FACULTÉ DE GESTION, ÉCONOMIE & SCIENCES

isea edn

MASTER : *M2 CYBER*

Durée : 4 heures

Enseignant : Guillaume Chervet

Nb page(s) : 11

- Le plagiat reste proscrit, une vérification pourra être effectuée.

TYPE DE BESOINS :			
☒ INTERCALAIRE SIMPLE	☐ ~~INTERCALAIRE DOUBLE~~	☐ CALCULATRICE	☐ SANS CALCULATRICE
☐ ~~COPIES DS~~	☐ SALLE INFO	☒ ORDI PERSO	☐ CLE USB
☒ AVEC DOCUMENTS (Internet)	☐ SANS DOCUMENTS	☒ SURVEILLANTS	☒ BROUILLONS
☐ PLAN COMPTABLE	☐ CODE (A PRECISER)	☐ DICTIONNAIRE (A PRECISER)	☐ AUTRES (A PRECISER)

Merci d'apporter vos précisions au niveau des consignes particulières svp.

CONSIGNES PARTICULIERES : Ce devoir nécessite votre ordinateur personnel ainsi qu'une connexion à internet et un accès à une prise électrique. Tous les supports sont autorisés. Chaque étudiant a un contenu d'examen et un rendu d'examen qui est différent.....
.....

.....

Examen Janvier 2025 – Session 1

Microsoft Technologies

N° Etudiant : .....

Microsoft Technologies

Tous les supports sont autorisés (votre ordinateur, internet). Il n'y a aucun piège dans ce partiel, l'idée est de parcourir sous une autre forme ce qui a été réalisé en cours.

1. Compétences attendues

- Respecter organisation et nommage d'un projet C#
- Créer une librairie indépendante
- Gérer l'asynchronisme/parallélisme pour un calcul nécessitant du CPU
- Consommer une librairie tierce disponible depuis Nuget
- Tester unitairement une librairie/débugger
- Savoir créer, manipuler un projet solution .sln
- Savoir créer, manipuler un projet C# .csproj
- Gérer les références entre les projets de la même solution
- Savoir compiler un projet avec le dotnet cli
- Mettre en place un pipeline d'intégration continue avec Github Action
- Mettre en place une minimal Web Api
- Être Autonome

2. Objectifs

Vous allez créer une librairie qui va permettre de recevoir en entrée une ou plusieurs images dans lesquelles il faut rechercher des objets. En sortie, vous retrouverez un tableau des coordonnées des objets détectés pour chaque image ainsi que l'image avec les contours des détections dessinés.

La technique utilisée est une Intelligence Artificielle très connu de détection d'object qui s'apelle Yolo. Plus extactement, Yolo Tiny v2, c'est une version très légère et rapide. L'IA a déjà été entrainé sur une série d'objet : "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"

Exemple d'image d'entrée

Exemple d'image de sortie

Pour cela vous allez utiliser la librairie ML.net de Microsoft : https://dotnet.microsoft.com/en-us/apps/ai/ml-dotnet

Votre librairie permettra de détecter des objets dans plusieurs images à la fois sans bloquer le thread courant.

Les exécutions seront réalisées en parallèle et par conséquent la charge sera mieux répartie sur tous vos processeurs ce qui va accélérer le temps de traitement.

3. A Rendre avant la fin du temps imparti

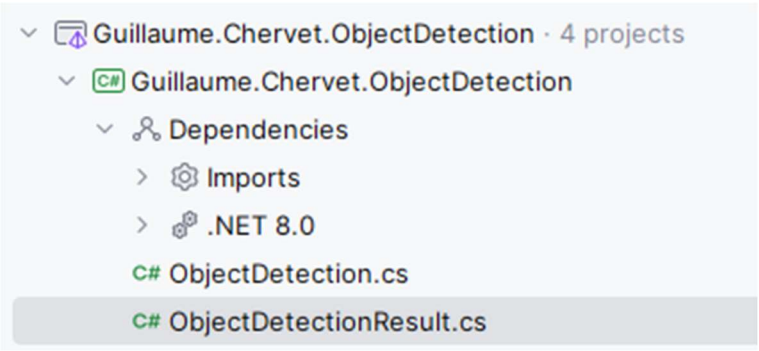
Votre projet « zippé » avec le format de nommage « numéro-la-catho.ObjectDetection.zip » dans la section « Icampus » de ce cours qui a été prévu pour déposer cet examen. Attention, il vous faut supprimer les répertoires "/bin" dans chaque projet sinon le « zip » sera trop gros. Il doit faire moins de 100 Mo.

