



BRIEF 2B – Conception Technique

Classification d'images pour un end-user



Présentation.....	4
Conception Technique de la solution envisagée.....	4
C1. Le site web prévu	4
Architecture.....	4
Cas d'utilisation.....	5
Graphe de dialogue.....	5
Maquettes.....	7
Technologies envisagées.....	9
C2. Monitoring.....	10
Principes retenus	10
C3. Base de données et Persistance des Données.	11
Jeu de données.....	11
Arborescence dossiers.....	12
Nomenclature fichiers	13
MCD.....	13
DD.....	13
C4. Organisation du code. Points délicats.	13
Création du projet Classifr.....	13
Custom Vision.....	13
Installer la bibliothèque Client (SDK).....	14
Créer l'application python.....	14
Authentifier le client.....	14
Créer le projet Custom Vision.....	15
Créer les étiquettes.....	15
Charger et étiqueter les images.....	15
Entraîner le projet.....	16
Publier l'itération actuelle.....	17
Tester le point de terminaison de prédiction.....	17
Ajout de nouvelles catégories	17
Récupérer l'Id du projet créer précédemment	18
Créer 2 nouvelles étiquettes	18
Charger et étiqueter les nouvelles images.....	18
Entraîner le modèle à partir de ces catégories.....	18



Publier la nouvelle itération	19
.....	19
Dépublier l'itération précédente	19
Tester le point de terminaison de prédiction.....	19
Hiérarchie des app dans Django (MVT).....	19



Présentation

- **Rappel du contexte :**

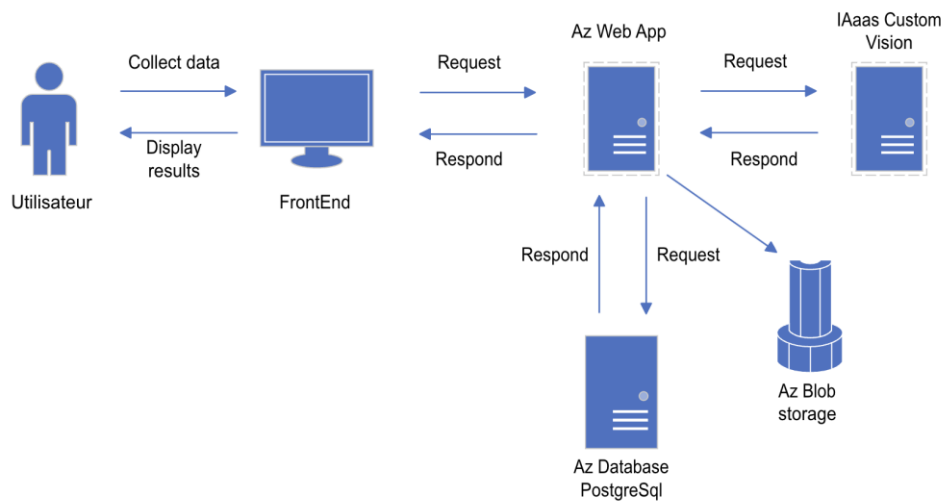
L'entreprise Classifr souhaite m'engager en tant que freelance pour une mission d'amélioration d'une application de classification d'images.

- Leur algorithme fonctionne correctement, mais ils aimeraient ajouter deux catégories d'images qui ne sont pas prises en charge actuellement.
- De plus, ils aimeraient que l'application à destination des utilisateurs permettant d'interagir avec cette IA de classification soit monitorée.

