DotChat

`etap-2`

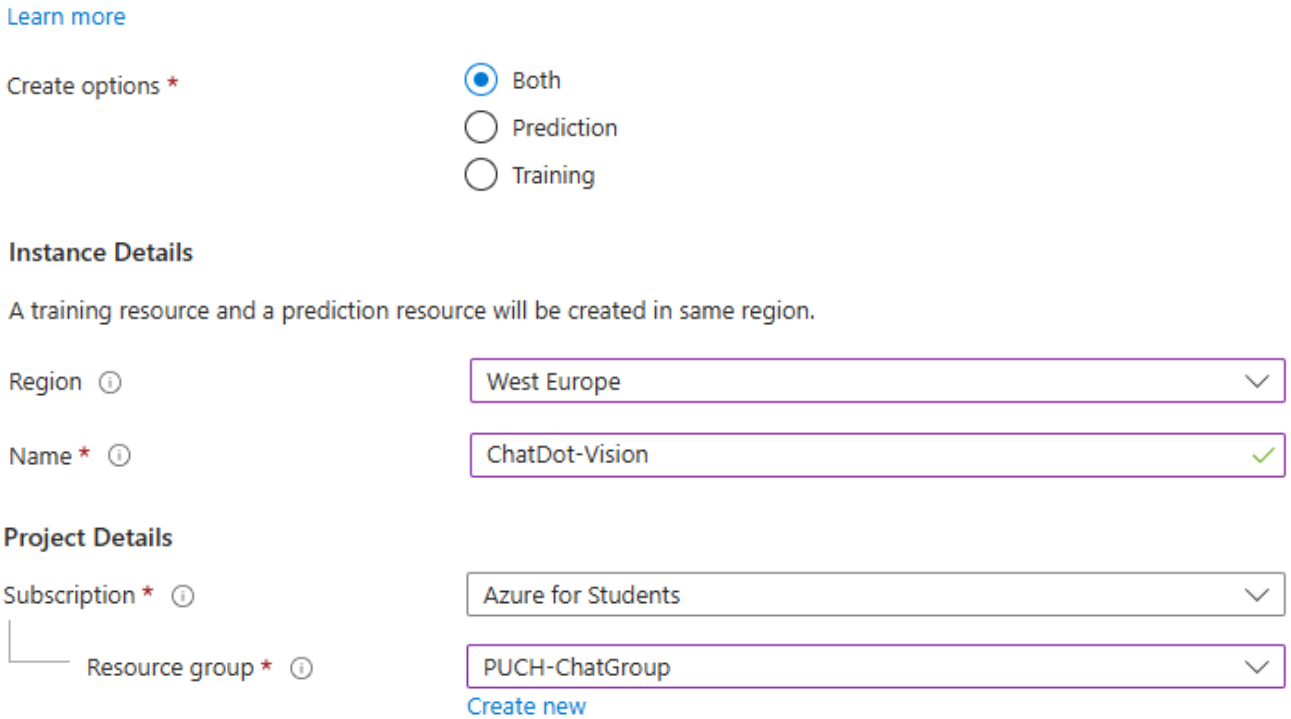
1. Custom Vision
2. temat 4: Podsumowania plików pdf

