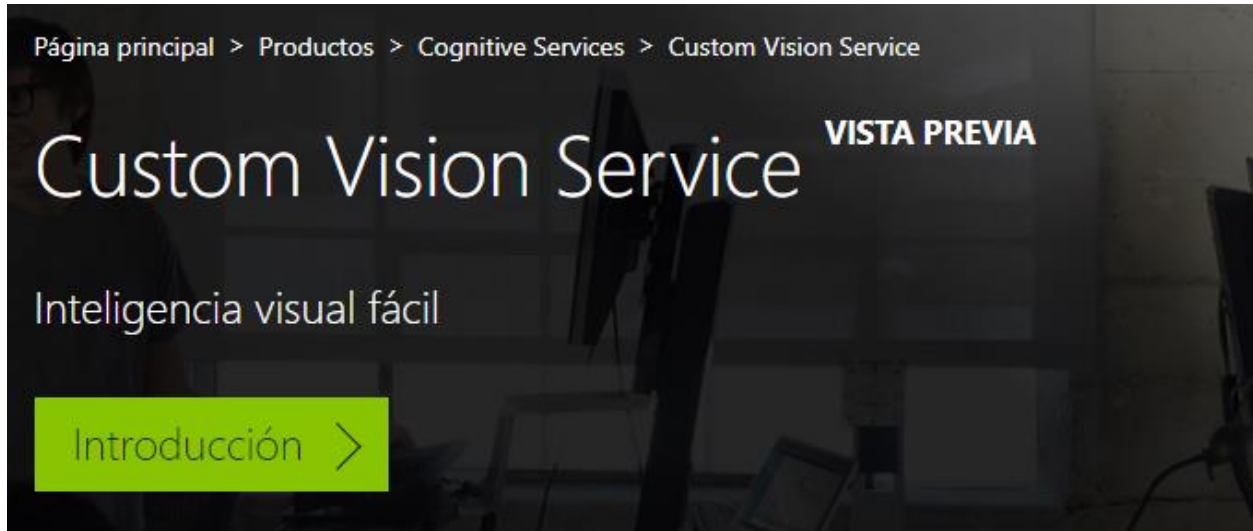


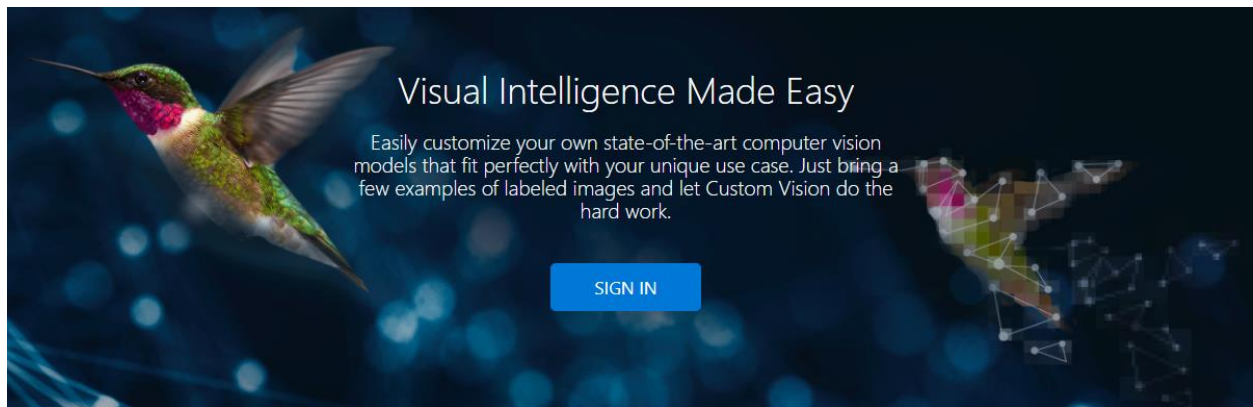
Práctica de Custom Vision API y Xamarin

URL del repositorio: <https://github.com/icebeam7/Pokedex>

Parte 1. Custom Vision API




Paso 1. Ingresa a <https://www.customvision.ai/> y da clic en **Sign In** para comenzar el registro al servicio.



Paso 2. Regístrate en el servicio con tu cuenta de Outlook/Hotmail/Live


Microsoft Custom Vision

Which account do you want to use?



Luis Antonio Beltran Prieto
luisantonio.beltranprieto@studentp...
Signed in

...

 Use another account

Paso 3. Proporciona los permisos solicitados dando clic en Aceptar.

Microsoft Custom Vision

App publisher website: microsoft.onmicrosoft.com

Microsoft Custom Vision needs permission to:

- View your basic profile ?
- View your email address ?
- Sign in as you ?

You're signed in as:

luisantonio.beltranprieto@studentpartner.com

[Show details](#)

Accept

Cancel

Paso 4. Acepta los términos del servicio.

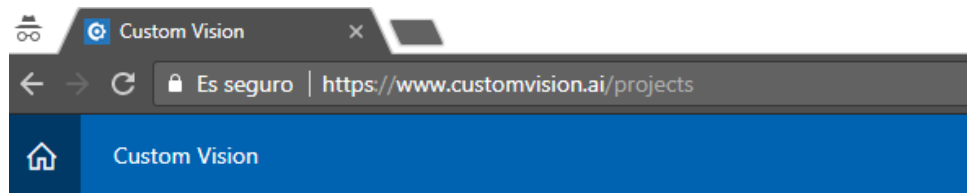
Terms of Service

Please note that Microsoft may retain copies of images uploaded for service improvement purposes. We won't publish your images or let other people use them.

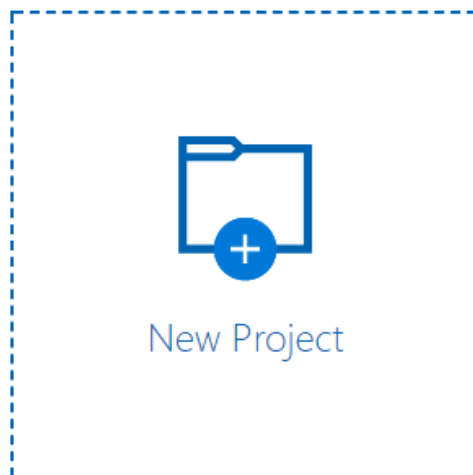
☒ I agree to the [Microsoft Cognitive Services Terms](#) and [Microsoft Privacy Statement](#)

I agree

Paso 5. Una vez configurado el entorno de trabajo, da clic en **New Project**.



My Projects



You have no projects yet, create one!

Paso 6. Ingresa los datos mostrados en la imagen siguiente (**Name, Description y Domain**) y da clic en **Create Project**.

New project

Name

Pokedex

Description

¿Quién es ese Pokémon?

Domains ⓘ

- ☒ General
- ☐ Food
- ☐ Landmarks
- ☐ Retail
- ☐ Adult
- ☐ General (compact)

Cancel

Create project

El clasificador que vamos a construir básicamente consiste en 3 pasos: **Subir imágenes, entrenar el proyecto y consumir el servicio.**

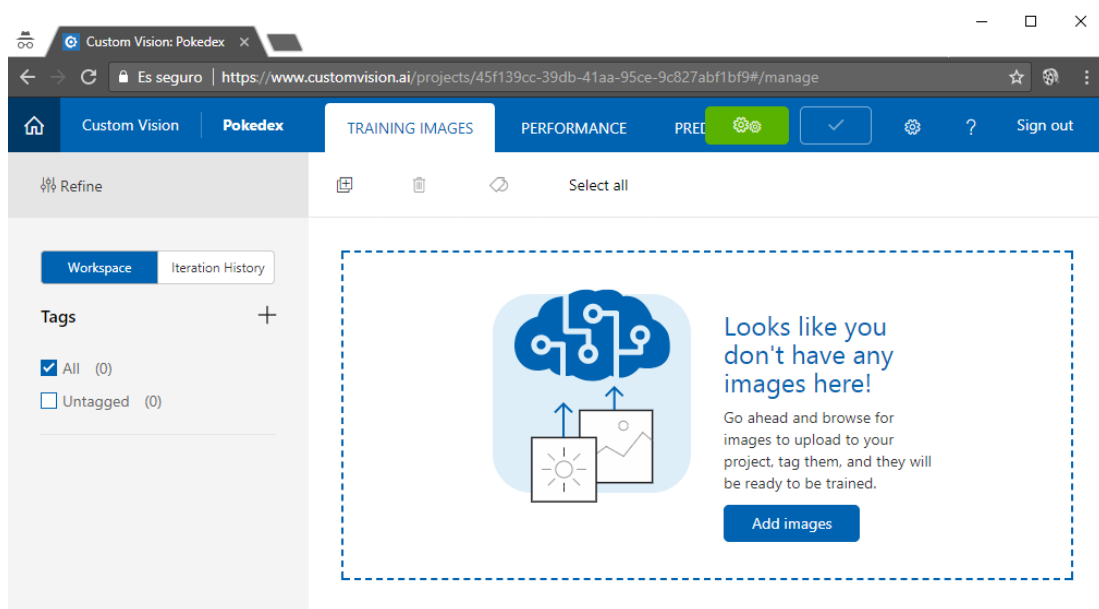


Improve your classifier!