Conception Technique de la solution envisagée

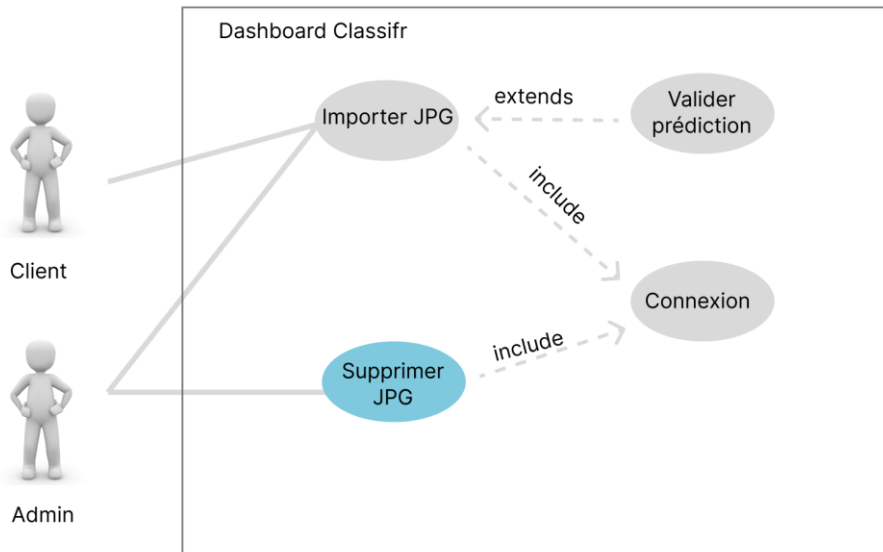
C1. Le site web prévu

Architecture





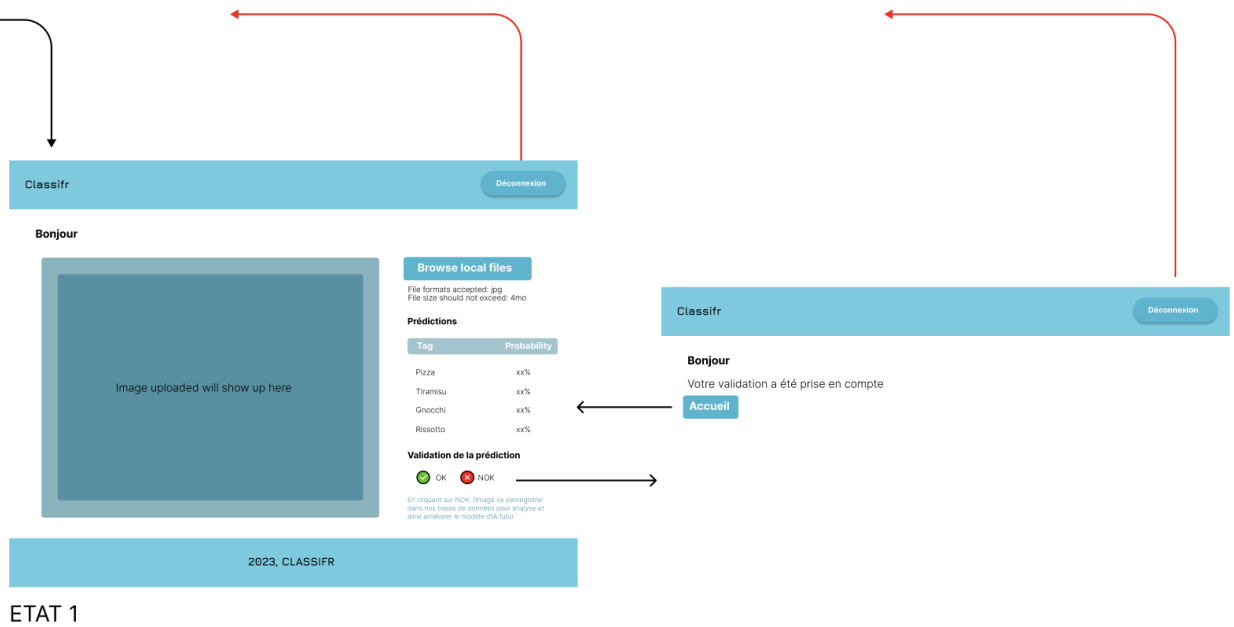
Cas d'utilisation



Graphe de dialogue

VISION CLIENT

CLIENT
ETAT 0
non connecté



ADMIN

ETAT 2



ETAT 0 :