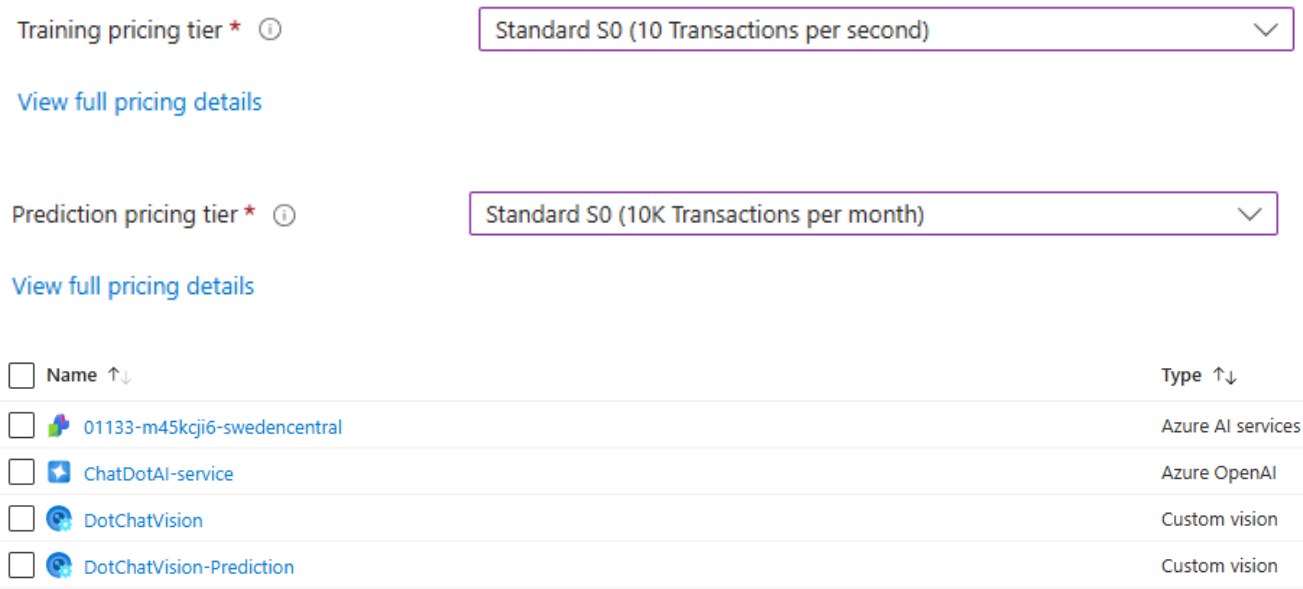
Repozytorium: github

```
:-------------------------------------------------------:
              D O T  C H A T  gpt-4
:-------------------------------------------------------:
 Here are some usefull commands:
  \user <username> - to register your username
  \system <text> - to provide context for the AI assistant
  \save <filename> - to save your chat history in a file
  \clear - to clear the chat history
  \exit - for leaving the chat
  \summary pdf <filename> - [in the making]
  \vision img <filename> - [in the making]
  ...
```

1. Custom Vision

Utworzenie zasobu Azure `Custom Vision`





Załadowanie zbioru danych

Wybrany został zbiór danych do klasyfikacji pogody na zdjęciach
kaggle

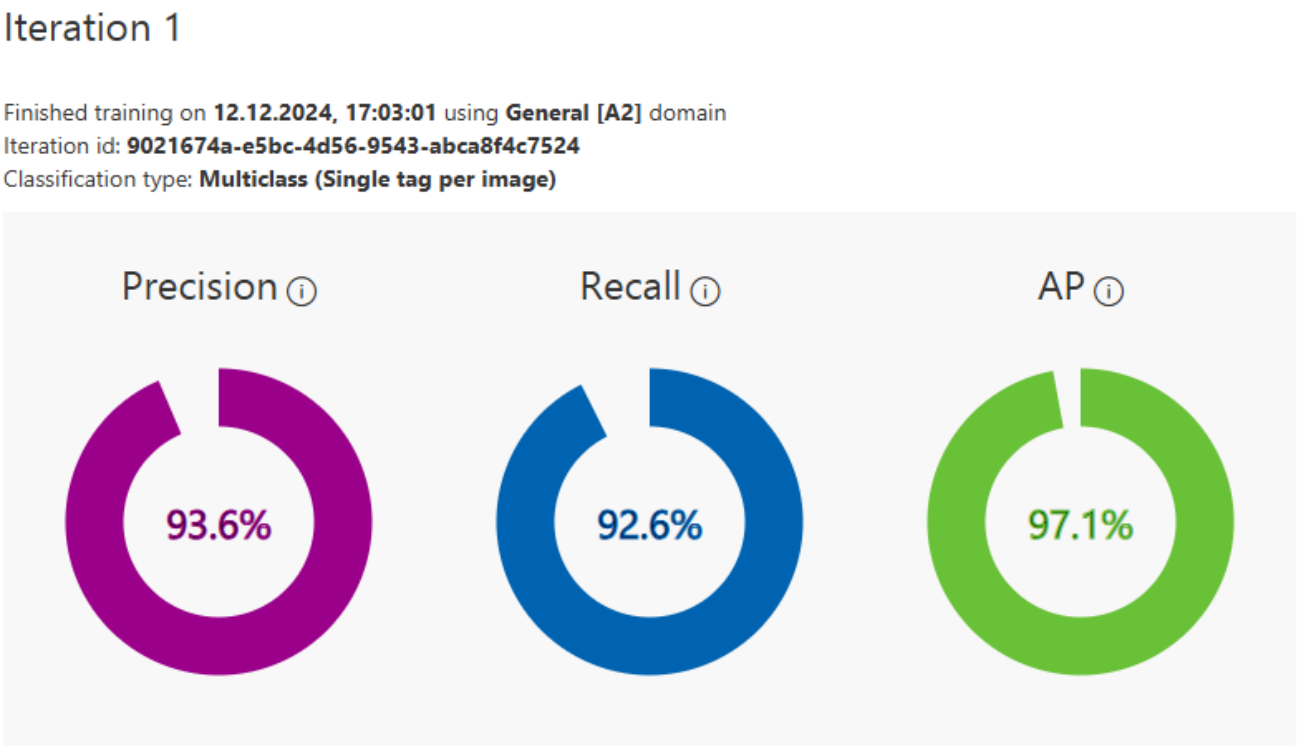
Dalej zbiór został podzielony na treningowy, testowy i walidacyjny:

```
import os
import shutil
from sklearn.model_selection import train_test_split

def split_dataset(input_dir, output_dir, train_ratio=0.7, val_ratio=0.15,
test_ratio=0.15):
    """
    Dzieli zbiór danych na podzbiory treningowy, walidacyjny i testowy.
    generated by v0
    """
    for class_name in os.listdir(input_dir):
        class_dir = os.path.join(input_dir, class_name)
        images = os.listdir(class_dir)

        train, test = train_test_split(images, test_size=1-train_ratio,
random_state=42)
        val, test = train_test_split(test, test_size=test_ratio/(test_ratio +
val_ratio), random_state=42)

        for subset, images in [('train', train), ('val', val), ('test', test)]:
            subset_dir = os.path.join(output_dir, subset, class_name)
            os.makedirs(subset_dir, exist_ok=True)
            for img in images:
                shutil.copy(os.path.join(class_dir, img), subset_dir)
```