4. Prérequis

Télécharger, installer et utiliser le sdk version SDK 8.0.404 :

https://dotnet.microsoft.com/en-us/download/dotnet/8.0

5. Initiation de la solution (.sln) et création du projet librairie (3 points)



Vous allez créer une librairie qui réalise le ObjectDetection dont le « TargetFramework » est « net8.0 ». Son nom et namespace doivent respecter la convention `VotrePrénom.VotreNom.ObjectDetection` (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom)

Fichier « ObjectDetectionResult.cs » Type de retour de votre librairie

```
namespace VotrePrénom.VotreNom.ObjectDetection;

public record ObjectDetectionResult
{
    public byte[] ImageData { get; set; }
    public IList<BoundingBox> Box { get; set; }
}
```

Fichier « ObjectDetection.cs » classe qui expose la méthode publique de votre librairie

```
namespace VotrePrénom.VotreNom.ObjectDetection;

public class ObjectDetection
{

    public async Task<IList<ObjectDetectionResult>>
DetectObjectInScenesAsync(IList<byte[]> imagesSceneData)
    {
        await Task.Delay(1000);
        // TODO implement your code here
        throw new NotImplementedException();
    }

}
```

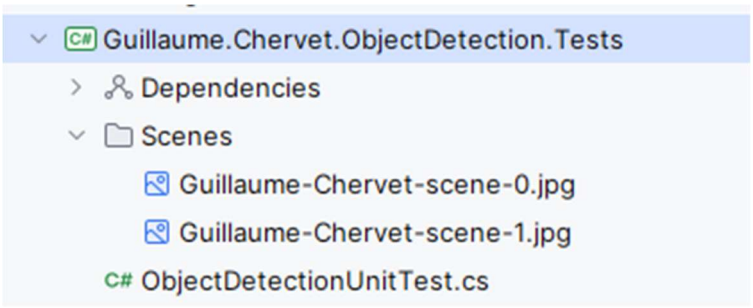
A Réaliser :

1. Créer un projet Solution `VotrePrénom.VotreNom.ObjectDetection.sln` à la racine de votre projet (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom)
2. Créer le projet « .csproj » pour initier votre librairie de détection d'objet en respectant les consignes de nommage ci-dessus. A vous de bien respecter la hiérarchie des répertoires des projets .NET sur le disque dur.
3. Associer votre nouveau projet à la solution « .sln »
4. Créer les fichiers ObjectDetectionResult.cs et ObjectDetection.cs en positionnant le code dans l'énoncé ci-dessous. Comme vous pouvez le constater, pour le moment votre librairie retournera une Exception « NotImplementedException » via la méthode « DetectObjectInScenes ».

6. Création d'un projet de test unitaire (4 points)

Attention : Lisez et prenez le temps de lire toute la section avant de commencer à coder.

Le test unitaire ci-dessous (cf. code) consommera la librairie que vous venez de créer. Il utilise deux images « scène » qui seront associées au projet de test unitaire dans le sous répertoire « Scenes », exemple :