Non connecté

ETAT 1 : une page accueil

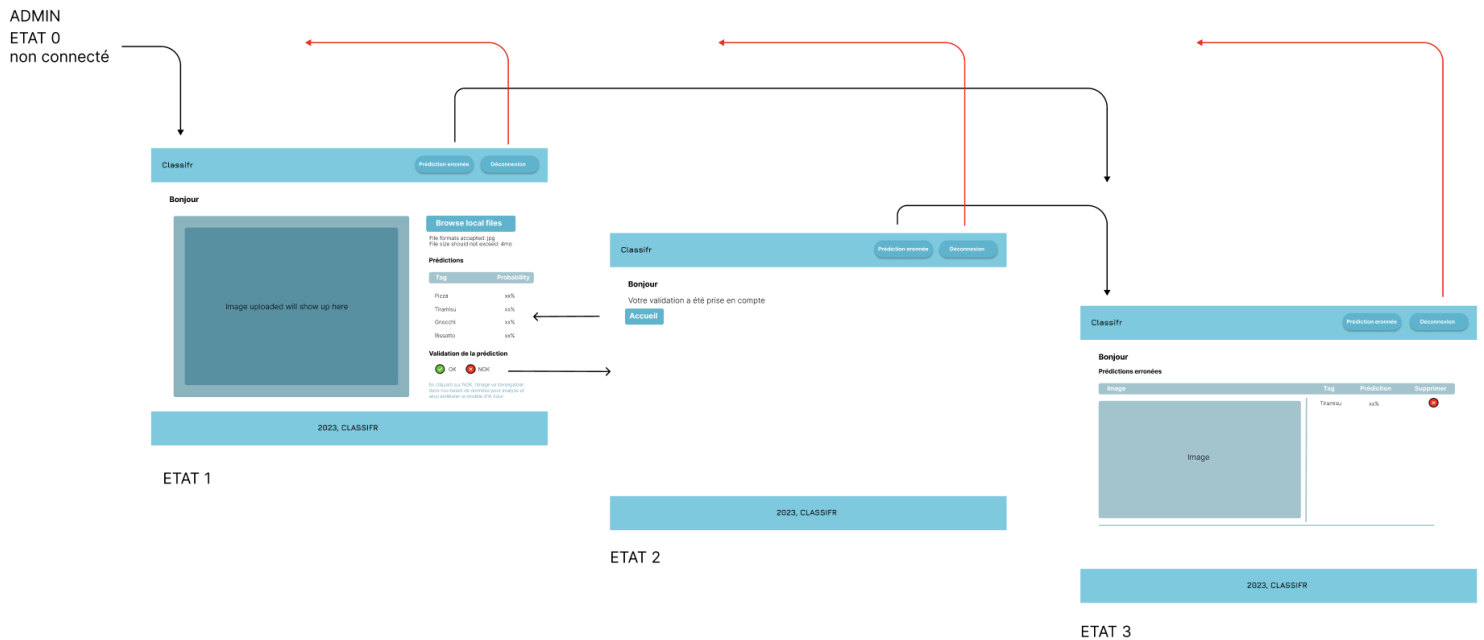
1 bouton cliquable "Browse local files" + 1 bouton cliquable "Déconnexion" + 2 boutons cliquable "OK" - "NOK"

1 zone prédiction

ETAT 2 : une page de confirmation

1 bouton cliquable "Déconnexion" + 1 bouton cliquable "Accueil"

VISION ADMIN





ETAT 0 :

Non connecté

ETAT 1 : une page accueil

1 bouton cliquable "Browse local files" + 1 bouton cliquable "Déconnexion" + 1 bouton cliquable "Prédiction erronée" + 2 boutons cliquable "OK" - "NOK"

1 zone prédiction

ETAT 2 : une page de confirmation

1 bouton cliquable "Déconnexion" + 1 bouton cliquable "Accueil" + 1 bouton cliquable "Prédiction erronée"

ETAT 3 : une page "liste des prédictions erronées"

1 bouton cliquable "Déconnexion" + 1 bouton cliquable "Accueil" + 1 bouton cliquable "Prédiction erronée"

Maquettes

The first mockup shows the login page with a header 'Classifr', a central login form with fields for 'Username' and 'Password', and a 'Connexion' button. The footer contains '2023, CLASSIFR'.

The second mockup shows the home page after login. It includes a 'Classifr' header with a 'Déconnexion' button on the right. Below the header is a 'Bonjour' greeting. The main area features a large image placeholder with the text 'Image uploaded will show up here'. To the right of the placeholder is a 'Browse local files' button, followed by file format and size instructions. Below this is a 'Prédictions' section with a table showing tags and probabilities. At the bottom of this section is a 'Validation de la prédiction' area with 'OK' and 'NOK' buttons. The footer contains '2023, CLASSIFR'.

Tag	Probability
Pizza	xx%
Tiramisu	xx%
Gnocchi	xx%
Risotto	xx%

Validation de la prédiction

OK NOK

En cliquant sur NOK, l'image va s'enregistrer dans nos bases de données pour analyse et ainsi améliorer le modèle d'IA futur



Classifr

Déconnexion

Bonjour

Votre validation a été prise en compte

Accueil

2023, CLASSIFR

Classifr

Prédiction erronée

Déconnexion

Bonjour

Image uploaded will show up here

Browse local files

File formats accepted: jpg
File size should not exceed: 4mo

Prédictions

Tag	Probability
Pizza	xx%
Tiramisu	xx%
Gnocchi	xx%
Risotto	xx%

Validation de la prédiction

OK

NOK

En cliquant sur NOK, l'image va être enregistrée dans nos bases de données pour analyse et ainsi améliorer le modèle d'IA futur