Zbiory zostały załadowane do `customvision.ai` i otagowane

Tagi	Przykład
train_tags.png	train_image.png

Trenowanie modelu

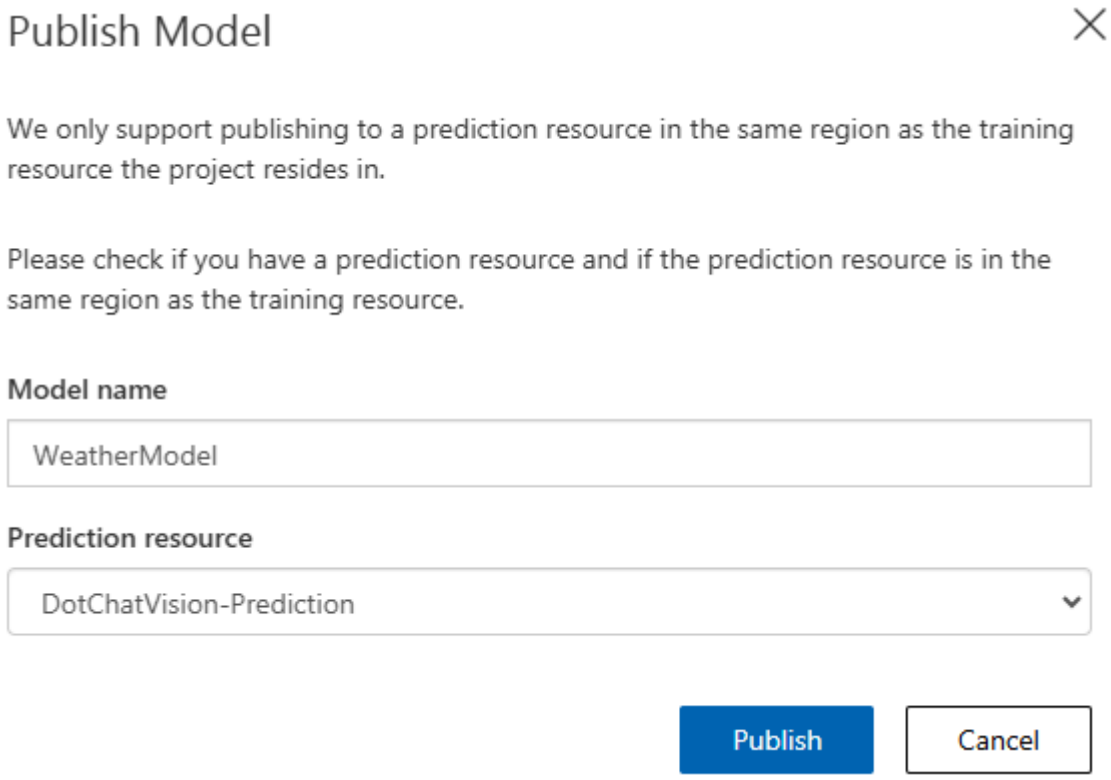


Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
rainbow	100.0%	100.0%	100.0%	158
lightning	100.0%	100.0%	100.0%	257
frost	88.5%	81.8%	92.6%	330
rain	98.6%	94.5%	98.3%	366
hail	100.0%	97.5%	99.8%	399
glaze	86.6%	84.5%	93.7%	421
dew	97.9%	97.9%	99.8%	479
sandstorm	92.7%	92.7%	96.8%	480
fogsmog	93.0%	91.5%	95.5%	585
snow	91.8%	93.3%	95.3%	601
rime	90.1%	91.3%	95.6%	801