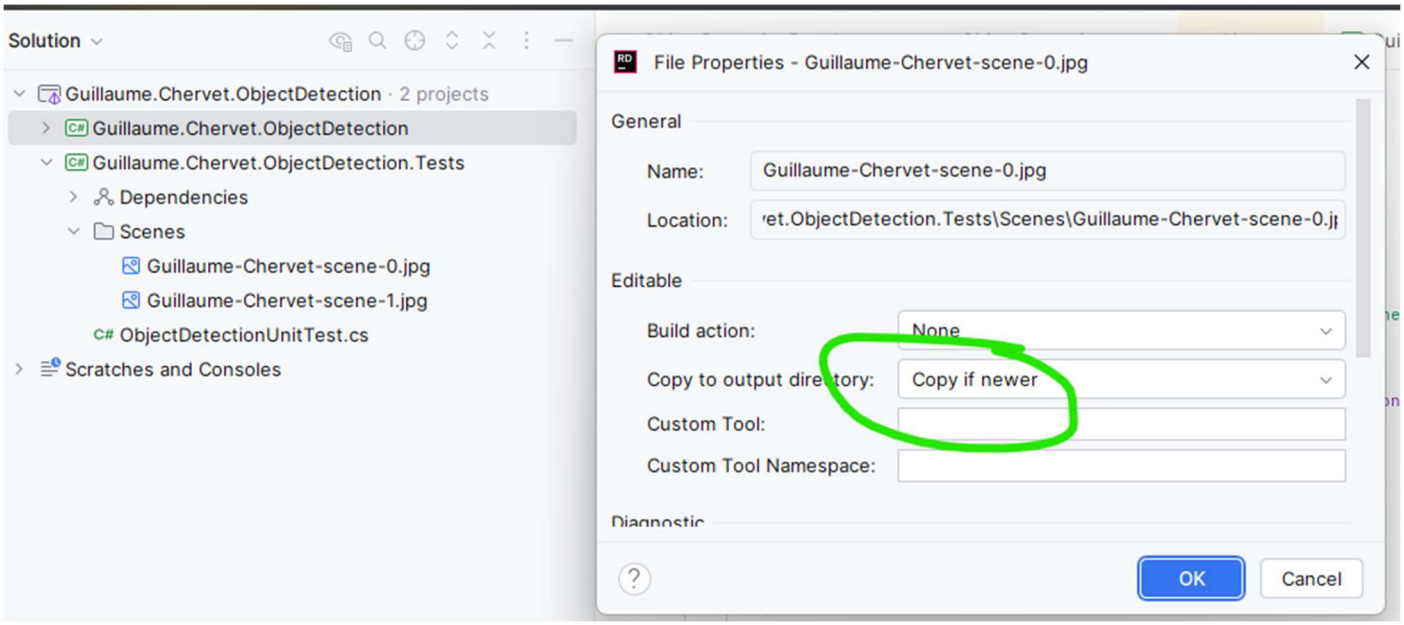
Vous avez deux images à télécharger depuis GitHub. Trouver l'image qui contient déjà votre nom et prénom :

https://github.com/guillaume-chervet/ms-exam-january-2025/tree/main/images

- VotrePrénom-VotreNom-scene-0.jpg => à mettre dans le répertoire « Scenes »
- VotrePrénom-VotreNom-scene-1.jpg => à mettre dans le répertoire « Scenes »

(bien sûr « VotrePrénom » correspond à votre prénom et « VotreNom » à votre nom)

Pour que ce test fonctionne, il faut bien configurer les images afin qu'elles soient automatiquement copiées dans le répertoire de sortie de « build » des librairies C#. Exemple :



Clic droit sur l'image puis sélectionner « Properties ». Puis dans le champ « Copy to output directory » sélectionner « Copy if newer ».

Fichier ObjectDetectionUnitTest.cs

```
using System.Collections.Generic;
using System.IO;
using System.Reflection;
using System.Text.Json;
using System.Threading.Tasks;
using Xunit;


namespace VotrePrénom.VotreNom.ObjectDetection.Tests;

public class ObjectDetectionUnitTest
{
    [Fact]
    public async Task ObjectShouldBeDetectedCorrectly()
    {
        var executingPath = GetExecutingPath();
        var imageScenesData = new List<byte[]>();
        foreach (var imagePath in Directory.EnumerateFiles(Path.Combine(executingPath,
"Scenes")))
        {
            var imageBytes = await File.ReadAllBytesAsync(imagePath);
            imageScenesData.Add(imageBytes);
        }


        var detectObjectInScenesResults = await new
ObjectDetection().DetectObjectInScenesAsync(imageScenesData);


Assert.Equal["{\"Dimensions\":{\"X\":0,\"Y\":0,\"Height\":2,\"Width\":2},\"Label\":\"Ca
r\",\"Confidence\":0.5}]",JsonSerializer.Serialize(detectObjectInScenesResults[0].Box))
;

Assert.Equal["{\"Dimensions\":{\"X\":0,\"Y\":0,\"Height\":2,\"Width\":2},\"Label\":\"Ca
r\",\"Confidence\":0.5}]",JsonSerializer.Serialize(detectObjectInScenesResults[1].Box))
;
    }

    private static string GetExecutingPath()
    {
        var executingAssemblyPath = Assembly.GetExecutingAssembly().Location;
        var executingPath = Path.GetDirectoryName(executingAssemblyPath);
        return executingPath;
    }
}
```