2023, CLASSIFR

Classifr

Prédiction erronée

Déconnexion

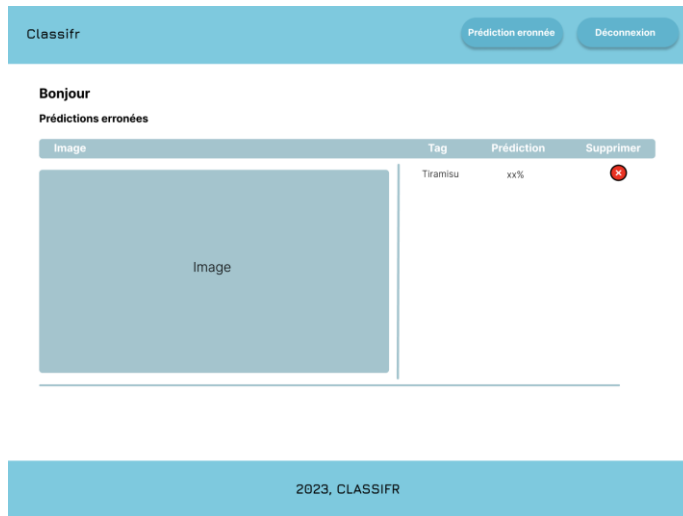
Bonjour

Votre validation a été prise en compte






Accueil

2023, CLASSIFR




8



Technologies envisagées

<p>Custom Vision</p> <p>Azure Custom Vision est un service de reconnaissance d'image qui vous permet de créer, de déployer et d'améliorer vos propres modèles d'identificateurs d'images. Un identificateur d'images applique des étiquettes à des images en fonction de leurs caractéristiques visuelles. Chaque étiquette représente une classification ou un objet. On peut utiliser Custom Vision par le biais d'un kit de développement logiciel (SDK) de bibliothèque de client, d'une API REST ou du portail web Custom Vision</p>	
<p>VisualStudio</p> <p>Est un environnement de développement intégré .NET qui peut être utilisé pour modifier, déboguer et générer du code, puis publier une application. En plus d'un éditeur de code et d'un débogueur, Visual Studio inclut des compilateurs, des outils de complétion de code, des concepteurs graphiques et des fonctionnalités de contrôle de code source pour faciliter le processus de développement de logiciels.</p>	
<p>HTML</p> <p>HTML signifie « <i>HyperText Markup Language</i> » qu'on peut traduire par « langage de balises pour l'hypertexte ». Il est utilisé afin de créer et de représenter le contenu d'une page web et sa structure. D'autres technologies sont utilisées avec HTML pour décrire la présentation d'une page (<u>CSS</u>) et/ou ses fonctionnalités interactives (<u>JavaScript</u>).</p>	
<p>CSS</p> <p>C'est un <i>langage de feuille de style</i>, c'est-à-dire qu'il permet d'appliquer des styles sur différents éléments sélectionnés dans un document HTML</p>	
<p>Bootstrap</p> <p>Est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option.</p>	



Python Est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété, Python est un langage idéal pour l'écriture de scripts et le développement rapide d'applications dans de nombreux domaines et sur la plupart des plateformes.	
Django Django est un Framework Python de haut niveau, permettant un développement rapide de sites internet, sécurisés, et maintenables.	
PostgreSQL PostgreSQL est un système de gestion de base de données relationnelle orienté objet puissant et open source qui est capable de prendre en charge en toute sécurité les charges de travail de données les plus complexes.	

C2. Monitoring

Principes retenus

Le monitoring sera basé sur l'interaction de l'utilisateur avec la bonne ou mauvaise prédiction de son image uploader.

Il pourra cliquer sur un bouton ou autre (bouton OK ou NOK), confirmant ou non la bonne prédiction (voir exemple ci-dessous qui sera visible sur sa page d'upload d'image)

Browse local files

File formats accepted: jpg
File size should not exceed: 4mo

Prédictions

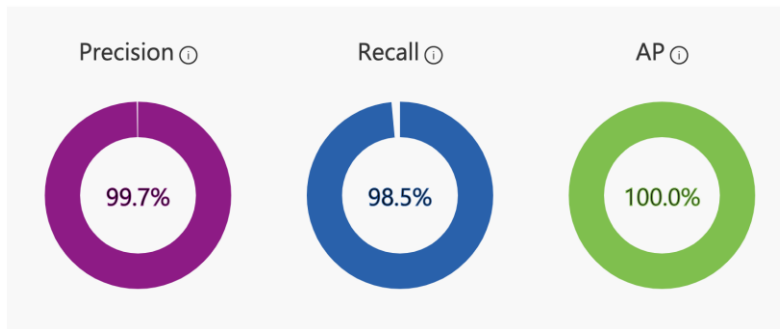
Tag	Probability
Pizza	xx%
Tiramisu	xx%
Gnocchi	xx%
Risotto	xx%

Validation de la prédiction

☒ OK ☐ NOK

En cliquant sur NOK, l'image va s'enregistrer dans nos bases de données pour analyse et ainsi améliorer le modèle d'IA futur

Un autre monitoring disponible est celui des métriques de performance du modèle de classification d'image, il est visible dans le projet sur le site Custom Vision



C3. Base de données et Persistance des Données.