Publikowanie modelu

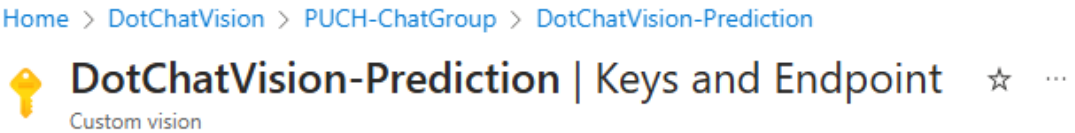
customvision.ai > Performances > Publish

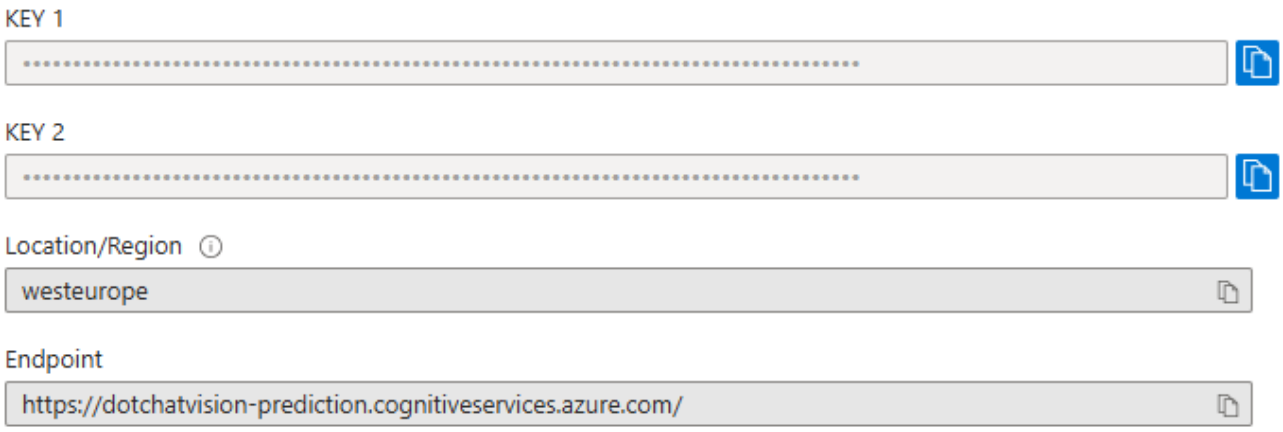


Uzyskanie odpowiednich parametrów API i umieszczenie ich w `appsettings.json`.

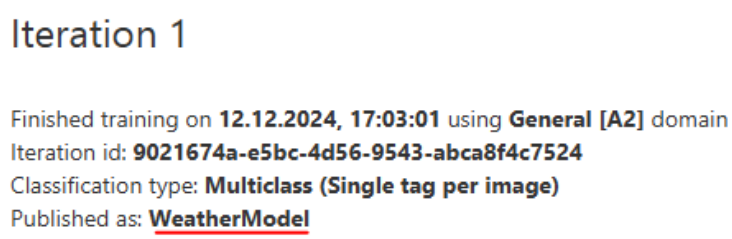
```
{
  "AzureCustomVision": {
    "Project": "WeatherClassification",
    "ProjectId": "<procject id>",
    "ResourceId": "<resource id>",
    "Training": {
      "Endpoint": "<endpoint 1>",
      "ApiKey": "<api key 1>"
    },
    "Prediction": {
      "PublishedName": "WeatherModel",
      "IterationId": "<iteration id>",
      "Endpoint": "<endpoint 2>",
      "ApiKey": "<api key 2>"
    }
  }
}
```

Prediction api key i endpoint





PublishedName



ProjectId

Wykorzystanie API Azure Custom Vision

- utworzenie klasy `AzureCVService`: github
- dodanie zależności `Microsoft.Azure.CognitiveServices.Vision.CustomVision.Training` i `Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction`

```
dotnet add package
Microsoft.Azure.CognitiveServices.Vision.CustomVision.Training --version 2.0.0
dotnet add package
Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction --version
2.0.0
```