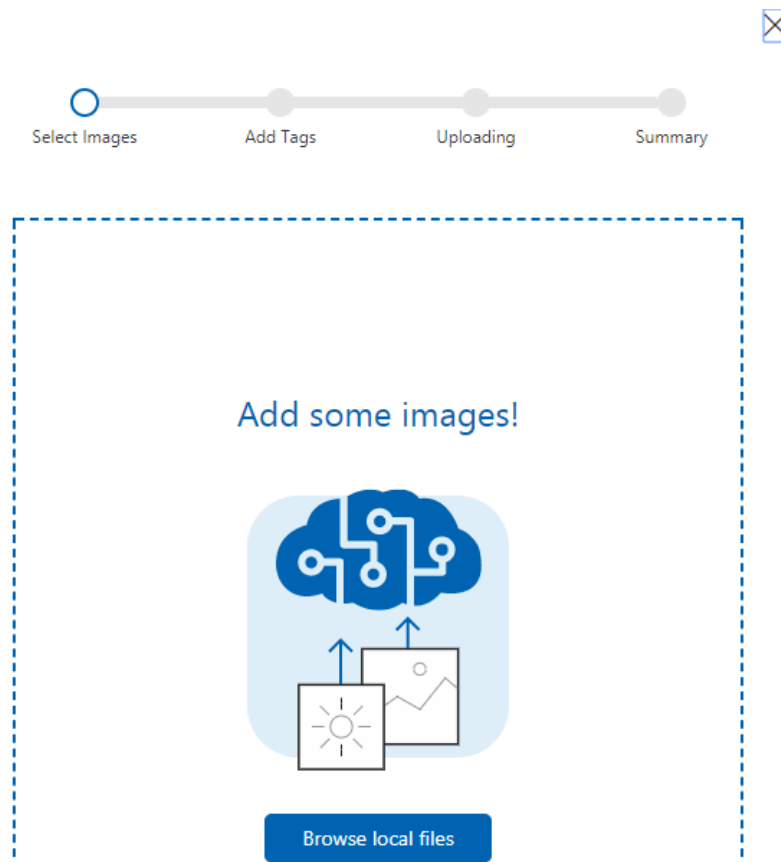
To get started follow these steps:

- 1 Upload your images**
First step is to add some images to your project and tag them. You'll need a minimum of 5 tagged images.
[Go](#)
- 2 Train your project**
Before you can setup your prediction endpoint, you will need to train your project first.
[Train](#)
- 3 Start using your prediction endpoint**
Once your project is trained, start using your prediction endpoint which will allow you to send images directly to your project for prediction

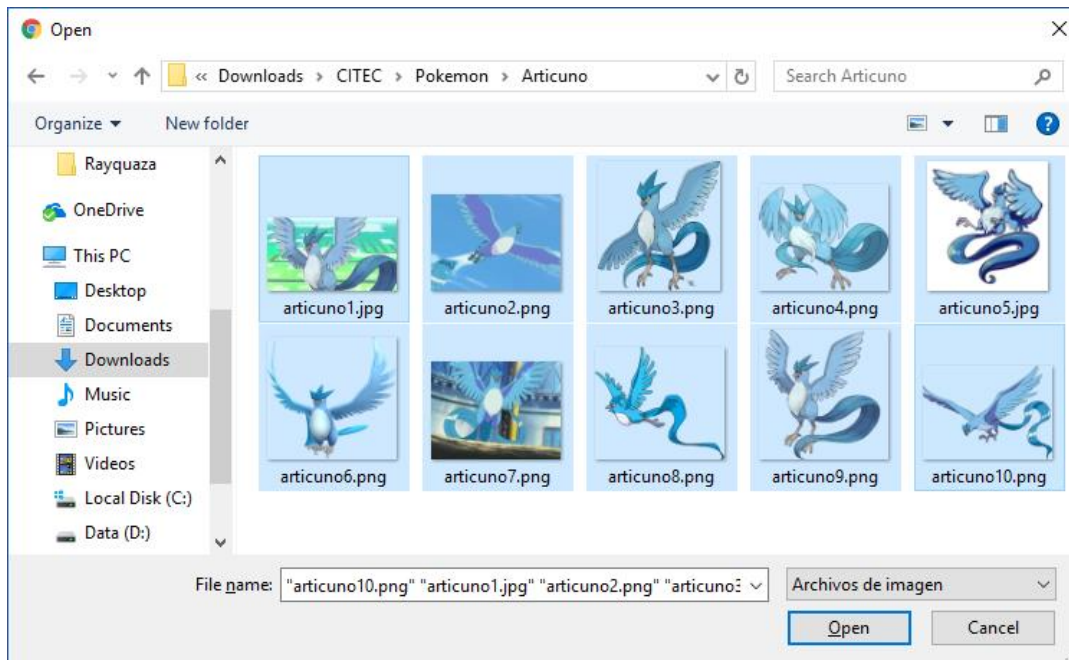
Paso 7. Da clic en el botón **Add images**.



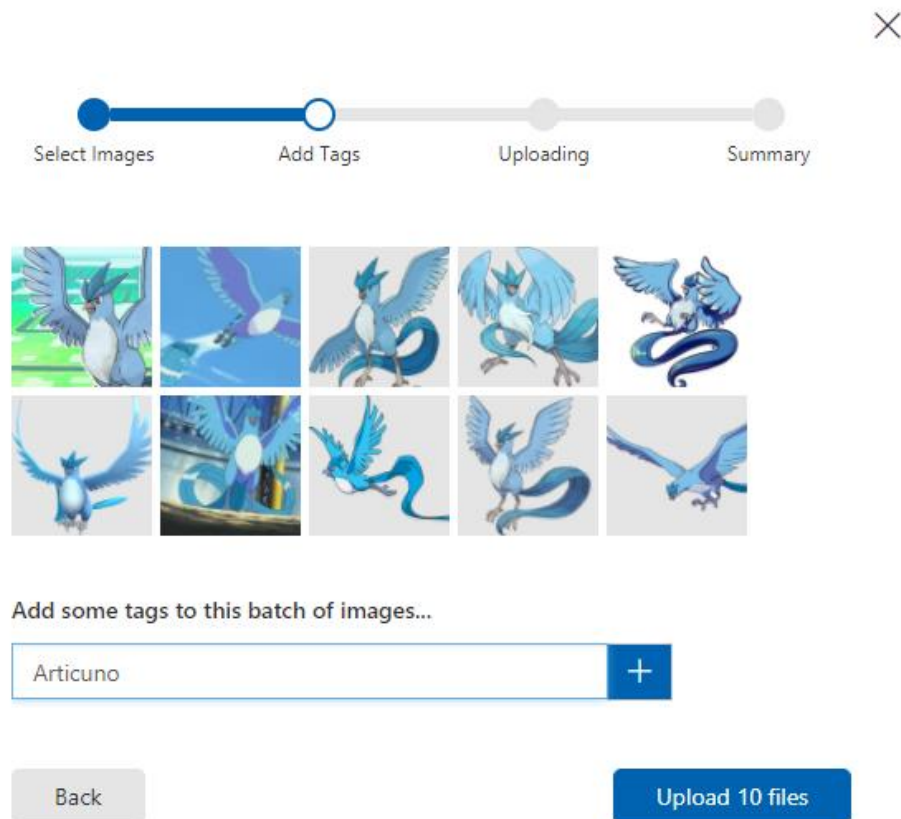
Paso 8. Da clic en el botón **Browse local files**.



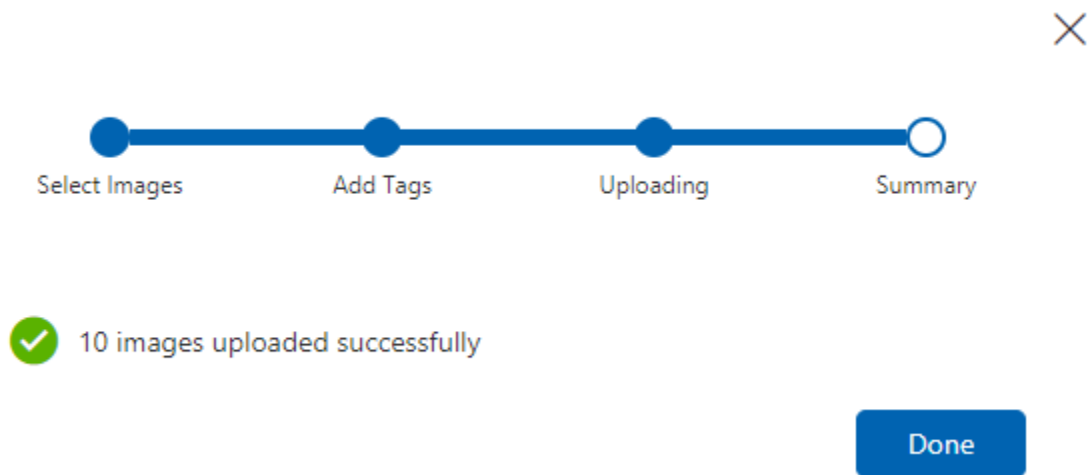
Paso 9. Selecciona las 10 imágenes proporcionadas en la carpeta **Articuno**.



Paso 10. Agrega la etiqueta (tag) **Articuno** y da clic en **Upload 10 files**.