A Réaliser :

5. Créer un projet de test unitaire VotrePrénom.VotreNom.ObjectDetection.Tests qui utilise le framework de test « xunit » et dont le « TargetFramework » est « net8.0 ». A vous de bien respecter la hiérarchie des répertoires des projets .NET sur le disque dur.
6. Associer votre projet de test au projet solution.
7. Copier le test unitaire ci-dessus toujours en remplaçant « VotrePrénom » par votre prénom et « VotreNom » par votre nom.
8. Depuis votre projet de test unitaire, vous devez référencer votre librairie de ObjectDetection.
9. Télécharger, positionner, configurer les deux images afin qu'elles permettent au test de bien récupérer les images lors de l'exécution.

En l'état le test unitaire compile correctement, mais son exécution doit échouer (**rouge**) car votre librairie lance une Exception.

7. Implémenter votre librairie de ObjectDetection (4 points)

Attention : Lisez et prenez le temps de lire toute la section avant de commencer à coder.

Votre librairie devra référencer les packages nugets ci-dessous :

- **Microsoft.ML version 4.0.0**
- **Microsoft.ML.ImageAnalytics version 4.0.0**
- **Microsoft.ML.OnnxRuntime version 1.20.1**
- **Microsoft.ML.OnnxTransformer version 4.0.0**
- **System.Drawing.Common version 9.0.0**

System.Drawing.Common nécessite libgdiplus. Vous pouvez l'installer avec les commandes suivantes :

Sous macOS :

- brew install mono-libgdiplus

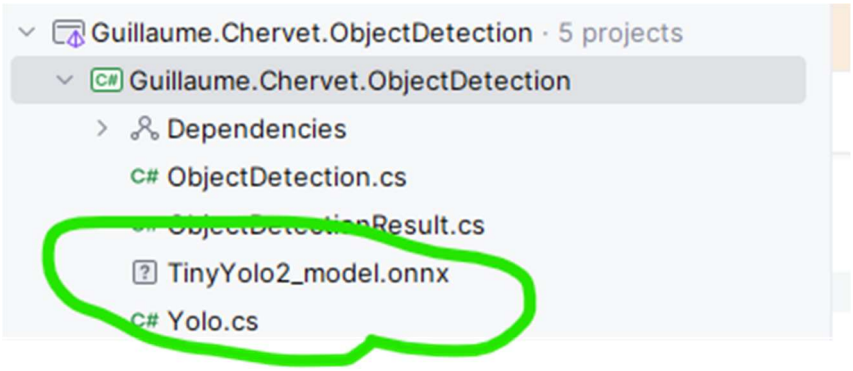
Sous Linux (exemple pour Ubuntu) :