- klasa `AzureCVService`
  - odczytuje wartości klucza api i endpointu z pliku konfiguracyjnego

```
public AzureCVService(IConfiguration configuration)
{
    _resourceId = configuration["AzureCustomVision:ResourceId"] ?? "";
    _projectId = configuration["AzureCustomVision:ProjectId"] ?? "";
    _publishedName =
configuration["AzureCustomVision:Prediction:PublishedName"] ?? "";
    _predictionKey = configuration["AzureCustomVision:Prediction:ApiKey"]
?? "";
    _predictionEndpoint =
configuration["AzureCustomVision:Prediction:Endpoint"] ?? "";
}
```

  - tworzy obiekt klienta API Azure Custom Vision

```
private CustomVisionPredictionClient getClient()
{
    return new CustomVisionPredictionClient(new
    Microsoft.Azure.CognitiveServices.
    Vision.CustomVision.Prediction.
    ApiKeyServiceClientCredentials(this._predictionKey))
    {
        Endpoint = this._predictionEndpoint
    };
}
```

  - wywołuje metodę `ClassifyImageAsync` podając id projektu, nazwę publikacji modelu, stream pliku

```
public async Task<ImagePrediction> PredictOneFile(string imageFile)
{
    if (!File.Exists(imageFile))
        throw new ArgumentException($"Path {imageFile} if not valid.");
    var client = getClient();
    using (var imageStream = new FileStream(imageFile, FileMode.Open))
    {
        Console.WriteLine("here");
        var prediction = await client.ClassifyImageAsync(
            new Guid(this._projectId),
            this._publishedName,
            imageStream
        );
        Console.WriteLine("also here");
        if (prediction == null)
```

```
            throw new Exception("Error in PredictOneFile: Prediction is
null");
        return prediction;
    }
}
```

- posiada metodę korzystjącą z `ClassifyImageUrlAsync`

```
public async Task<ImagePrediction> PredictOneUrl(string url)
{
    await IsValidImageUrlAsync(url);
    var client = getClient();
    var prediction = await client.ClassifyImageUrlAsync(
        new Guid(this._projectId),
        this._publishedName,
        new ImageUrl(url)
    );
    return prediction;
}
```

Integracja z czatem

- dodanie obiektu `AzureCVService` do klasy `Application`: github

```
  // atrybuty klasy Application
  private OpenAIService _service;
  private AzureCVService _visionService;

  // konstruktor
  public Application()
  {
      var builder = new ConfigurationBuilder()
          .SetBasePath(Directory.GetCurrentDirectory())
          .AddJsonFile("appsettings.json");
      var configuration = builder.Build();
      // Create OpenAI service
      _service = new OpenAIService(configuration);
      // Create Azure Custom Vision service
      _visionService = new AzureCVService(configuration);
  }
```

- obsłużenie komendy `\vision`: github

```
private bool ValidateVisionCommand(string userInput, out Mode mode, out
string imgSource)
{
```

```
    imgSource = string.Empty;
    mode = Mode.None;
    string pattern = @"^\\vision\s(img|url)\s\""(.*\.
(jpg|jpeg|png|gif|bmp)|(https?|ftp):\/\/([^\s\/$.?#].[^\s]*))\""$";
    Match match = Regex.Match(userInput, pattern,
RegexOptions.IgnoreCase);
    if (match.Success)
    {
        if (match.Groups[1].Value == "img")
            mode = Mode.File;
        else if (match.Groups[1].Value == "url")
            mode = Mode.Url;
        imgSource = match.Groups[2].Value;
        Console.WriteLine($"Requested image classification from
{ModeDescription.get(mode)}: {imgSource}");
        return true;
    }
    else
    {
        return false;
    }
}
```

```
// public async Task run()
//   while (true)
//     ...
//     if (words.Length > 0)
//       ...
//       switch (command)
//          ...
            case "\\vision":
              try
              {
                  if (words.Length < 3)
                  {
                      Console.WriteLine("Not enough arguments
provided.");
                      break;
                  }
                  if (ValidateVisionCommand(userInput, out Mode mode, out
string imgSource))
                  {
                      try
                      {
                          var prediction = await
_visionService.PredictOne(imgSource, mode);
                          Console.WriteLine(":------------------
Predicting weather ------------------:");
                          foreach (var label in prediction.Predictions)
                          {
```

```
                                Console.WriteLine($"- {label.TagName}:
{label.Probability * 100:F2} %");
                            }
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine($"Error: {ex.Message}");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Command was invalid.");
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Error: {ex.Message}");
                }
                break;
```

- dodanie instrukcji użytkownika

```
// public string GetGreetings()
greetings += " \\vision [options] - predicts weather from given image
with Azure Custom Vision\n";
greetings += " \\vision img \"<path to img>\"\n";
greetings += " \\vision url \"<url with img>\"\n";
greetings += " ...\n";
```