Jeu de données

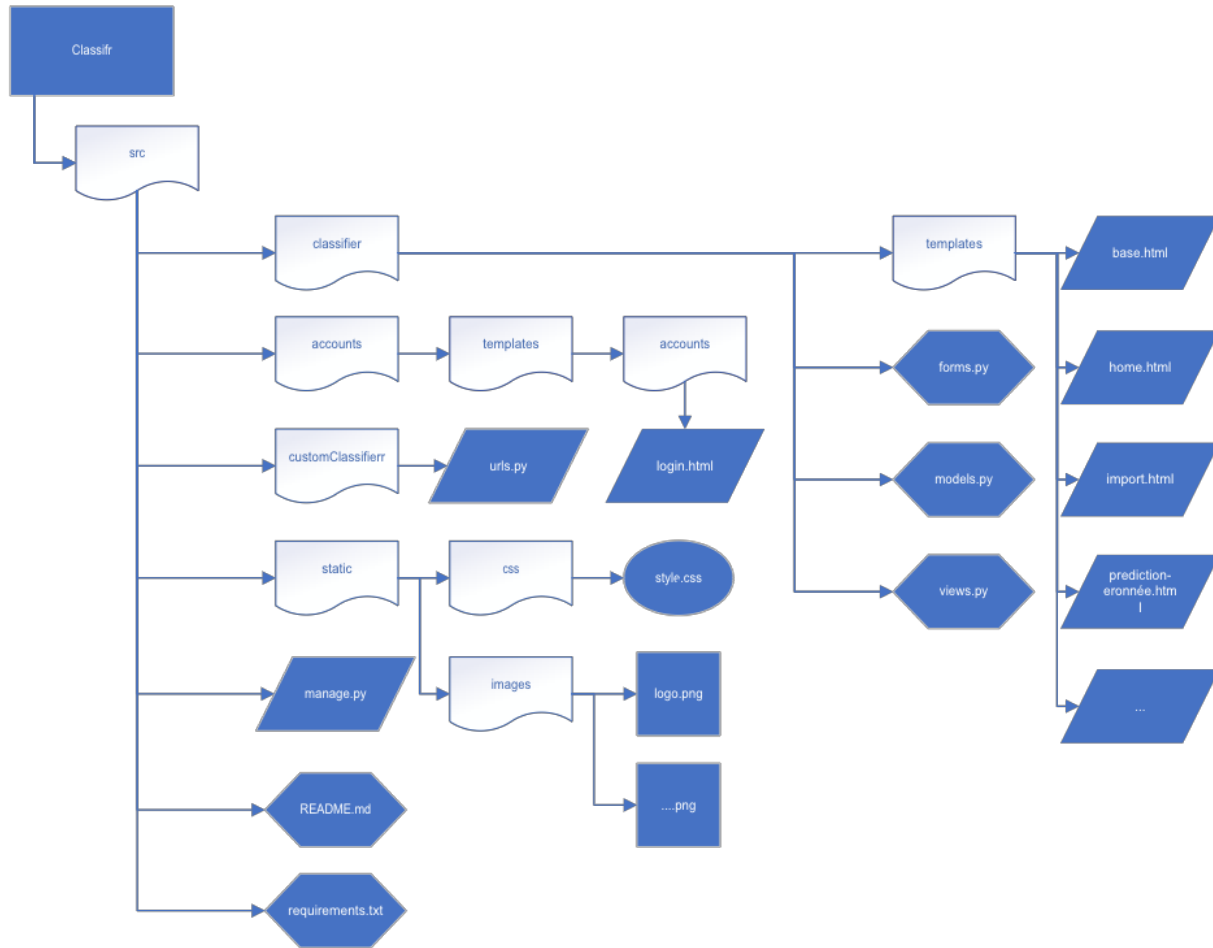
Le jeu de données provient du site Kaggle intitulé “food-101”.

Dans ce jeu de données, il y a 101 catégories d’aliments dont près de 1000 images par catégories, pour mon projet je vais utiliser 4 catégories, pizza, tiramisu, cannoli et risotto

[food-101](#)



Arborescence dossiers



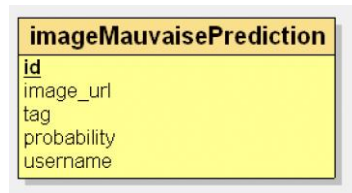


Nomenclature fichiers

Dans le dossier images, les différents fichiers seront aux formats soient png ou jpg avec comme nom xx.png ou xx.jpg (xx représentant un chiffre incrémenté)

MCD

Un MCD avec une seule entité (peut être amené à évoluer au fil de la conception du projet)



DD

	Type de données	Définition
id	INT	Identifiant unique
Image_url	VARCHAR (250) NOT NULL	Adresse de stockage de l'image dans le Blob Storage
tag	VARCHAR (50) NOT NULL	Tag identifiant l'objet détecté sur l'image
probability	VARCHAR (5) NOT NULL	% de probabilité de l'objet détecté
username	VARCHAR (50) NOT NULL	Nom de la personne qui a uploadé l'image

C4. Organisation du code. Points délicats.

