearch ccs = new AzureCognitiveSearch();
```

  - Call the SearchQueries method with the text to search, for example the next code searches by **CEO**.

```
var results = ccs.SearchingQueries("CEO");
```

  - Call the method to display the result.

```
DisplayResult(results);
```

  - Create the method to display the result after the Main method.

```
static void DisplayResult(IEnumerable<SearchResult<DocumentModel>> results)
{
    foreach (SearchResult<DocumentModel> result in results)
    {
        Console.WriteLine($"metadata_storage_path: {result.Document.metadata_s
torage_path}, metadata_storage_name: {result.Document.metadata_storage_name}"
+
            $"content: {result.Document.content}, translated_text: {result.Doc
ument.translated_text}, merged_content: {result.Document.merged_content}" +
            $"");
        Console.WriteLine("People: ");
        foreach (string p in result.Document.people)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Locations: ");
        foreach (string p in result.Document.locations)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Organizations: ");
        foreach (string p in result.Document.organizations)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Key Phrases: ");
        foreach (string p in result.Document.keyPhrases)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Text: ");
        foreach (string p in result.Document.text)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Layout Text: ");
        foreach (string p in result.Document.layoutText)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Image Tags: ");
        foreach (string p in result.Document.imageTags)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Image Caption: ");
        foreach (string p in result.Document.imageCaption)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine("Image Celebrities: ");
        foreach (string p in result.Document.imageCelebrities)
        {
            Console.WriteLine($"\t{p}");
        }
        Console.WriteLine();
    }
}
```

- Save the document.
- Go back to the Terminal and access to the main path.

```
cd..
```

- Add the library reference.

```
dotnet  add AzureSearchKnowledgeConsole/AzureSearchKnowledgeConsole.csproj reference
AzureSearchKnowledge/AzureSearchKnowledge.csproj
```

- Access to the console project path.

```
cd AzureSearchKnowledgeConsole
```

- Build the project.

```
dotnet build
```

  - The class should look like the next code:

```
using System;
using AzureSearchKnowledge;
using System.Collections.Generic;
using Microsoft.Azure.Search.Models;

namespace AzureSearchKnowledgeConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            AzureCognitiveSearch ccs = new AzureCognitiveSearch();
            var results = ccs.SearchingQueries("CEO");
            DisplayResult(results);
        }
        static void DisplayResult(IEnumerable<SearchResult<DocumentModel>> results)
        {
            foreach (SearchResult<DocumentModel> result in results)
            {
                Console.WriteLine($"metadata_storage_path: {result.Document.metadata_storag
e_path}, metadata_storage_name: {result.Document.metadata_storage_name}" +
                    $"content: {result.Document.content}, translated_text: {result.Document
.translated_text}, merged_content: {result.Document.merged_content}" +
                    $"");
                Console.WriteLine("People: ");
                foreach (string p in result.Document.people)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Locations: ");
                foreach (string p in result.Document.locations)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Organizations: ");
                foreach (string p in result.Document.organizations)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Key Phrases: ");
                foreach (string p in result.Document.keyPhrases)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Text: ");
                foreach (string p in result.Document.text)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Layout Text: ");
                foreach (string p in result.Document.layoutText)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Image Tags: ");
                foreach (string p in result.Document.imageTags)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Image Caption: ");
                foreach (string p in result.Document.imageCaption)
                {
                    Console.WriteLine($"\t{p}");
                }
                Console.WriteLine("Image Celebrities: ");
                foreach (string p in result.Document.imageCelebrities)
                {
                    Console.WriteLine($"\t{p}");
                  }
                  Console.WriteLine();
              }
          }
      }
}
```

Test the application.

- Back to the terminal run the console project.

```
dotnet run
```

- Explore the results.

NOTE: You can change the text to search and the output to print.

Work with Text skill

Analyze text

- Go to the terminal and access to the AzureSearchKnowledge path.
- Install the package to work with Text Analytics.