- wyniki

```
:-------------------------------------------------------:
             D O T  C H A T  gpt-4
:-------------------------------------------------------:
 Here are some usefull commands:
 \user <username> - to register your username
 \system <text> - to provide context for the AI assistant
 \save <filename> - to save your chat history in a file
 \clear - to clear the chat history
 \exit - for leaving the chat

 \vision [options] - predicts weather from given image with Azure Custom Vision
 \vision img "<path to img>"
 \vision url "<url with img>"
 ...

User: \vision url "https://images.pexels.com/photos/1003124/pexels-photo-1003124.jpeg?auto=compress&
cs=tinysrgb&w=600"
Requested image classification from url: https://images.pexels.com/photos/1003124/pexels-photo-10031
24.jpeg?auto=compress&cs=tinysrgb&w=600
:------------------ Predicting weather ------------------:
- rime: 99,03 %
- snow: 0,88 %
- glaze: 0,04 %
- frost: 0,04 %
- fogsmog: 0,01 %
- lightning: 0,00 %
- rainbow: 0,00 %
- sandstorm: 0,00 %
- dew: 0,00 %
- hail: 0,00 %
- rain: 0,00 %
User: thank you :)
GPT-4: You're welcome! :)
```

2. Podsumowania plików Pdf

`temat 4` **Tworzenie streszczenia treści dokumentu PDF**

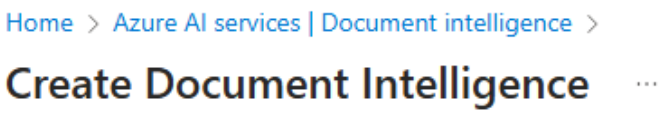
Opis zadania:

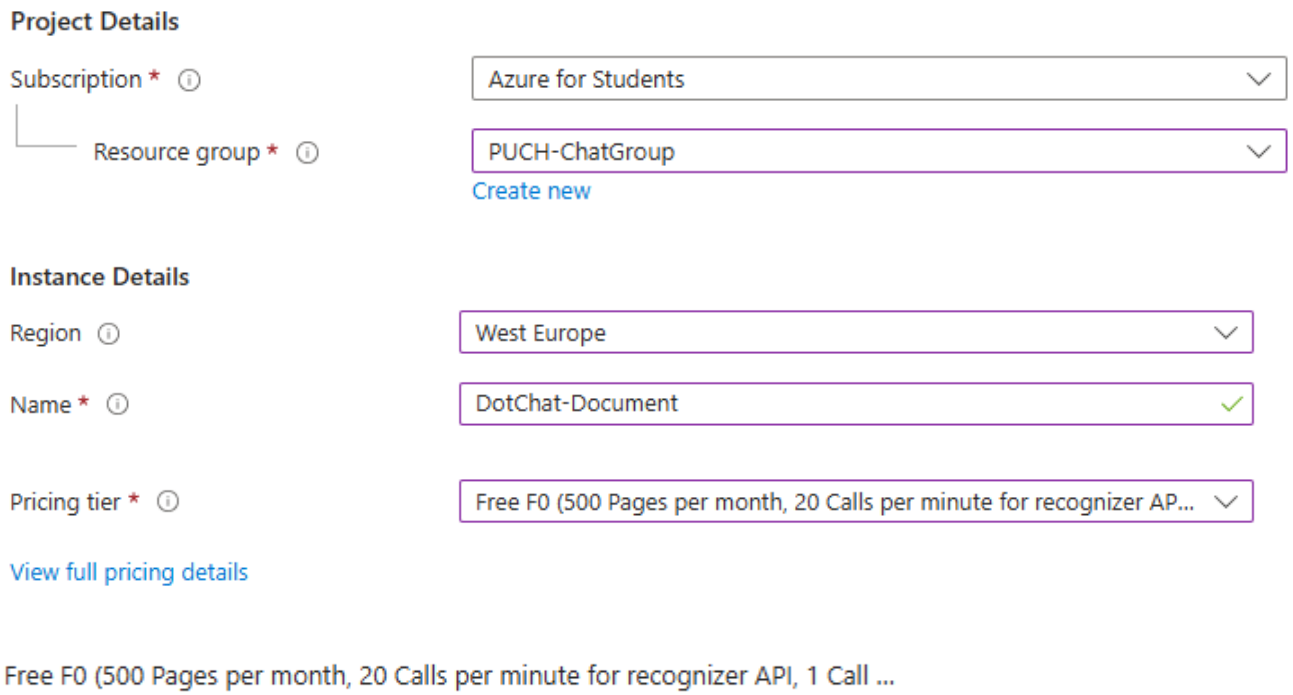
- Korzystając z OpenAI API (np. GPT-4), załaduj plik PDF, a następnie prześlij jego zawartość do modelu, aby wygenerował streszczenie.
- Wygenerowane streszczenie zapisz w pliku i wyświetl w konsoli.
- Program powinien mieć możliwość wygenerowania streszczeń wielu plików umieszczonych w folderze