Création du projet Classifr

Custom Vision

Avant toutes choses, avoir un abonnement Azure

Ensuite sur customvision.ai, créer une ressource All Cognitives Services en lien avec votre abonnement que j'ai nommé "Classifier"

Le point de terminaison (Endpoint), la clé (Key) ainsi que la Ressource Id (Resource Id) sont nécessaires afin de connecter notre application à Custom Vision



▼ Classifier

Subscription: Simplon DROM LA réunion Dev IA
Resource Group: GR16laurent
Resource Kind: All Cognitive Services

Key:

ea64d9ef43e245d6bd6d719b6420e798

Endpoint:

https://westeurope.api.cognitive.microsoft.com/

Resource Id:

/subscriptions/b5e3ab84-dbf9-4e8a-81d2-91337105a5e5/resourceGroups/GR16laurent/providers/Microsoft.CognitiveSe

Installer la bibliothèque Client (SDK)

Dans VSCode, il faut installer le Kit de développement (SDK) python de Custom Vision :

```
pip install azure-cognitiveservices-vision-customvision
```

Créer l'application python

Ensuite, importer les différentes librairies

```
from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateBatch, ImageFileCreateEntry
from msrest.authentication import ApiKeyCredentials
import os, time
```

Avec le code suivant, et la création des variables (training_endpoint,...), on va connecter notre application à notre ressource Custom Vision

```
#load the key and endpoint
training_endpoint='https://westeurope.api.cognitive.microsoft.com/'
training_key='ea64d9ef43e245d6bd6d719b6420e798'
prediction_endpoint='https://westeurope.api.cognitive.microsoft.com/'
prediction_key='ea64d9ef43e245d6bd6d719b6420e798'
prediction_resource_id='/subscriptions/b5e3ab84-dbf9-4e8a-81d2-91337105a5e5/resourceGroups/GR16laurent/providers/Microsoft.CognitiveServices/accounts/Classifier'
```

Authentifier le client

Ci-dessous, on va instancier les objets *CustomVisionTrainingClient* et *CustomVisionPredictionClient* qui va nous permettre d'interagir dans notre projet sur Custom Vision, c'est-à-dire pouvoir entraîner un modèle de classification d'image et ensuite pouvoir prédire l'objet détecté en utilisant le point de terminaison publié

```
#Authenticate the client
credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(training_endpoint, credentials)
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(prediction_endpoint, prediction_credentials)
```



Créer le projet Custom Vision

Ensuite, je vais créer un nouveau projet de Classification d'image, avec comme domaine "Food"

```
# Find the domain id
classification_domain = next(domain for domain in trainer.get_domains() if domain.type == "Classification" and domain.name == "Food")

# Create a new project
publish_iteration_name = "Iteration1"
project_name = "Italian Food classifier"
project_description = "A Custom Vision project to identify Italian food"
domain_id = classification_domain.id
classification_type = "Multiclass"
print ("Creating project...")
project = trainer.create_project(project_name, project_description, domain_id, classification_type)
```

Créer les étiquettes

Je vais créer les étiquettes de classification (catégories) qui vont représenter les catégories déjà existantes de l'entreprise Classifr

```
# Create tags for gnocchi and risotto
cannoli_tag = trainer.create_tag(project.id, "cannoli")
risotto_tag = trainer.create_tag(project.id, "risotto")
```

Charger et étiqueter les images

J'upload des images de "cannoli" et "de "risotto" dans mon projet créé sur Custom Vision afin d'entraîner mon modèle de classification

```
# Upload and tag images
images_folder = os.path.join(os.path.dirname(__file__), "images", "Train")
tags_folder_names = [ "cannoli", "risotto" ]

print("Adding images...")

for tag_num in range(0, 2):
    if tag_num == 0:
        tag = cannoli_tag
    else:
        tag = risotto_tag
    for batch_num in range(0, 16): # 16 lots de 60 images pour 960 images
        image_list = []
        for image_num in range(1, 61):
            file_name = f"{tags_folder_names[tag_num]}_{60*batch_num + image_num}.jpg"
            with open(os.path.join(images_folder, tags_folder_names[tag_num], file_name), "rb") as image_contents:
                image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[tag.id]))

        upload_result = trainer.create_images_from_files(project.id, ImageFileCreateBatch(images=image_list))

        print(f"Uploading image {file_name}")
        if not upload_result.is_batch_successful:
            print("Image batch upload failed.")
            for image in upload_result.images:
                print("Image status: ", image.status)
            exit(-1)
        print(f"{tags_folder_names[tag_num]} Uploaded")
```