```
dotnet add package Azure.AI.TextAnalytics -v 5.0.0
```

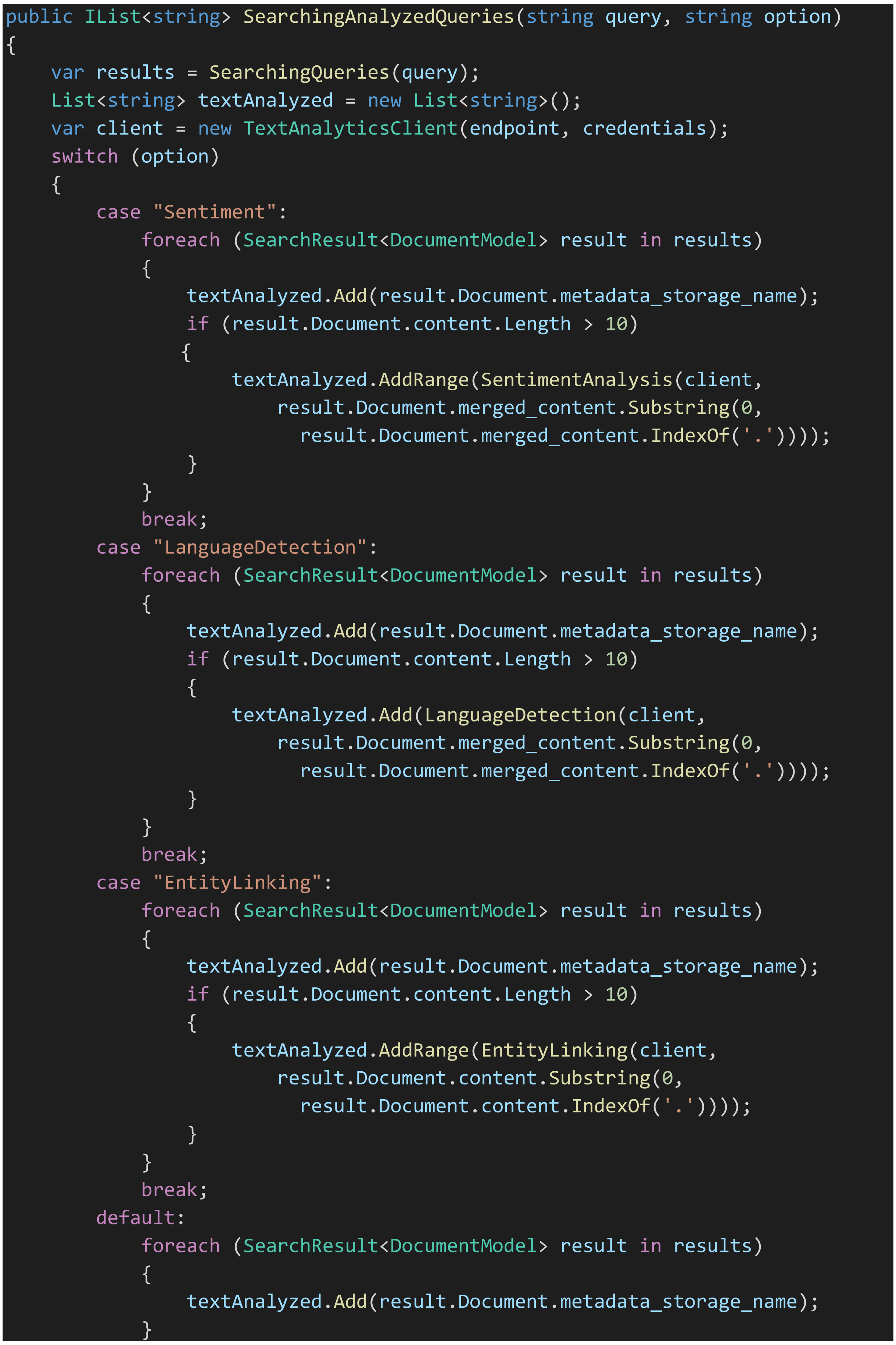
- Build the project.

```
dotnet build
```

- Open the class AzureCognitiveSearch and add the namespace on the top of the document.

```
using Azure.AI.TextAnalytics;
```

  - Create the method to analyze the text after the SearchQueries method.



```csharp
public IList<string> SearchingAnalyzedQueries(string query, string option)
{
    var results = SearchingQueries(query);
    List<string> textAnalyzed = new List<string>();
    var client = new TextAnalyticsClient(endpoint, credentials);
    switch (option)
    {
        case "Sentiment":
            foreach (SearchResult<DocumentModel> result in results)
            {
                textAnalyzed.Add(result.Document.metadata_storage_name);
                if (result.Document.content.Length > 10)
               {
                    textAnalyzed.AddRange(SentimentAnalysis(client,
                        result.Document.merged_content.Substring(0,
                          result.Document.merged_content.IndexOf('.'))));
                }
            }
            break;
        case "LanguageDetection":
            foreach (SearchResult<DocumentModel> result in results)
            {
                textAnalyzed.Add(result.Document.metadata_storage_name);
                if (result.Document.content.Length > 10)
                {
                    textAnalyzed.Add(LanguageDetection(client,
                        result.Document.merged_content.Substring(0,
                          result.Document.merged_content.IndexOf('.'))));
                }
            }
            break;
        case "EntityLinking":
            foreach (SearchResult<DocumentModel> result in results)
            {
                textAnalyzed.Add(result.Document.metadata_storage_name);
                if (result.Document.content.Length > 10)
                {
                    textAnalyzed.AddRange(EntityLinking(client,
                        result.Document.content.Substring(0,
                          result.Document.content.IndexOf('.'))));
                }
            }
            break;
        default:
            foreach (SearchResult<DocumentModel> result in results)
            {
                textAnalyzed.Add(result.Document.metadata_storage_name);
            }
            break;
    }
    return textAnalyzed;
}
```

- Add the method to detect the sentiment after the SearchingAnalyzedQueries method.