- sudo apt-get install -y libgdiplus

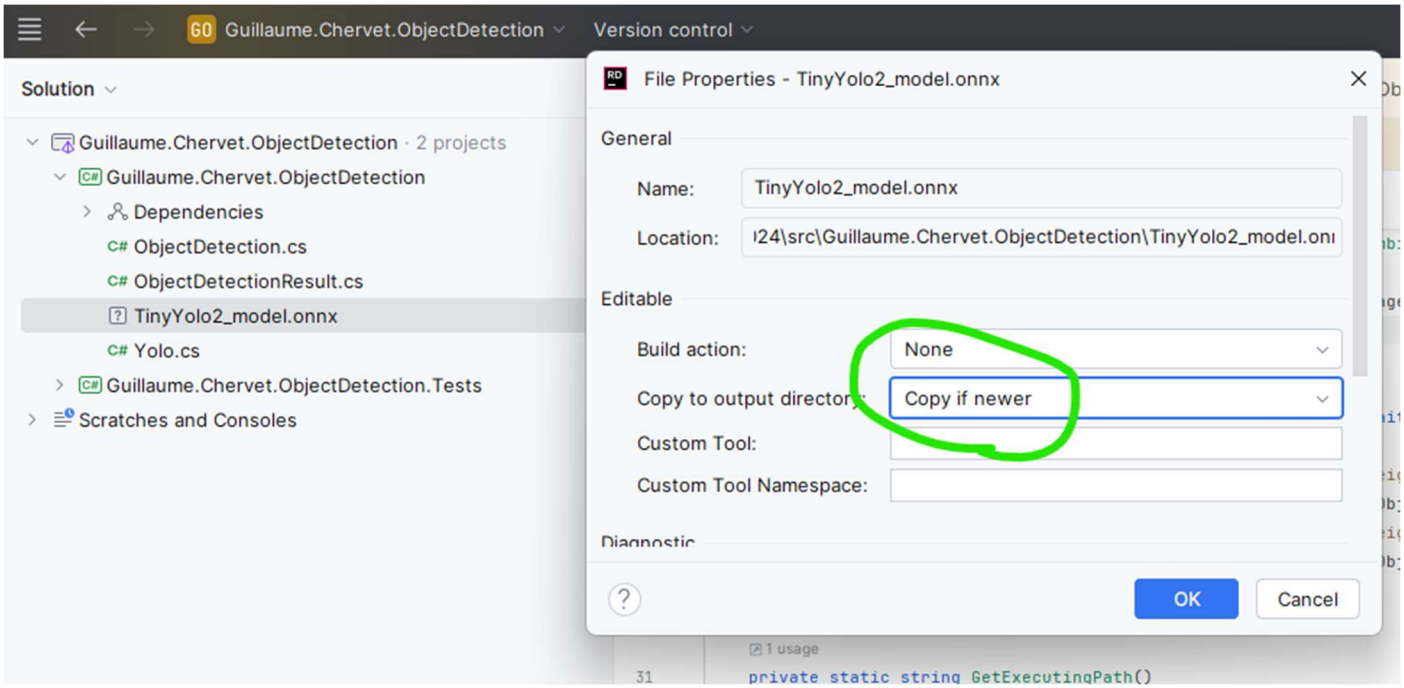
Vous devez télécharger les deux fichiers ci-dessous depuis l'adresse GitHub et les copier dans votre librairie.

https://github.com/guillaume-chervet/ms-exam-january-2025/tree/main/library

- TinyYolo2_model.onnx (c'est une IA au format onnx)
- Yolo.cs (c'est un wrapper qui va vous simplifier l'utilisation de l'IA, il ne permet déctecter qu'une seule image à la fois. Il ne faudra pas modifier le code de ce fichier)



Configurer l'IA pour qu'elle soit toujours copiée dans le répertoire d'Exécution :



Exemple fonctionnel et synchrone d'utilisation de de l'IA Yolo en C# :

```
// Import
using ObjectDetection;


// Code qui utilise Yolo.cs
var image = new byte[]; // une image au format tableau de byte
var tinyYolo = new Yolo();
var result = tinyYolo.Detect(image);
```

Votre librairie devra réaliser tous les traitements de détection en parallèle.

A Réaliser :

1. Dans votre librairie de détection d'objet, importer les librairies nécessaires depuis nuget. Si vous bloquez à faire fonctionner la librairie, passez à la suite en retournant des faux résultats en dur afin de pouvoir continuer l'examen.
2. Copier depuis Github l'IA et le fichier Yolo.cs qui va vous aider à réaliser votre librairie.
3. Réaliser l'implémentation complète du code dans la méthode « DetectObjectInScenesAsync » afin que plusieurs images de type « scènes » puissent être traitées en parallèle via l'utilisation de « Task.Run ». Votre méthode utilisera la méthode « tinyYolo.Detect » présent dans le fichier « Yolo.cs » .
4. Une fois votre librairie fonctionnelle, dans le test unitaire, remplacer les résultats attendus de type "[{\"Dimensions\":{\"X\":0,\"Y\":0,\"Height\":2,\"Width\":2},\"Label\":\"Car\",\"Confidence\":0.5}]" par vos propres valeurs afin que votre test unitaire soit passant (au vert), le debugger peut vous aider.