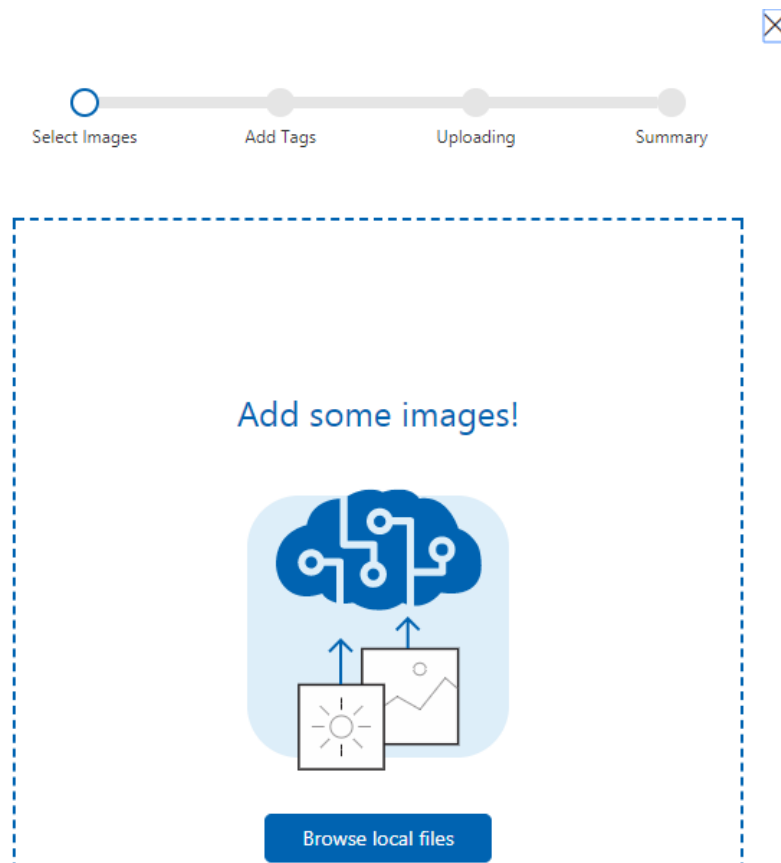


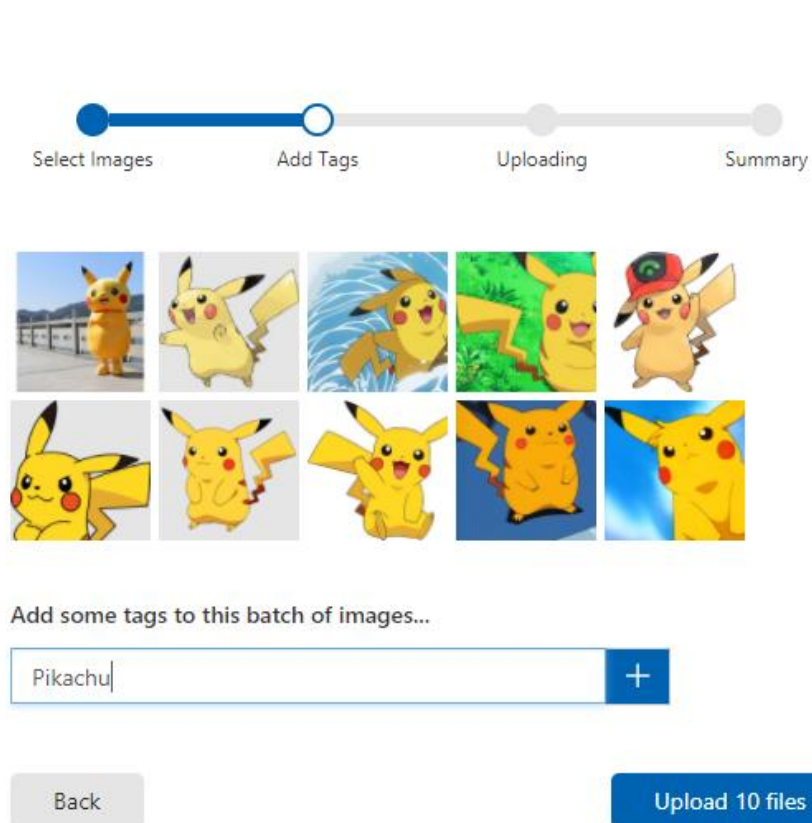
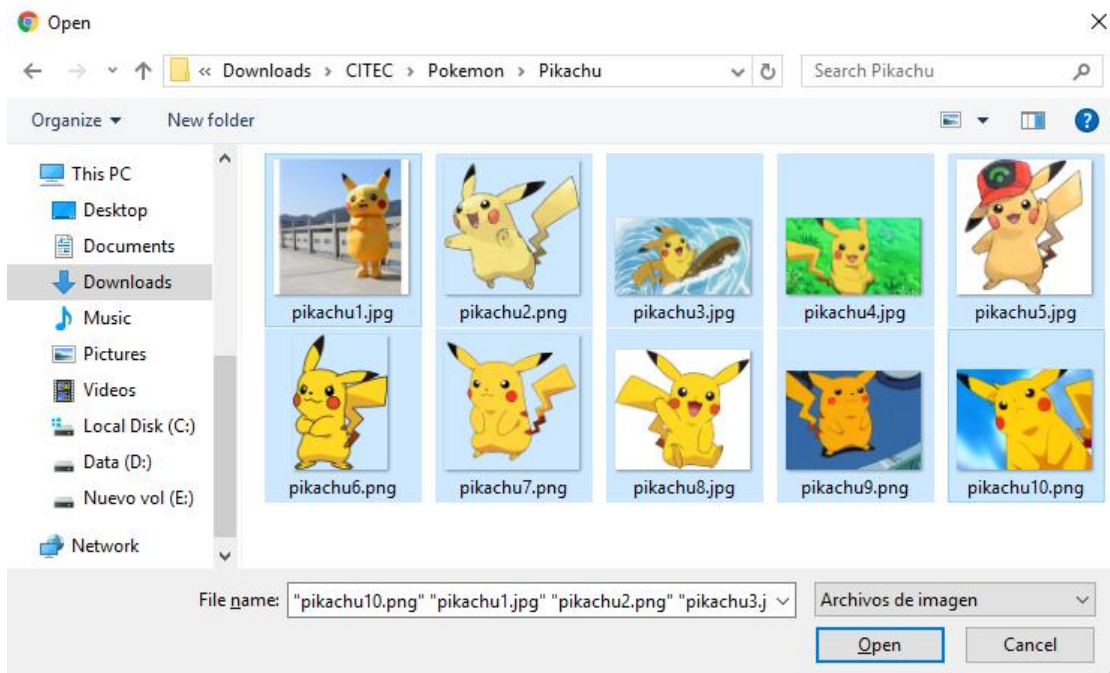
Una vez cargadas las imágenes en el Proyecto, recibirás el siguiente mensaje de éxito.




Paso 11. Repite el proceso (pasos 7 al 10) para las carpetas **Pikachu** y **Rayquaza** con sus etiquetas respectivas. Observa las siguientes imágenes:

Pikachu:








Select Images

Add Tags


Uploading

Summary

 10 images uploaded successfully

Done

Rayquaza:



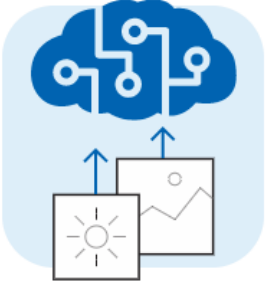
Select Images

Add Tags

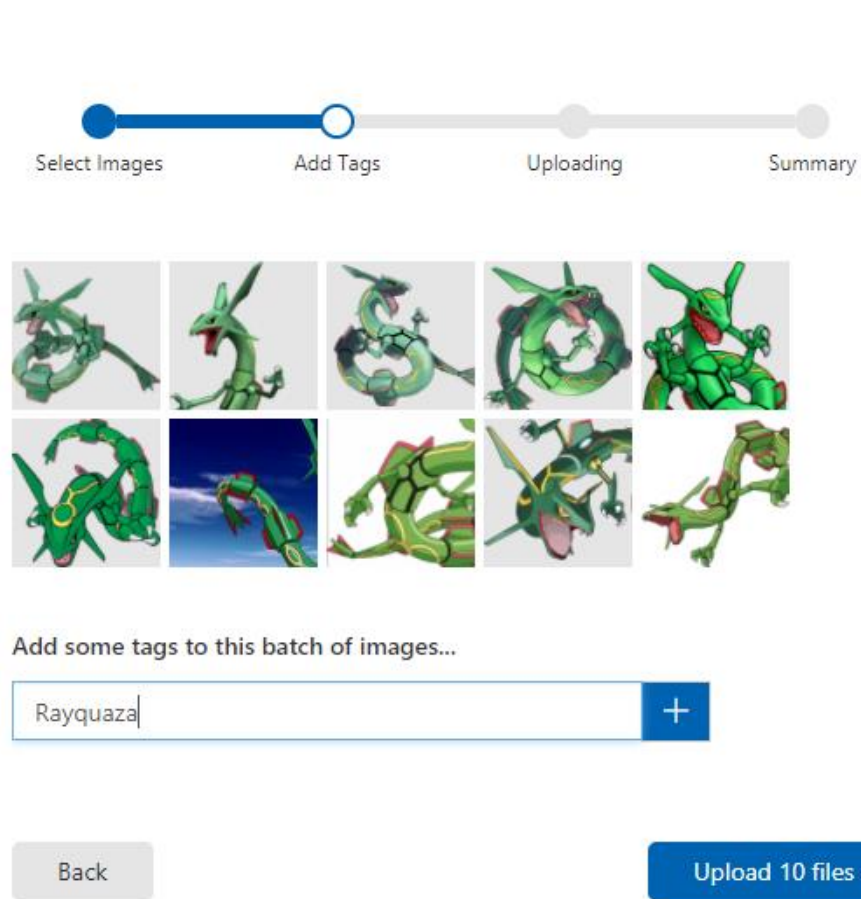
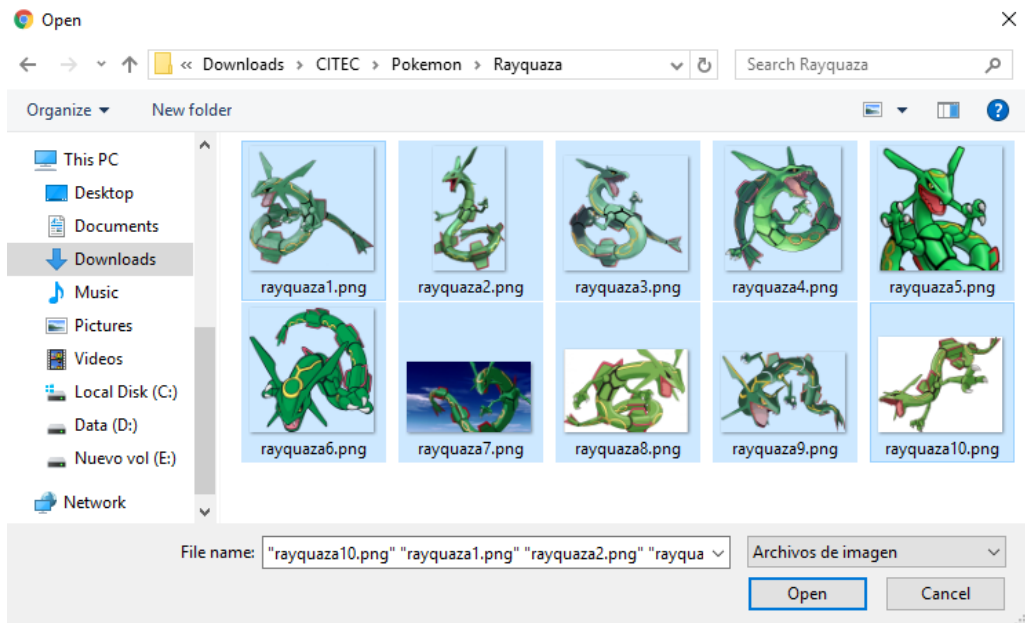
Uploading

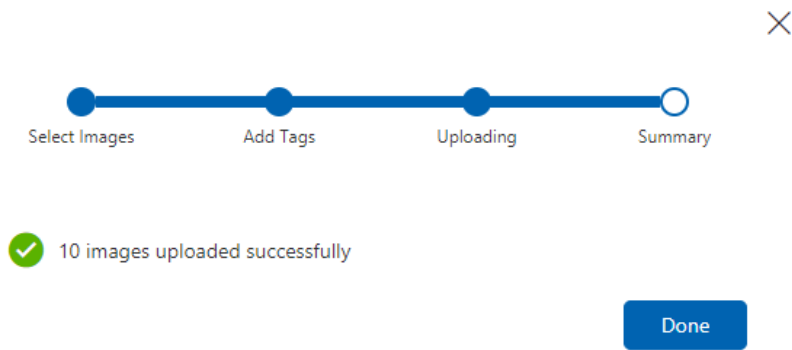
Summary

Add some images!