```
static IList<string> SentimentAnalysis(TextAnalyticsClient client,
string inputText)
{
    if (inputText == string.Empty)
        inputText = "NA";
    IList<string> result = new List<string>();
    DocumentSentiment documentSentiment = client.AnalyzeSentiment(inputText);
    result.Add($"Document sentiment: {documentSentiment.Sentiment}\n");
    foreach (var sentence in documentSentiment.Sentences)
    {
        result.Add($"\tText: \"{sentence.Text}\" \n" +
            $"\tSentence sentiment: {sentence.Sentiment} \n" +
            $"\tPositive score: {sentence.ConfidenceScores.Positive:0.00} \n"+
            $"\tNegative score: {sentence.ConfidenceScores.Negative:0.00} \n"+
            $"\tNeutral score: {sentence.ConfidenceScores.Neutral:0.00}\n");
    }
    return result;
}
```

- Add the method to detect the language after the SentimentAnalysis method.

```
static string LanguageDetection(TextAnalyticsClient client, string inputText)
{
    if (inputText == string.Empty)
        inputText = "NA";
    DetectedLanguage detectedLanguage = client.DetectLanguage(inputText);
    return $"Language:\t{detectedLanguage.Name},\t ISO6391:
{detectedLanguage.Iso6391Name}\n";
}
```

- Add the method to detect entity linking after the LanguageDetection.

```
static IList<string> EntityLinking(TextAnalyticsClient client, string
inputText)
{
    if (inputText == string.Empty)
        inputText = "NA";
    IList<string> result = new List<string>();
    string matches;
    var response = client.RecognizeLinkedEntities(inputText);
    foreach (var entity in response.Value)
    {
        matches = "";
        foreach (var match in entity.Matches)
        {
            matches = matches + ($"\tText: {match.Text}, Score: {match.Confide
nceScore:F2} \n");
        }
        result.Add($"Name: {entity.Name},\tID: {entity.DataSourceEntityId},\tU
RL: {entity.Url}\tData Source: {entity.DataSource} \n Matcher: \n {matches}");
    }
    return result;
}
```

- Save the document.
- Open the terminal and build the project.

```
dotnet build
```

Consume analyzed text methods

- Open the program.cs from the console project.
- Remove the following lines from the Main method.

```
var results = ccs.SearchingQueries("CEO");
DisplayResult(results);
```

- Call the SearchingAnalyzedQueries method with the text to search, for example the next code searches for **Microsoft** and it gets the **sentiment**.

```
var result = ccs.SearchingAnalyzedQueries("Microsoft", "Sentiment");
```

- Create the following loop to print the result of the analysis:

```
foreach (string s in result)
{
    Console.WriteLine(s);
}
```

- Save the file.
- Access to the console project path.

```
cd AzureSearchKnowledgeConsole
```

- Build the project.

```
dotnet build
```

Test the application.

- Back to the terminal run the console project.

```
dotnet run
```

- Explore the results.

NOTE: You can change the text to search and the output to print. Options to analyze the text are:

  - Sentiment
  - EntitityLinking
  - LanguageDetection

Work with Image skill

Analyze image

- Go to the terminal and access to the AzureSearchKnowledge path.
- Install the package to work with Storage Accounts.

```
dotnet add package Azure.Storage.Blobs -v 12.6.0
```

- Install the package to work with Computer Vision.

```
dotnet add package Microsoft.Azure.CognitiveServices.Vision.ComputerVision -v
5.0.0
```

- Build the project.

```
dotnet build
```

- Open the AzureCognitiveSearch class.
- Add the namespaces to the top of the file.

```
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Sas;
using System.Net;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;
using System.Threading.Tasks;
```

- After the variables add the features list to get from the image.

```
private static readonly List<VisualFeatureTypes> features = new List<VisualFea
tureTypes>()
{
    VisualFeatureTypes.Categories,
    VisualFeatureTypes.Description,
    VisualFeatureTypes.Faces,
    VisualFeatureTypes.Objects,
    VisualFeatureTypes.Color
};
```

- Add the method ImageFeatureAsync after the EntityLinking method, where it calls the SearchingQueries method and from the result analyses only the images. To analyze each image, it gets the URI with the SAS of the container.