Exemple de code « mocké » qui retourne un faux résultat :

```
public async Task<IList<ObjectDetectionResult>> DetectObjectInScenesAsync(IList<byte[]>
imagesSceneData)
{
    IList<ObjectDetectionResult> results = new List<ObjectDetectionResult>();
    results.Add(new ObjectDetectionResult
        { ImageData = [0],
            Box = new List<BoundingBox>()
            {
                new() {Confidence=0.5f,Label="Car", Dimensions = new
BoundingBoxDimensions() { Height = 2, Width = 2, Y = 0, X= 0} },
            }} );
    results.Add(new ObjectDetectionResult
    { ImageData = [0],
        Box = new List<BoundingBox>()
        {
            new() {Confidence=0.9f,Label="Flower", Dimensions = new
BoundingBoxDimensions() { Height = 1, Width = 1, Y = 1, X= 1} },
        }} );
    await Task.Delay(1000);
    return results;
}
```

8. Création d'un projet console qui consomme votre librairie (3 points)

A Réaliser :

1. Créer un projet « Console » qui consomme votre librairie.
   Son nom et namespace doivent respecter la convention VotrePrénom.VotreNom.ObjectDetection.Console
   (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom). A vous de bien respecter la hiérarchie des répertoires des projets .NET. A vous de bien associer ce projet à la solution .sln.

   Le premier argument injecté à l'application doit être le chemin complet du répertoire qui contient les images de « scène » à la racine de celui-ci.

   Exemple d'utilisation de votre projet console :