Dodanie zależności do obsługi plików pdf

```
dotnet add package PdfSharpCore
dotnet add package Azure.AI.FormRecognizer
```

Dodanie zasobu Azure Document Intelligence





Integracja z czatem

- utworzenie klasy `OpenAIPdfService` i wczytanie kluczy API
- utworzenie klienta `HttpClient` do podsumowania tekstu i `DocumentAnalysisClient` do ekstrakcji treści z plików pdf
- dodanie metod realizujących funkcjonalności

```
    // konstruktor OpenAIPdfService
    _httpClient = new HttpClient();
    _httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
    _httpClient.DefaultRequestHeaders.Add("api-key", _apiKey);
    _httpClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
configuration["Azure:Subscription"]);

    _docApiKey = configuration["AzureDocumentAI:ApiKey"] ?? "";
    _docEndpoint = configuration["AzureDocumentAI:Endpoint"] ?? "";
    _docClient = new DocumentAnalysisClient(new Uri(_docEndpoint), new
AzureKeyCredential(_docApiKey));
```

```
    // określienie którą funkcjonalność wybrano (pojedyńczy plik / wiele
plików)
    public async Task Summarize(string textSource, string destination,
SummaryMode mode, bool verbose)
    {
      switch (mode)
      {
        case SummaryMode.File:
```

```
            await SummarizeOne(textSource, destination, verbose);
            break;
          case SummaryMode.Folder:
            await SummarizeMany(textSource, destination, verbose);
            break;
          default:
            Console.WriteLine("SummaryMode currently unimplemented");
            break;
        }
      }
```

```
    // wyciągnęcie tekstu z pliku z użyciem
Azure.AI.FormRecognizer.DocumentAnalysis
    private async Task<string> ExtractFromPdf(string filePath)
    {
      StringBuilder text = new StringBuilder();

      using (FileStream fs = File.OpenRead(filePath))
      {
        // Call the AnalyzeDocument method
        AnalyzeDocumentOperation operation = await
_docClient.AnalyzeDocumentAsync(WaitUntil.Completed, "prebuilt-document", fs);

        // Wait for the operation to complete
        AnalyzeResult result = await operation.WaitForCompletionAsync();

        // Output the extracted text
        Console.WriteLine("Extracting text...");
        foreach (var page in result.Pages)
        {
          Console.WriteLine($"Page {page.PageNumber}");
          foreach (var line in page.Lines)
          {
            // Console.WriteLine(line.Content);
            text.Append(line.Content.ToString());
          }
        }
      }
      return text.ToString();
    }
```

```
    // podsumowanie jednego pliku
    private async Task SummarizeOne(string sourcePath, string destPath, bool
verbose)
    {
      try
      {
        string text = await SummarizePdf(sourcePath);
```

```
        if (ValidateDestFilename(ref destPath, SaveMode.Md))
        {
          SaveTextAsMd(text, destPath);
        }
        if (verbose)
        {
          Console.WriteLine(":-----------------------------------------------
-------:");
          Console.WriteLine($" Summarizing: {sourcePath}");
          Console.WriteLine(":-----------------------------------------------
-------:");
          Console.WriteLine(text);
          Console.WriteLine($" Summary in: {destPath}");
        }
        else
        {
          Console.WriteLine($"Summary of {sourcePath} in: {destPath}");
        }

      }
      catch (Exception ex)
      {
        Console.WriteLine($"Failed to process file: {sourcePath},
{ex.Message}");
      }
    }
```

```
    // podsumowanie pliów w całym folderze
    private async Task SummarizeMany(string sourceDir, string destDir, bool
verbose)
    {
      Directory.CreateDirectory(destDir);
      string[] pdfFiles = Directory.GetFiles(sourceDir, "*.pdf");

      foreach (string filePath in pdfFiles)
      {
        try
        {
          await SummarizeOne(filePath, filePath.Replace(sourceDir, destDir),
verbose);
        }
        catch (Exception ex)
        {
          Console.WriteLine($"Failed to process {filePath}: {ex.Message}");
        }
      }
    }
```

- dodanie kluczowej funkcji `SummarizePdf`
  Podobnie jak czat w poprzednim etapie zadania, budowane jest zapytanie `OpenAIRequest.Message`.
  Dodawana jest komenda systemowa, w celu uzyskania konkretnego zachowania (podsumownaie tekstu).
  Tekst jest wychwytywany z pliku pdf za pomocą metodzy `ExtractFromPdf` i dodawany do waidomości typu
  user message. Wykonywane jest zapytanie do klienta Http. Odpoiwiedź jest opcjonalnoe drukowana na
  ekran i zapisywana we wakazanym folderze.