Entraîner le projet

On va maintenant entraîner le modèle de classification

```
# Training
print ("Training...")
iteration = trainer.train_project(project.id)
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project.id, iteration.id)
    print ("Training status: " + iteration.status)
    print ("Waiting 10 seconds...")
    time.sleep(20)
```

Une fois le modèle entraîné, le service Custom Vision offre trois métriques de performance de ce modèle

- la *Précision* : qui permet de définir la précision d'une prédiction
- le *Recall* : définit le % de cas positif réellement prédit par rapport au nombre total de cas positif de l'ensemble des données
- *AP* (Average Precision) : mesure la qualité de la prédiction pour l'ensemble des données du test

```
# Get iteration performance information
threshold = 0.5
iter_performance_info = trainer.get_iteration_performance(project.id, iteration.id, threshold)
print("Iteration Performance:")
print(f"\tPrecision: {iter_performance_info.precision*100 :.2f}%\n"
      f"\tRecall: {iter_performance_info.recall*100 :.2f}%\n"
      f"\tAverage Precision: {iter_performance_info.average_precision*100 :.2f}%")

print("Performance per tag:")
for item in iter_performance_info.per_tag_performance:
    print(f"* {item.name}:")
    print(f"\tPrecision: {item.precision*100 :.2f}%\n"
          f"\tRecall: {item.recall*100 :.2f}%\n"
          f"\tAverage Precision: {item.average_precision*100 :.2f}%")
```

Ici le "Threshold" est tout aussi important, il représente un seuil qui détermine la décision de prédiction. C'est une valeur numérique qui définit la limite entre les prédictions positives et négatives

Il est nécessaire de procéder à une analyse approfondie et à un ajustement continu pour maximiser les performances du modèle



Publier l'itération actuelle

En publiant l'itération, on va pouvoir rendre disponible le modèle pour pouvoir l'interroger au travers du point de terminaison

```
# Publish the current iteration
print("Publishing the current iteration...")
trainer.publish_iteration(project.id, iteration.id, publish_iteration_name, prediction_resource_id)
print("Iteration published!")
```

Tester le point de terminaison de prédiction

Ici j'ai 6 images numéroté de 1 à 6, dans le dossier "Test" qui vont être prédit tour à tour

```
# Test - Make a prediction
print("Testing the prediction endpoint...")
test_images_folder_path = os.path.join(os.path.dirname(__file__), "images", "Test")

for img_num in range(1,6):
    test_image_filename = str(img_num) + ".jpg"
    with open(os.path.join(test_images_folder_path, test_image_filename), "rb") as image_contents:
        results = predictor.classify_image(project.id, publish_iteration_name, image_contents.read())

    # Display the results
    print(f"Testing image {test_image_filename}...")
    for prediction in results.predictions:
        print(f"\t{prediction.tag_name}: {prediction.probability*100 :.2f}%")
```

Ajout de nouvelles catégories

On reprend les étapes d'import de librairies, la création des variables de connexion de notre application à Custom Vision ainsi que l'authentification du client

```
from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateBatch, ImageFileCreateEntry
from msrest.authentication import ApiKeyCredentials
import os, time

#load the keys and endpoints from the configuration file
training_endpoint='https://westeurope.api.cognitive.microsoft.com/'
training_key='ea64d9ef43e245d6bd6d719b6420e798'
prediction_endpoint='https://westeurope.api.cognitive.microsoft.com/'
prediction_key='ea64d9ef43e245d6bd6d719b6420e798'
prediction_resource_id='/subscriptions/b5e3ab84-dbf9-4e8a-81d2-91337105a5e5/resourceGroups/GRI6laurent/providers/Microsoft.CognitiveServices/accounts/Classifier'

#Authenticate the client
credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(training_endpoint, credentials)
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(prediction_endpoint, prediction_credentials)
```



Récupérer l'Id du projet créer précédemment

Dans notre projet sur Custom Vision, l'Id du projet se trouve dans les paramètres du projet

Je crée une variable avec cet Id afin de pouvoir spécifiquement interagir sur ce projet

```
#Select the project id
project_id = '4dc702af-8edf-4742-a0c5-5162d58cd19d'
```

Créer 2 nouvelles étiquettes

```
# Create tags
pizza_tag = trainer.create_tag(project_id, "pizza")
tiramisu_tag = trainer.create_tag(project_id, "tiramisu")
```