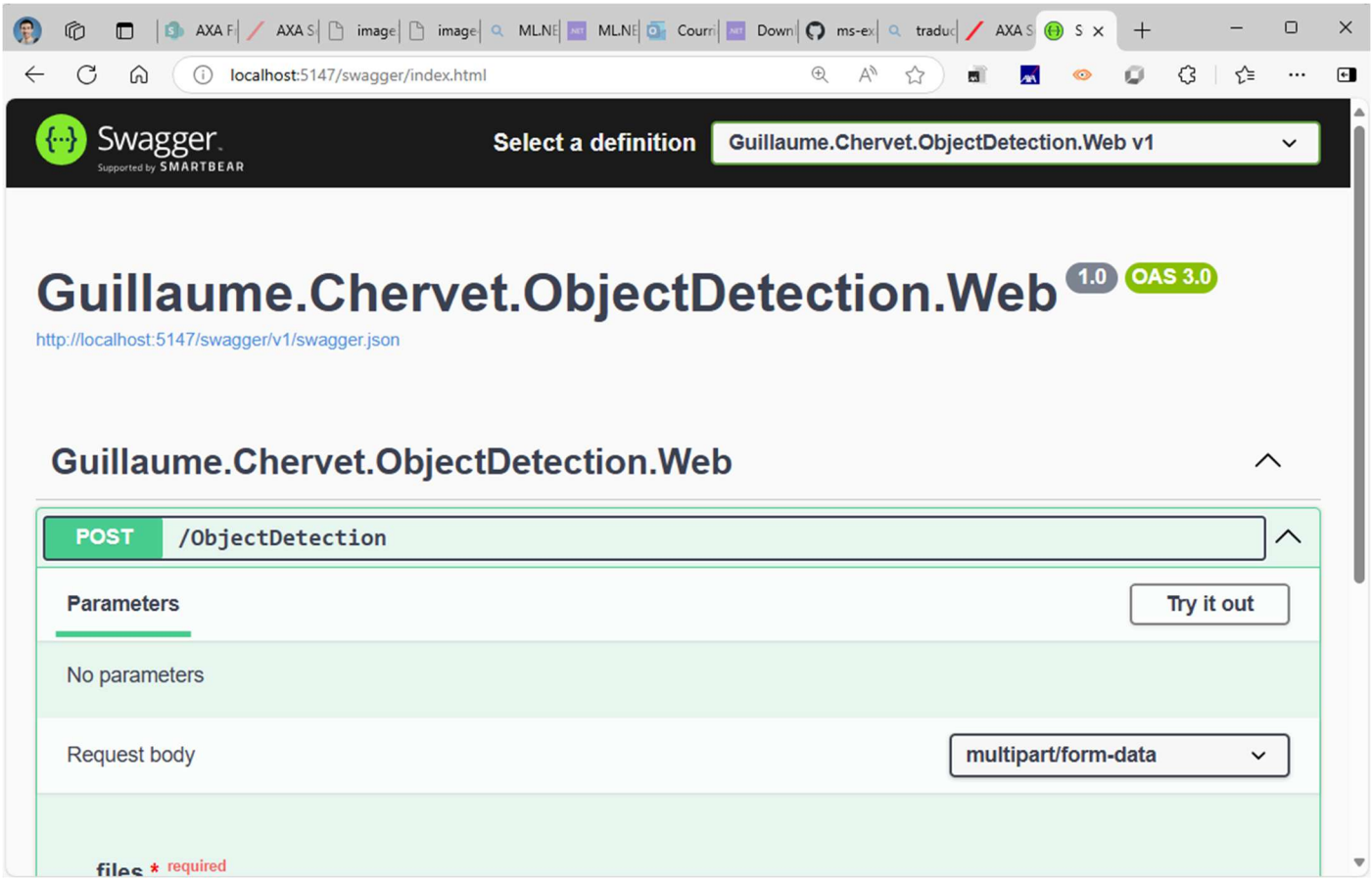
   - *dotnet run "C:\Scenes"*

Le projet console doit imprimer dans la console toutes les détections au format JSON pour toutes les images présentes dans le répertoire qui contient les « Scènes ».

```
foreach (var objectDetectionResult in detectObjectInScenesResults)
{
    System.Console.WriteLine($"Box:
{JsonSerializer.Serialize(objectDetectionResult.Box)}");
}
```

9. Exposition via une Api Web (3 points)

L'objectif de cette section est d'exposer votre librairie via une WebApi et de la rendre testable manuellement dans un browser via Swagger.



A réaliser :

1. Créer un projet « WebApi » de type minimal API qui consomme votre librairie.
   Son nom et namespace doivent respecter la convention VotrePrénom.VotreNom.ObjectDetection.WebApi (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom). A vous de bien respecter la hiérarchie des répertoires des projets .NET. A vous de bien associer ce projet au projet solution.

2. Ajouter Swagger et l'interface Swagger UI.

3. Ajouter un une route HTTP POST « ObjectDetection » qui contient le code ci-dessous puis compléter l'implémentation de la méthode afin qu'elle retourne l'image qui affiche la zone détectée et retournée par votre librairie. Ainsi dans le « Browser » vous pourrez voir visuellement le résultat de la détection.

```
app.MapPost("/ObjectDetection", async ([FromForm] IFormFileCollection files) =>
{
    if (files.Count < 1)
        return Results.BadRequest();

    using var sceneSourceStream = files[0].OpenReadStream();
    using var sceneMemoryStream = new MemoryStream();
    sceneSourceStream.CopyTo(sceneMemoryStream);
    var imageSceneData = sceneMemoryStream.ToArray();

    // Your implementation code
    throw new NotImplementedException();

    // La méthode ci-dessous permet de retourner une image depuis un tableau de bytes,
var imageData = new bytes[];
    var imageData = new byte[]{};
    return Results.File(imageData, "image/jpg");

}).DisableAntiforgery();
```

---

10. Intégration continue avec GitHubAction (3 points)

A réaliser :

1. Ajouter un fichier «.gitignore » adapté à votre projet C#.
2. Open sourcer votre projet sur github en suivant la convention https://github.com/your-account/VotrePrénom-VotreNom-ObjectDetection
3. **Important :** Mettre à la racine de votre projet un README.md avec dedans un lien vers votre projet github. Cela permettra de retrouver votre github pour l'évaluation.
4. Ajouter un fichier « global.json » adapté à votre version du SDK.
5. Le but de cette question est de créer une build d'intégration continue en utilisant les GithubAction. Ajouter un fichier « ./.github/wokflows/main.yml » dans votre projet. Configurer le fichier pour qu'il :
    a. Télécharge la bonne version du SDK dotnet
    b. Exécute le test unitaire (sans se soucier de la couverture de code)
    c. Publie votre projet Console en sortie de votre build. Le projet publié devra être un .exe autonome compatible Windows 10.