Browse local files





Una vez concluida la carga de imágenes, observa el detalle por etiqueta:

Workspace

Iteration History

Tags

+

☐ All (30)

☒ Articuno (10) ...

☒ Pikachu (10) ...

☒ Rayquaza (10) ...

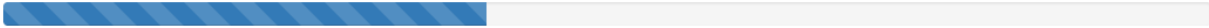
☐ Untagged (0)

Paso 12. Da clic en el botón **Train**.

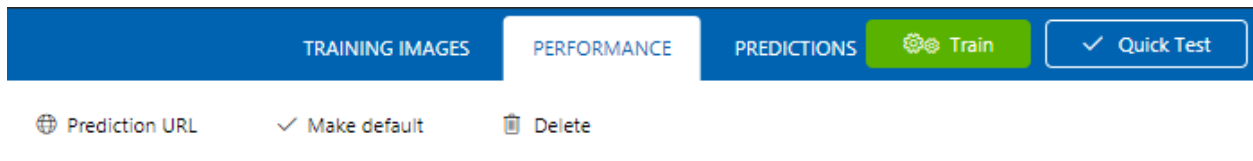


Observa el proceso de entrenamiento

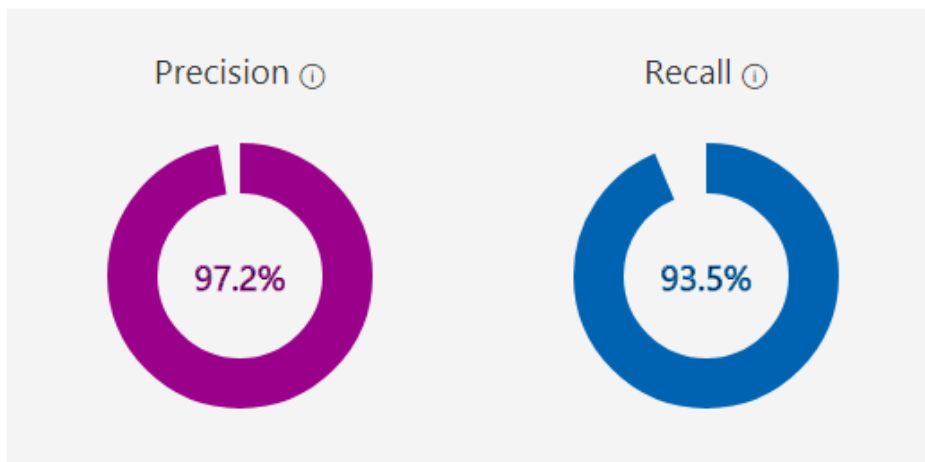
Iteration 1



Una vez finalizado, aparecerá la siguiente pantalla de resumen del proceso:



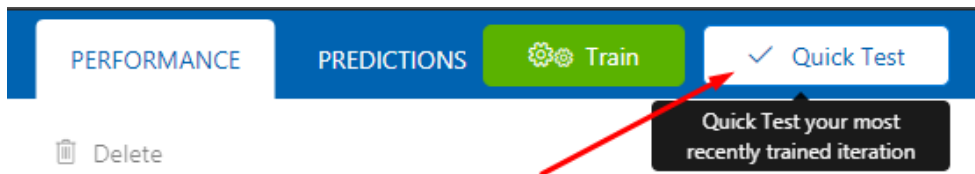
Iteration 1



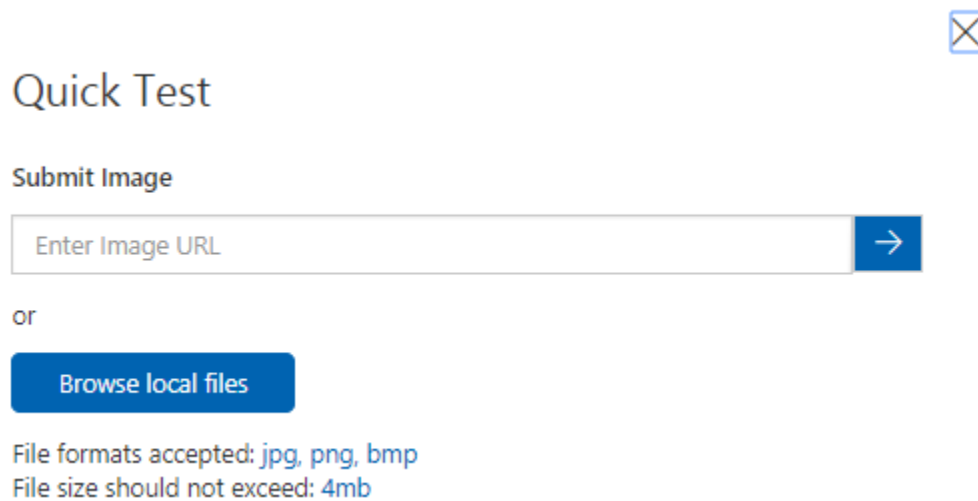
Performance Per Tag

Tag	Precision	Recall
Rayquaza	100.0%	100.0%
Articuno	93.3%	88.9%
Pikachu	100.0%	91.7%

Paso 13. Para verificar el modelo creado, da clic en el botón **Quick Test**.



Paso 14. Da clic en el botón **Browse local files**.



Paso 15. De la carpeta **Test**, selecciona **una a una** las imágenes correspondientes a los primeros 6 tests. Observa los resultados.

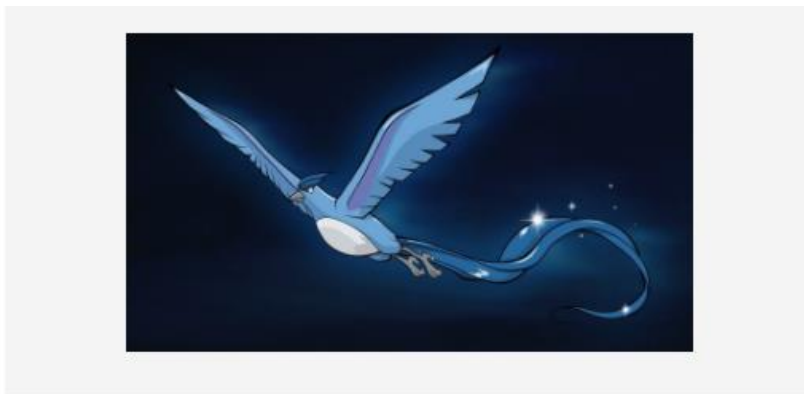
Test 1: OK



Results

Tag	Probability
Pikachu	99.6%
Rayquaza	0%
Articuno	0%

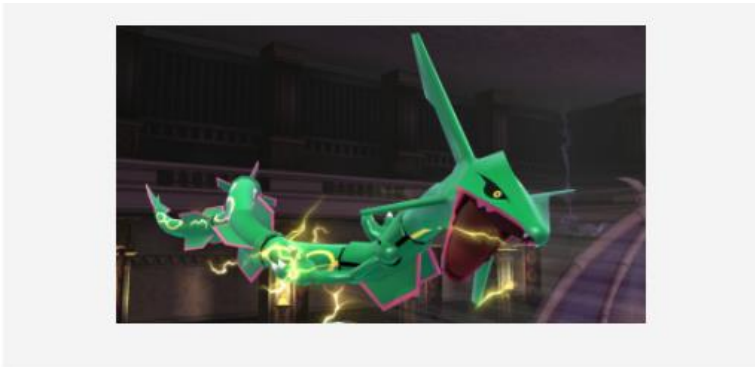
Test 2: OK



Results

Tag	Probability
Articuno	99.9%
Pikachu	0%
Rayquaza	0%

Test 3: OK



Results

Tag	Probability
Rayquaza	99.6%
Pikachu	0%
Articuno	0%

Test 4: OK



Results

Tag	Probability
Articuno	0%
Rayquaza	0%
Pikachu	0%

Test 5: Incorrecto



Results

Tag	Probability
Pikachu	98.2%
Articuno	0%
Rayquaza	0%

Test 6: Incorrecto



Results

Tag	Probability
Rayquaza	99.9%
Pikachu	0%
Articuno	0%

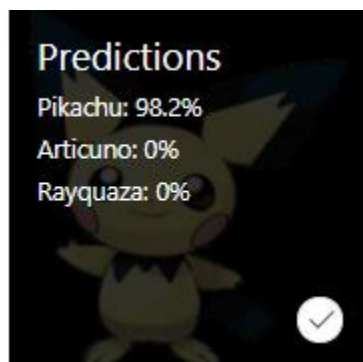
Paso 16. Vamos a introducir otra categoría para mejorar el modelo. Da clic en **Predictions**.



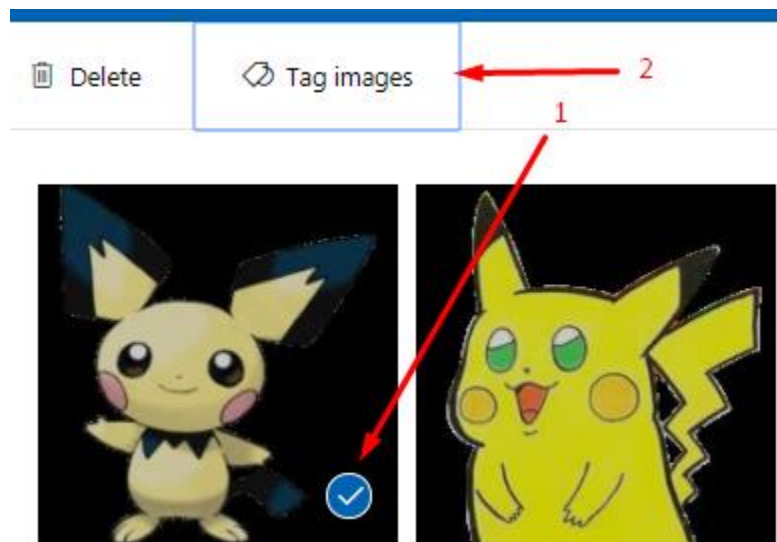
Observa que aparecen los tests realizados en el paso anterior.