- dodanie obsługi nowej komendy czatu

```
        private bool ValidateSummaryCommand(string userInput, out SummaryMode
mode, out string pdfSource, out string pdfDest, out bool verbose) {
            pdfSource = string.Empty;
            pdfDest = string.Empty;
            mode = SummaryMode.None;
            verbose = false;

            // Console.WriteLine("Here 1");
            string pattern = @"^\\summarize\s+(pdf|dir)\s+(-v|\--verbose)?\s*""
([^""]+)""\s+to\s+""([^""]+)""\s*(-v|\--verbose)?$";
            // Console.WriteLine("Here 2");
            Match match = Regex.Match(userInput, pattern,
RegexOptions.IgnoreCase);
            // Console.WriteLine("Here 3");
            if (match.Success)
            {
                if (match.Groups[1].Value == "pdf")
                    mode = SummaryMode.File;
                else if (match.Groups[1].Value == "dir")
                    mode = SummaryMode.Folder;
                else
                    return false;

                pdfSource = match.Groups[3].Value;
                Console.WriteLine($"Source {pdfSource}");
                pdfDest = match.Groups[4].Value;
                Console.WriteLine($"Dest {pdfDest}");

                if (match.Groups[2].Value != null || match.Groups[5].Value !=
null)
                    verbose = true;


                Console.WriteLine($"Requested pdf summary from
{SummaryModeDescription.get(mode)}: {pdfSource} to {pdfDest}");
                return true;
            }
```

```
            else
            {
                return false;
            }
        }
```

```
    // fragment switch w metodzie run()
    case "\\summarize":
        if (words.Length < 5) {
            Console.WriteLine("Not enough arguments provided");
            break;
        }
        if (! ValidateSummaryCommand(userInput, out SummaryMode sMode, out
string fileSource, out string fileDest, out bool verbose)) {
            Console.WriteLine("Command validation failed");
            break;
        }
        try {
           await _summaryService.Summarize(fileSource, fileDest, sMode,
verbose);
        } catch (Exception ex) {
            Console.WriteLine($"Error occured: {ex.Message}");
        }
        break;
    //...
```

Wyniki

```
User: \summarize dir "C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\doc-2" to "..
\..\resources"
Source C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\doc-2
Dest ..\..\resources
Requested pdf summary from folder: C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\
doc-2 to ..\..\resources
Extracting text...
Page 1
Page 2
:------------------------------------------------------:
 Summarizing: C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\doc-2\W1 copy.pdf
:------------------------------------------------------:
The text outlines the criteria for passing a course, which includes components like lect
ures, labs, and a project, each with specific point allocations and minimum passing scor
es. It introduces the concept of cloud computing, highlighting its accessibility, major
providers, cost model, and the Azure Marketplace which offers a wide range of services.
The text emphasizes why companies invest in cloud computing, citing scalability, cost op
timization, and innovation. It provides a quick overview of the Azure Marketplace and de
tails on creating resources in Azure, including navigating the Azure Portal and deployin
g popular services. Lastly, it discusses Azure's AI services, container, and compute cap
abilities, giving a comprehensive view of the resources and benefits of using Azure for
cloud computing needs.
 Summary in: C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\resources\W1 copy.md
Extracting text...
Page 1
Page 2
Failed to process file: C:\Users\MS\Desktop\PUCH\Laboratorium 2 AI\AIDotChat\doc-2\W1.pd
f, SummarizePdf exception: Error in SummarizePdf: http status code TooManyRequests, mess
age: {"error":{"code":"429","message": "Requests to the ChatCompletions_Create Operation
 under Azure OpenAI API version 2024-08-01-preview have exceeded token rate limit of you
r current AIServices S0 pricing tier. Please retry after 46 seconds. Please contact Azur
e support service if you would like to further increase the default rate limit."}}
User:
```

Jak widać kierowanie wielu zapytań do chatu (komenda `\summarize dir`) powodowała wykorzystanie limitu dla free tier. Natomiast podsumowanie wykonane dla pierwszego dokumentu zostało wykonane i zapisane w ścieżce docelowej.