```
public async Task ImageFeaturesAsync(string query)
{
    var sas = GetContainerSas();
    var results = SearchingQueries(query);
    ComputerVisionClient computerVision = new ComputerVisionClient(
        new ApiKeyServiceClientCredentials(cognitiveServiceKey),
        new System.Net.Http.DelegatingHandler[] { });
    computerVision.Endpoint = endpoint.ToString();
    string[] images = { ".jpg", ".png" };

    foreach (SearchResult<DocumentModel> result in results)
    {
        if (Array.Exists(images, E => E == result.Document.metadata_storage_na
me.Substring(result.Document.metadata_storage_name.Length - 4)))
        {
            string uri = StorageContainerAddress + "/" + result.Document.metad
ata_storage_name + sas;
            var imageFeatures = await AnalyzeRemoteAsync(computerVision, uri,
result.Document.metadata_storage_name);
            foreach (var imageFeature in imageFeatures)
            {
                Console.WriteLine(imageFeature);
            }
        }
    }
}
```

- Add the method GetContainersSas after the ImageFeatureAsync method. This method generates the SAS token for the container with 2 hours expiration time and read permissions.

```
private string GetContainerSas()
{
    StorageSharedKeyCredential storageSharedKeyCredential = new StorageSharedK
eyCredential(StorageAccountName, StorageAccountKey);

    BlobContainerClient container = new BlobContainerClient(new Uri(StorageCon
tainerAddress), new StorageSharedKeyCredential(StorageAccountName, StorageAcco
untKey));
    var policy = new BlobSasBuilder
    {
        Protocol = SasProtocol.HttpsAndHttp,
        BlobContainerName = container.Name,
        Resource = "c",
        StartsOn = DateTimeOffset.UtcNow,
        ExpiresOn = DateTimeOffset.UtcNow.AddHours(2),
        IPRange = new SasIPRange(IPAddress.None, IPAddress.None)
    };
    policy.SetPermissions(BlobSasPermissions.Read);
    var sas = policy.ToSasQueryParameters(storageSharedKeyCredential);
    BlobUriBuilder sasUri = new BlobUriBuilder(container.Uri);
    sasUri.Sas = sas;

    return $"?{sasUri.Sas}";
}
```

- Add the method AnalyzeRemoteAsync after the GetContainersSas method. This method gets the features specify from the List<VisualFeatureTypes> object created before.

```
private static async Task<List<string>> AnalyzeRemoteAsync(ComputerVisionClien
t computerVision, string imageUrl, string imageName)
{
    ImageAnalysis analysis = await computerVision.AnalyzeImageAsync(imageUrl,
features);
    return DisplayResults(analysis, imageUrl, imageName);
}
```

- Add the method DisplayResult after the AnalyzeRemoteAsync method. This method displays the result of the analyzed image.

```
private static List<string> DisplayResults(ImageAnalysis analysis, string imag
eUri, string imageName)
{
    List<string> imageAnalyzed = new List<string>();
    string feature = string.Empty;
    imageAnalyzed.Add(imageName);

    foreach (var cat in analysis.Description.Captions)
        feature += cat.Text + " ";
    imageAnalyzed.Add($"\n\tDescription: {feature}\n");
    feature = string.Empty;

    foreach (var cat in analysis.Color.DominantColors)
        feature += cat + " ";
    imageAnalyzed.Add($"\tDominant Colors: {feature} \n");
    feature = string.Empty;

    foreach (var cat in analysis.Categories)
        feature += cat.Name + " ";
    imageAnalyzed.Add($"\tCategories: {feature}\n");
    feature = string.Empty;

    foreach (var cat in analysis.Faces)
        feature += $"\t\tGender: {cat.Gender}, Age: {cat.Age} \n";
    imageAnalyzed.Add($"\tFaces: \n{feature}");
    feature = string.Empty;

    foreach (var cat in analysis.Objects)
        feature += cat.ObjectProperty + " ";
    imageAnalyzed.Add($"\tObjects: {feature}");
    feature = string.Empty;

    return imageAnalyzed;
}
```

Consume analyzed images method

- Open the program.cs from the console project.
- Remove the following lines from the Main method.

```
var result = ccs.SearchingAnalyzedQueries("Microsoft", "Sentiment");
foreach (string s in result)
{
    Console.WriteLine(s);
}
```

- Call the SearchingAnalyzedQueries method with the text to search, for example the next code searches for **Everything**.

```
ccs.ImageFeaturesAsync("*").Wait();
```

- Access to the console project path.

```
cd AzureSearchKnowledgeConsole
```

- Save the file.
- Build the project.

```
dotnet build
```

Test the application

- Back to the terminal run the console project.

```
dotnet run
```

- Explore the results.

NOTE: You can change the text to search and the output to print.