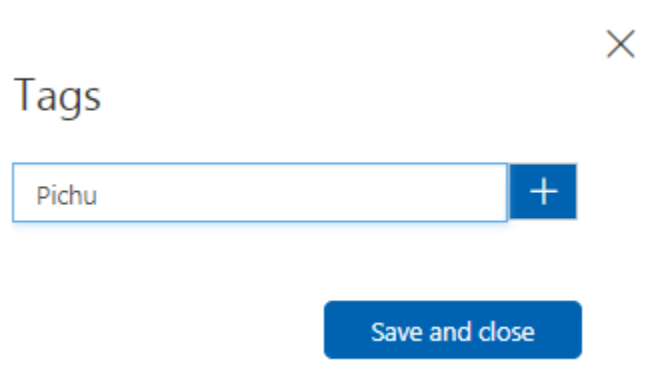
Si das clic en alguna imagen, aparece el resultado de la predicción



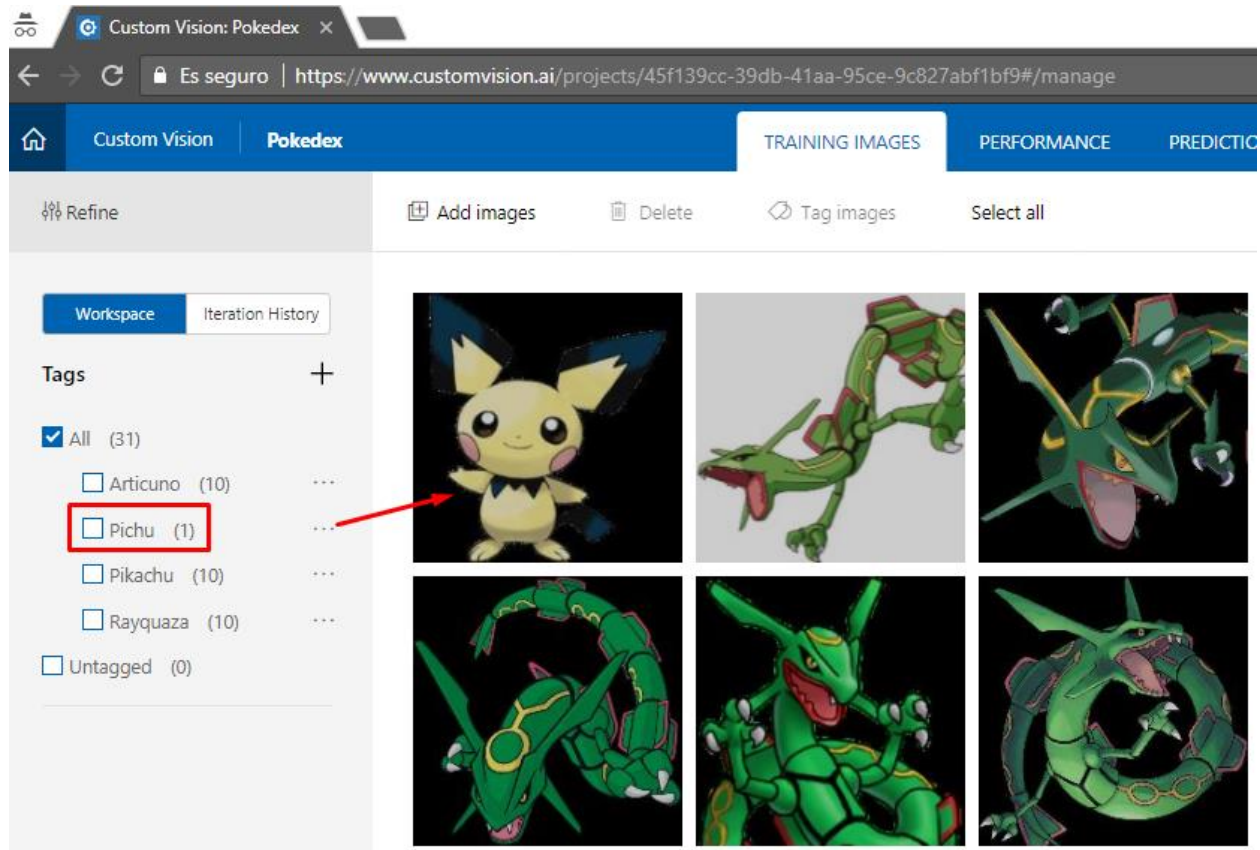
Paso 17. Selecciona la imagen mostrada en la figura y da clic en el botón **Tag images**.




Paso 18. Coloca el Tag **Pichu** y da clic en **Save and close**.

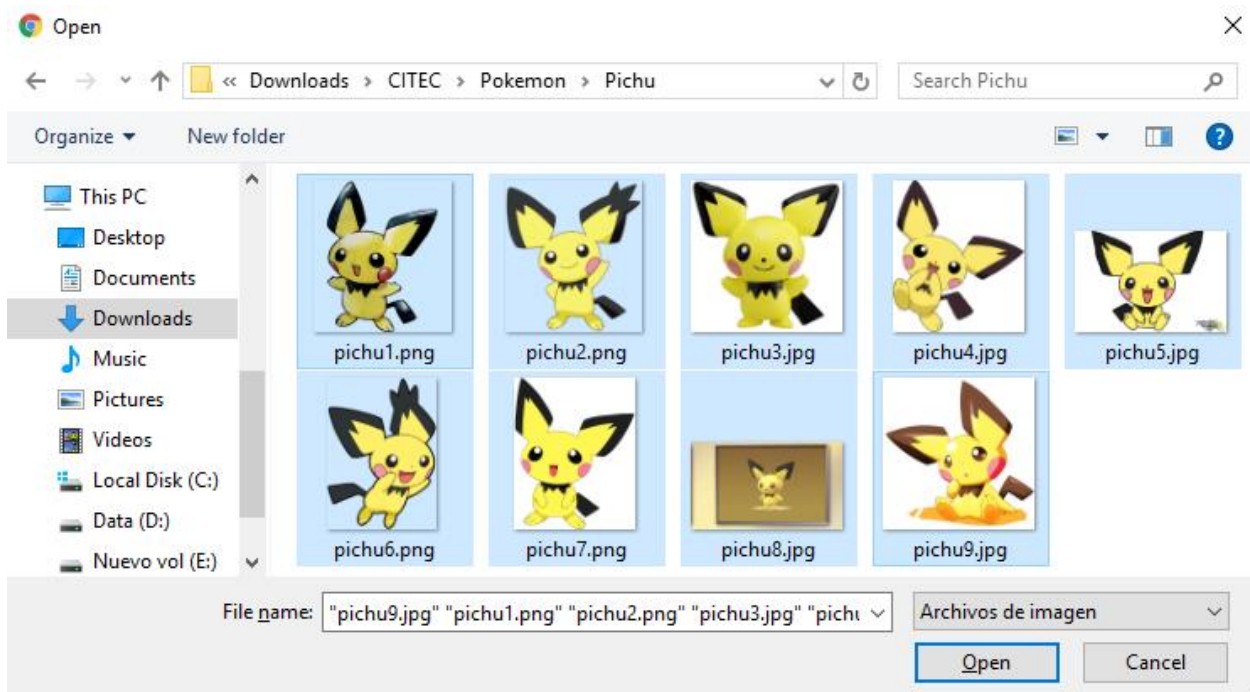
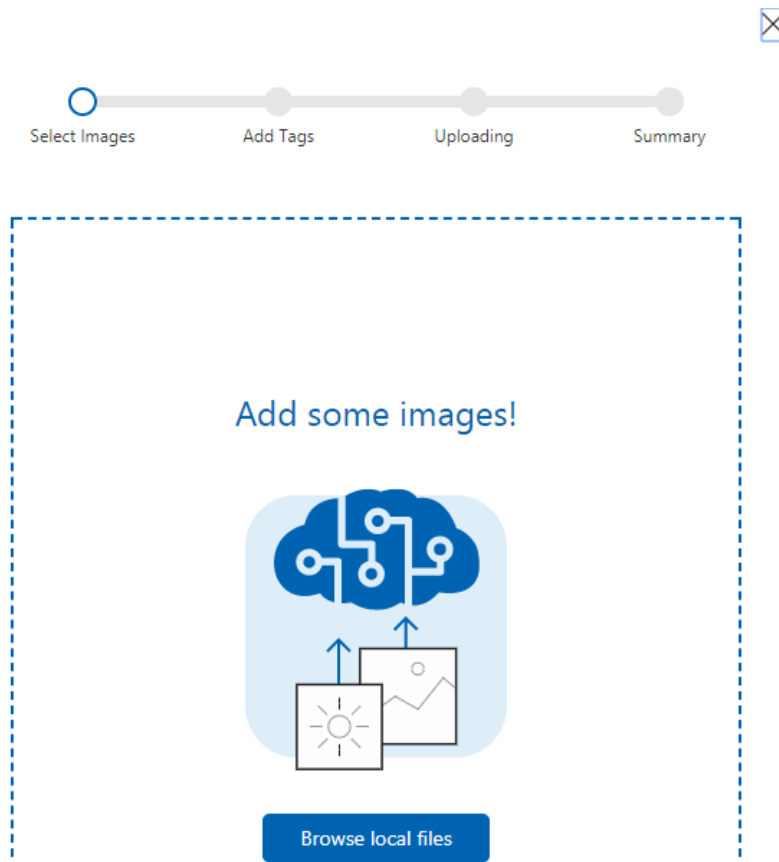


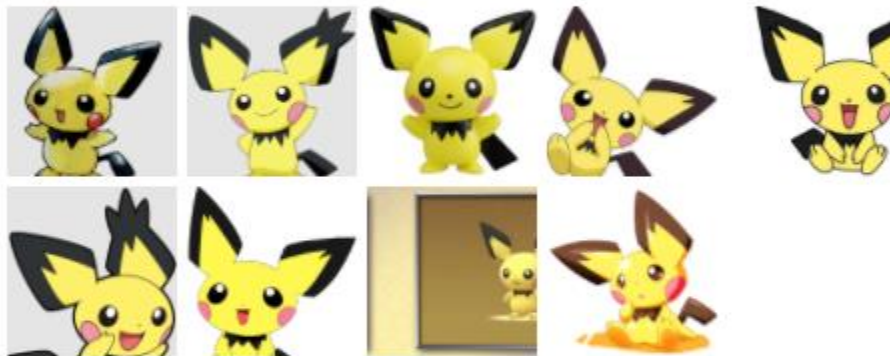
Observa que la imagen ha sido movida a la sección **Training images**.