Charger et étiqueter les nouvelles images

```
# Upload and tag images
images_folder = os.path.join(os.path.dirname(__file__), "images", "Train")
tags_folder_names = ["pizza", "tiramisu"]

print("Adding images...")

for tag_num in range(0, 2):
    if tag_num == 0:
        tag = pizza_tag
    else:
        tag = tiramisu_tag
    for batch_num in range(0, 16): # 16 lots de 60 images pour 960 images
        image_list = []
        for image_num in range(1, 61):
            file_name = f"{tags_folder_names[tag_num]} ({60*batch_num + image_num}).jpg"
            with open(os.path.join(images_folder, tags_folder_names[tag_num], file_name), "rb") as image_contents:
                image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[tag.id]))

        upload_result = trainer.create_images_from_files(project_id, ImageFileCreateBatch(images=image_list))

        print(f"Uploading image {file_name}")
        if not upload_result.is_batch_successful:
            print("Image batch upload failed.")
            for image in upload_result.images:
                print("Image status: ", image.status)
            exit(-1)
        print(f"{tags_folder_names[tag_num]} Uploaded")

print(f"{tags_folder_names[tag_num]} Uploaded")
```

Entraîner le modèle à partir de ces catégories

J'entraîne le modèle avec ces nouvelles catégories

```
# Training
print ("Training...")
iteration = trainer.train_project(project_id, ["pizza", "tiramisu"])
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project_id, iteration.id)
    print ("Training status: " + iteration.status)
    print ("Waiting 10 seconds...")
    time.sleep(20)
```



Je récupère les métriques de performance de ce modèle

```
# Get iteration performance information
threshold = 0.85
iter_performance_info = trainer.get_iteration_performance(project_id, iteration.id, threshold)
print("Iteration Performance:")
print(f"\tPrecision: {iter_performance_info.precision*100 :.2f}%\n"
      f"\tRecall: {iter_performance_info.recall*100 :.2f}%\n"
      f"\tAverage Precision: {iter_performance_info.average_precision*100 :.2f}%")

print("Performance per tag:")
for item in iter_performance_info.per_tag_performance:
    print(f"* {item.name}:")
    print(f"\tPrecision: {item.precision*100 :.2f}%\n"
          f"\tRecall: {item.recall*100 :.2f}%\n"
          f"\tAverage Precision: {item.average_precision*100 :.2f}%")
```

Publier la nouvelle itération

```
# Publish the current iteration
publish_iteration_name = iteration.name
print("Publishing the current iteration...")
trainer.publish_iteration(project_id, iteration.id, publish_iteration_name, prediction_resource_id)
print("Iteration published!")
```

Dépublier l'itération précédente

```
# Get the list of published iterations
published_iterations = trainer.get_iterations(project_id)

# Find the previous iteration
previous_iteration = None
for iteration in published_iterations:
    if iteration.name == publish_iteration_name:
        break
    previous_iteration = iteration

# Unpublish the previous iteration
if previous_iteration:
    trainer.unpublish_iteration(project_id, previous_iteration.id)
    print(f"Iteration {previous_iteration.name} unpublished.")
```

Tester le point de terminaison de prédiction

J'ai 6 images numérotées de 7 à 12 dans le dossier "Test" qui vont être prédit tour à tour

```
# Test - Make a prediction
print("Testing the prediction endpoint...")
test_images_folder_path = os.path.join(os.path.dirname(__file__), "images", "Test")
for img_num in range(7,12):
    test_image_filename = str(img_num) + ".jpg"
    with open(os.path.join(test_images_folder_path, test_image_filename), "rb") as image_contents:
        results = predictor.classify_image(project_id, image_contents.read())

    # Display the results
    print(f"Testing image {test_image_filename}...")
    for prediction in results.predictions:
        print(f"\t{prediction.tag_name}: {prediction.probability*100 :.2f}%")
```

Hiérarchie des app dans Django (MVT)

Une application sera dédiée à la gestion des comptes utilisateurs et une autre pour le projet Custom Vision Classification d'image.



Pour l'Utilisateur, tout va se jouer au niveau de la *View* lors de l'upload d'image par l'utilisateur.

L'upload déclenchera la fonction dans le *View* qui lancera le processus de prédiction (connexion au projet Custom Vision) et le résultat sera restitué au travers du *Template* (*page accueil, zone prédiction*).

La conservation de l'image, lorsque l'utilisateur identifiera que la prédiction n'est pas bonne sera enregistrer au travers du *Model* grâce à la création d'une *class imageMauvaisePrediction* (voir le point [MCD](#)). Image qui s'enregistrera dans Azure Blob Storage.

Pour l'*Admin*, la gestion des images qui n'ont pas été prédit comme il se doit, il pourra les voir grâce au *Template* de la page "Prédiction erronée" qui répertoriera toutes ces images. Il pourra les supprimer depuis cette page, une fois analyse de celle-ci effectué.

Pour tous ce qui est de la gestion de l'application, ajout de nouvelles images, de nouvelles catégories, d'entraînement du modèle, de publication d'itération etc.,... l'*Admin* (ici le développeur), les gèrera en Backend soit au travers de la partie IDE ou sur le site de Custom Vision.