Paso 19. Da clic en el botón **Add images** y repite el proceso que ya conoces para agregar las imágenes de la carpeta **Pichu** incluyendo dicha etiqueta. Observa las imágenes.

 Add images





Add some tags to this batch of images...

Pichu



Pichu

Back

Upload 9 files



✓ 9 images uploaded successfully

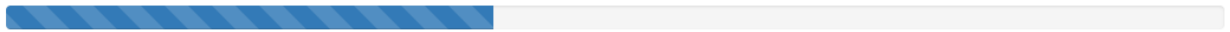
Done

Paso 20. Vuelve a dar clic en el botón **Train**.



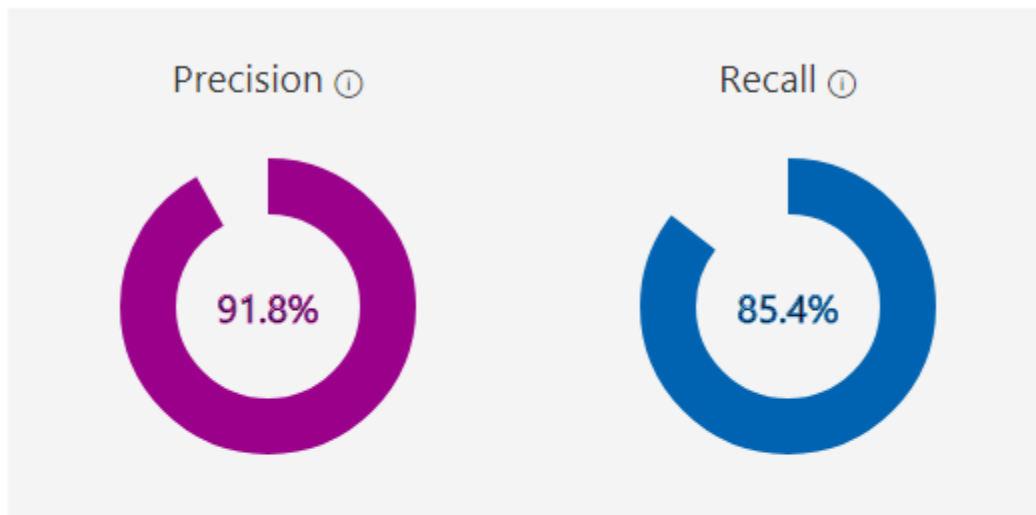
Observa que una nueva iteración (experiment) es creada.

Iteration 2



Al finalizar el entrenamiento, aparecerá nuevamente el resumen del proceso.

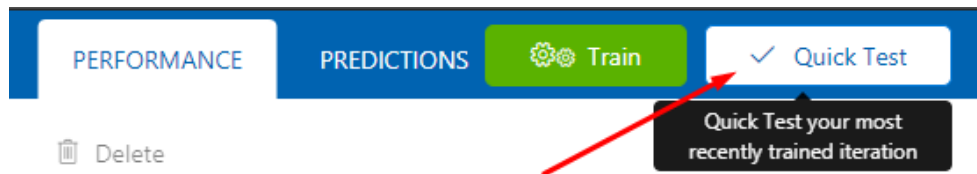
Iteration 2



Performance Per Tag

Tag	Precision	Recall
Rayquaza	100.0%	100.0%
Pichu	83.3%	91.7%
Articuno	91.7%	100.0%
Pikachu	100.0%	50.0%

Paso 21. Da clic de nuevo en **Quick Test**.



Paso 22. Utiliza el botón **Browse local files** y selecciona de la carpeta **Test** las imágenes 7 y 8. Observa los resultados.

A screenshot of the 'Quick Test' interface. At the top right is a close button (X in a blue square). The title 'Quick Test' is centered. Below it is the 'Submit Image' section, which includes a text input field with the placeholder 'Enter Image URL' and a blue arrow button to its right. Below the input field is the word 'or'. Underneath is a blue button labeled 'Browse local files'. At the bottom, there is text indicating 'File formats accepted: jpg, png, bmp' and 'File size should not exceed: 4mb'.

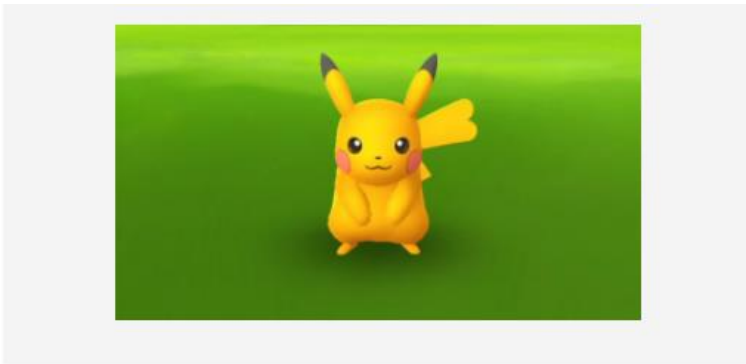
Test 7: OK



Results

Tag	Probability
Pichu	97.3%
Rayquaza	0%
Pikachu	0%
Articuno	0%

Test 8: OK

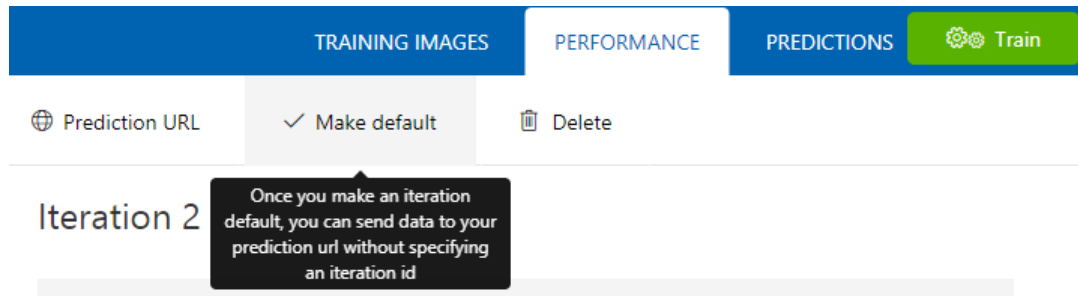


Results

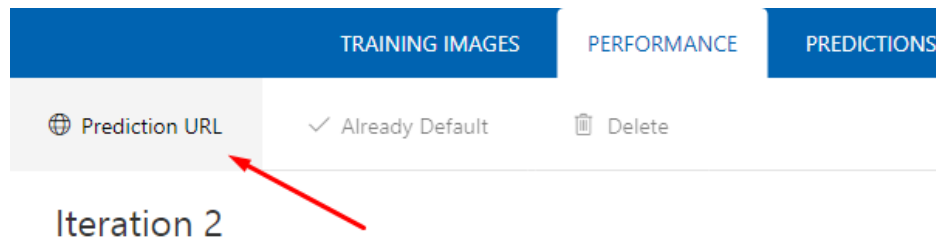
Tag	Probability
Pikachu	99.7%
Pichu	0%
Rayquaza	0%
Articuno	0%

Para poder consumir desde una aplicación el modelo generado, se necesita obtener la URL del servicio.

Paso 23. Da clic en el botón **Make default** de la pestaña **Performance**.



Paso 24. Da clic en el botón **Prediction URL**.



Paso 25. Copia la URL que aparece en la sección image file, así como la **Prediction Key** (los usaremos en la Parte 2) y toma nota de las consideraciones de cómo debe hacerse la petición URL (encabezados del request).

×

How to use the Prediction API

If you have an image URL:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/Predictic
```

Set **Prediction-Key** Header to : **1c4156479aff4627bf4749b8756e81b7**
Set **Content-Type** Header to : **application/json**
Set Body to : **{"Url": "<image url>"}**

If you have an image file:

```
https://southcentralus.api.cognitive.microsoft.com/customvision/v1.0/Predictic
```

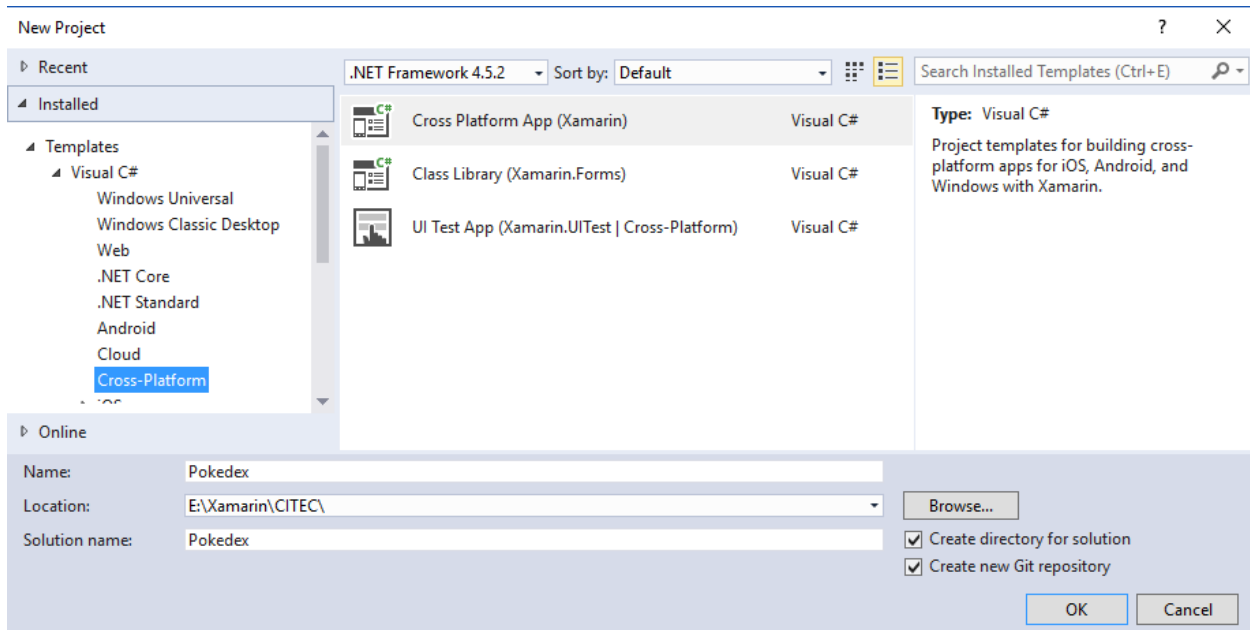
Set **Prediction-Key** Header to : **1c4156479aff4627bf4749b8756e81b7**
Set **Content-Type** Header to : **application/octet-stream**
Set Body to : **<image file>**

This iteration is marked as Default. If you mark another iteration as Default, the urls shown above will point to that iteration instead.

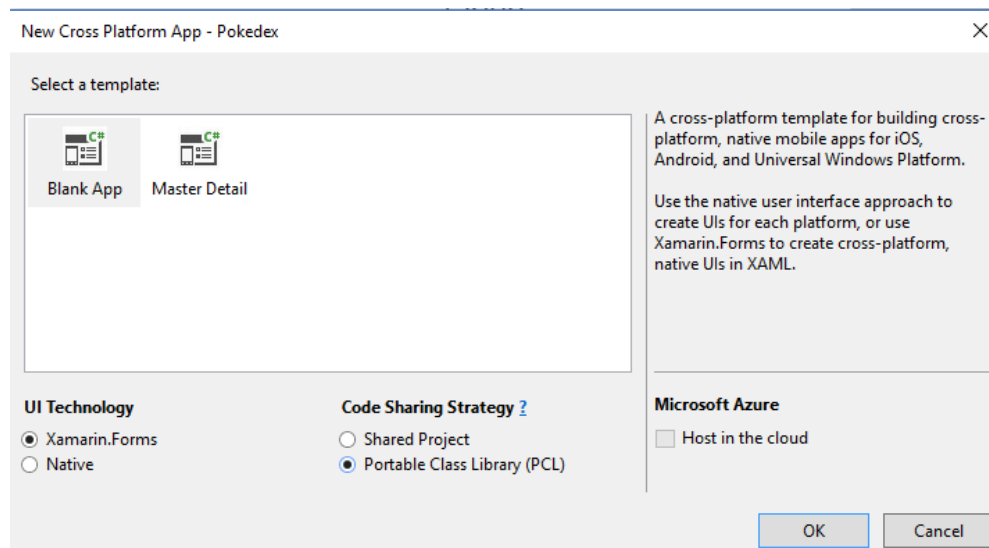
Got it!

Parte 2. Integración de Xamarin con Custom Vision API

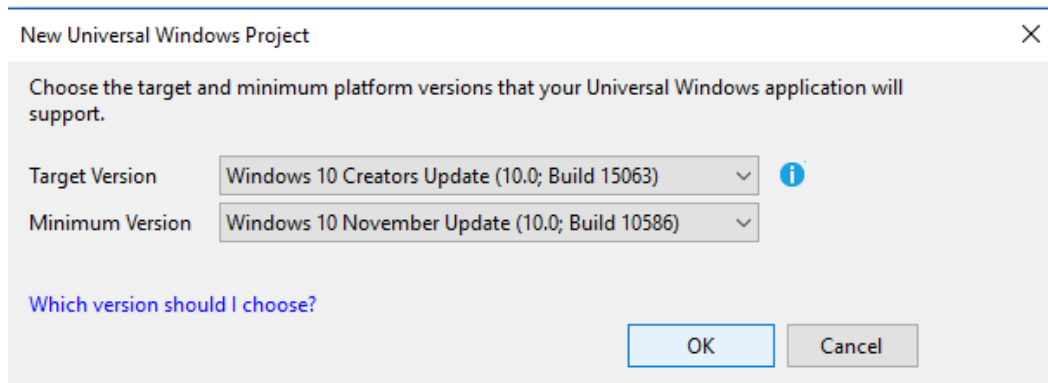
Paso 1. Abre **Visual Studio** y da clic en **Archivo → Nuevo Proyecto**. Selecciona la categoría **Cross-Platform → Cross Platform App (Xamarin)** y coloca el nombre de proyecto **Pokedex**.



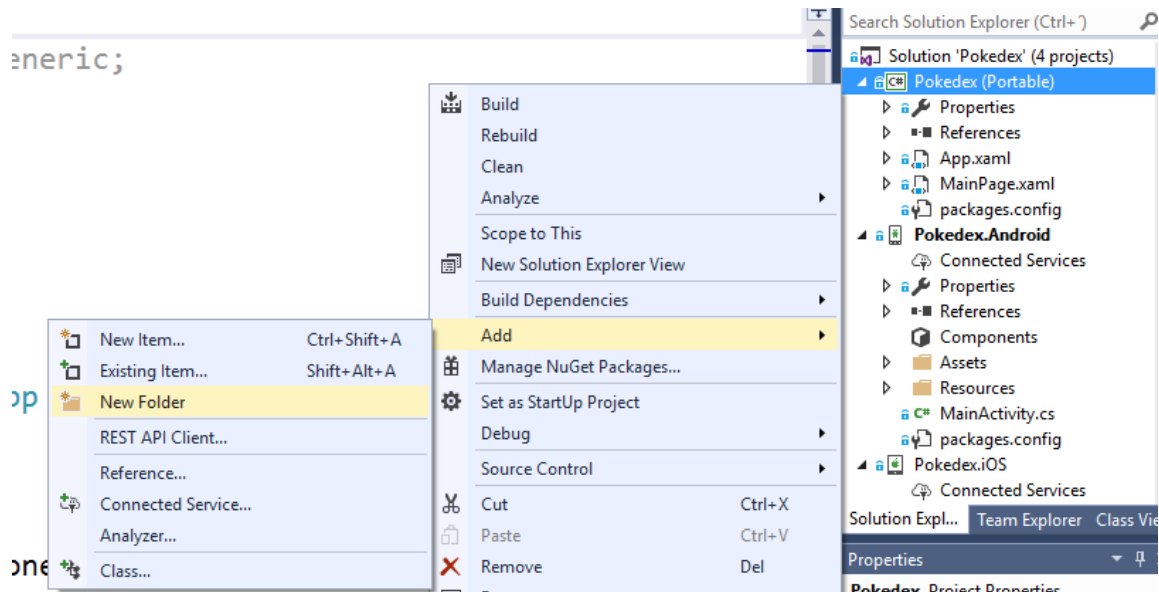
Paso 2. Selecciona **Portable Class Library (PCL)** para la estrategia de código compartido.



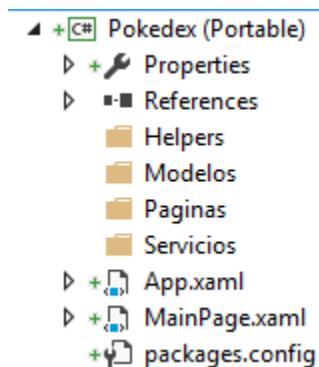
Paso 3. Si tienes instalado el SDK de Windows 10, simplemente da clic en Aceptar. En caso de que tengas una versión anterior, da clic en Cancelar.



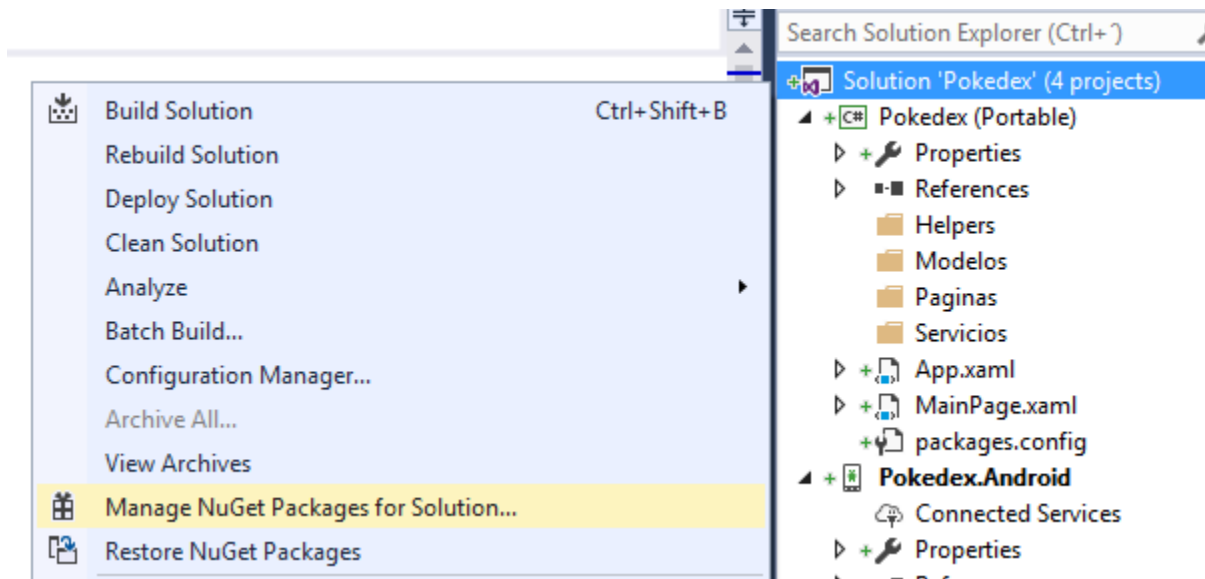
Paso 4. En el proyecto **Pokedex (Portable)**, da clic derecho y selecciona **Agregar → Nueva carpeta**.



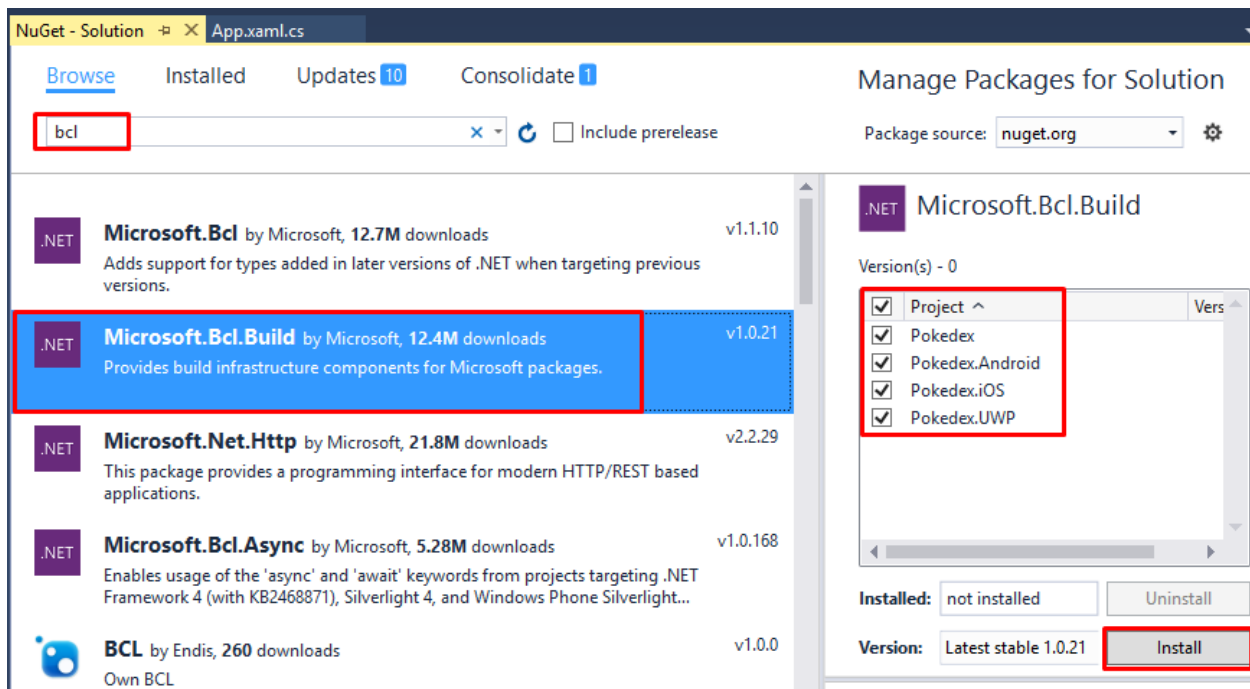
Paso 5. Agrega las siguientes carpetas al proyecto:



Paso 6. Da clic derecho en la solución **Pokedex** y elige **Administrar Paquetes Nuget** para la solución.



Paso 7. En la sección **Explorar**, escribe **bcl** e instala el paquete **Microsoft.Bcl.Build** para todos los proyectos (si te muestra un acuerdo de licencia de uso, acepta)



Paso 8. Ahora repite el proceso para instalar el paquete **Microsoft.Bcl**

The screenshot shows the NuGet Package Manager interface with the search filter set to 'bcl'. The list of packages includes Microsoft.Bcl (v1.1.10), Microsoft.Bcl.Build (v1.0.21), Microsoft.Net.Http (v2.2.29), Microsoft.Bcl.Async (v1.0.168), and BCL (v1.0.0). The right-hand pane displays the details for Microsoft.Bcl, showing it is not installed and the latest stable version is 1.1.10. The 'Install' button is highlighted.

Microsoft.Bcl by Microsoft, 12.7M downloads v1.1.10
Adds support for types added in later versions of .NET when targeting previous versions.

Microsoft.Bcl.Build by Microsoft, 12.4M downloads v1.0.21
Provides build infrastructure components for Microsoft packages.

Microsoft.Net.Http by Microsoft, 21.8M downloads v2.2.29
This package provides a programming interface for modern HTTP/REST based applications.

Microsoft.Bcl.Async by Microsoft, 5.28M downloads v1.0.168
Enables usage of the 'async' and 'await' keywords from projects targeting .NET Framework 4 (with KB2468871), Silverlight 4, and Windows Phone Silverlight...

BCL by Endis, 260 downloads v1.0.0
Own BCL

Microsoft.Bcl
Version(s) - 0

☒ Project ^
☒ Pokedex
☒ Pokedex.Android
☒ Pokedex.iOS
☒ Pokedex.UWP

Installed: not installed Uninstall

Version: Latest stable 1.1.10 Install

Paso 9. Posteriormente, instala el paquete **Microsoft.Net.Http**

The screenshot shows the NuGet Package Manager interface with the search filter set to 'bcl'. The list of packages includes Microsoft.Bcl (v1.1.10), Microsoft.Bcl.Build (v1.0.21), Microsoft.Net.Http (v2.2.29), Microsoft.Bcl.Async (v1.0.168), and BCL (v1.0.0). The right-hand pane displays the details for Microsoft.Net.Http, showing it is not installed and the latest stable version is 2.2.29. The 'Install' button is highlighted.

Microsoft.Bcl by Microsoft, 12.7M downloads v1.1.10
Adds support for types added in later versions of .NET when targeting previous versions.

Microsoft.Bcl.Build by Microsoft, 12.4M downloads v1.0.21
Provides build infrastructure components for Microsoft packages.

Microsoft.Net.Http by Microsoft, 21.8M downloads v2.2.29
This package provides a programming interface for modern HTTP/REST based applications.

Microsoft.Bcl.Async by Microsoft, 5.28M downloads v1.0.168
Enables usage of the 'async' and 'await' keywords from projects targeting .NET Framework 4 (with KB2468871), Silverlight 4, and Windows Phone Silverlight...

BCL by Endis, 260 downloads v1.0.0
Own BCL

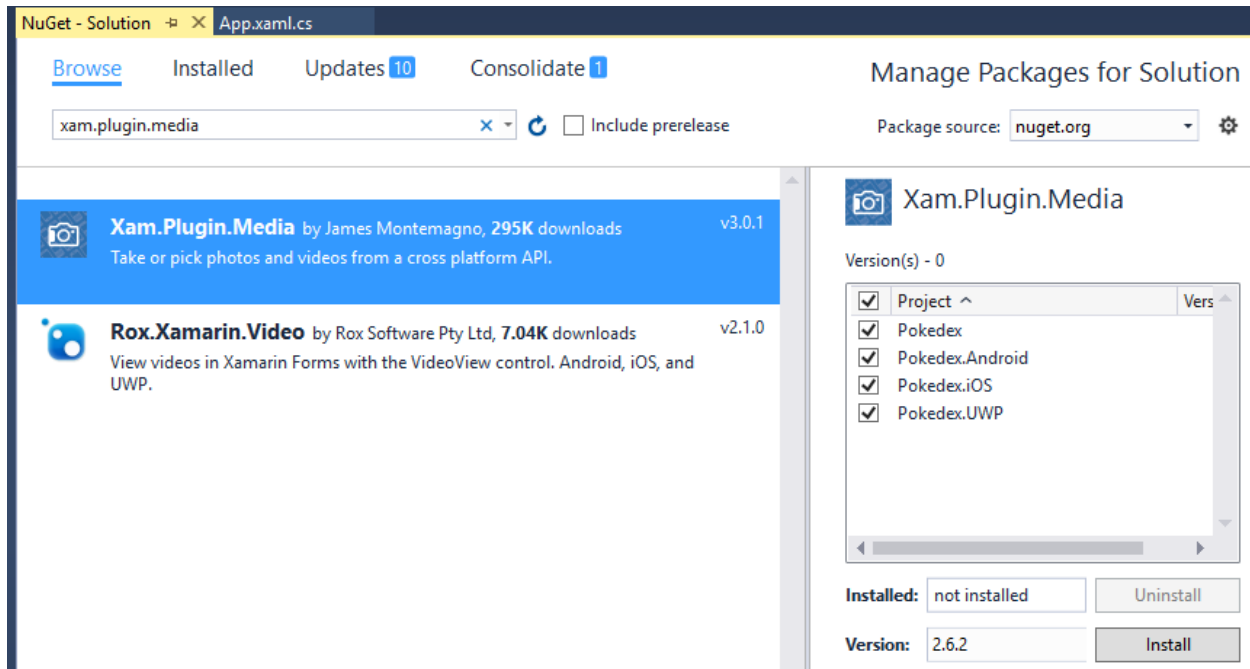
Microsoft.Net.Http
Version(s) - 0

☒ Project ^
☒ Pokedex
☒ Pokedex.Android
☒ Pokedex.iOS
☒ Pokedex.UWP

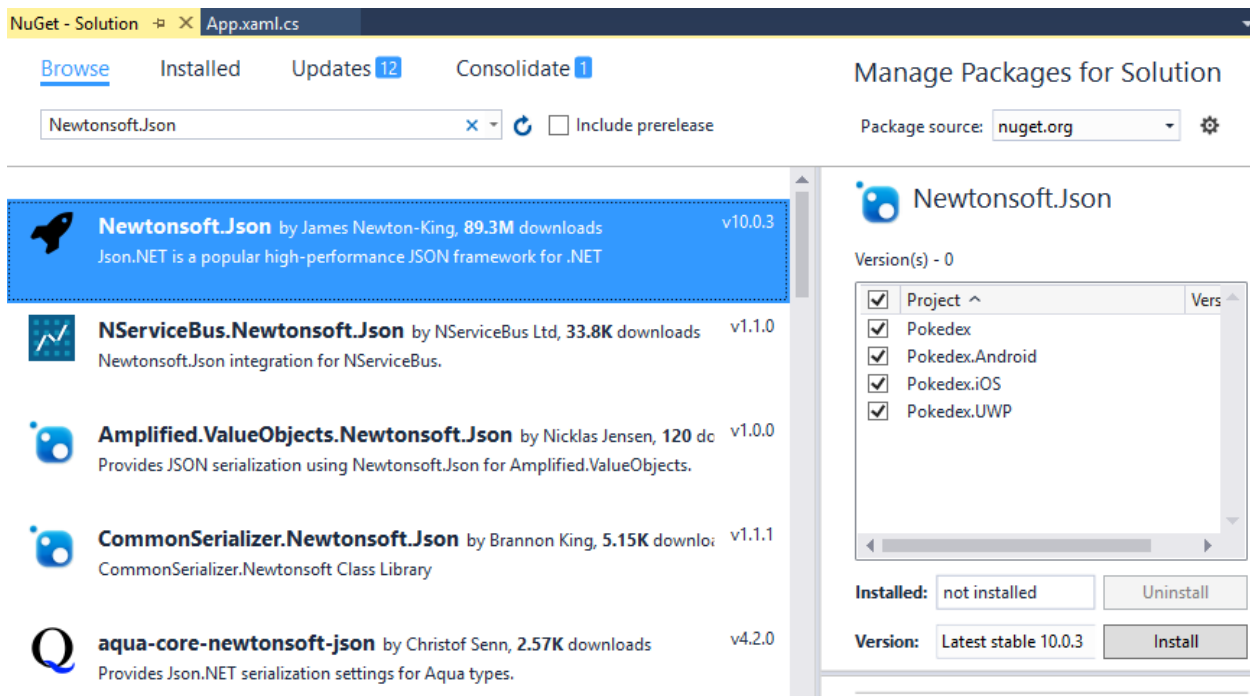
Installed: not installed Uninstall

Version: Latest stable 2.2.29 Install

Paso 10. Ahora busca el paquete **Xam.Plugin.Media**, selecciona todos los proyectos de la solución e instala la versión **2.6.2**



Paso 11. El último paquete por instalar (de momento) es **Newtonsoft.Json**. Búscalo en la lista y procede a instalarlo en todos los proyectos



Paso 12. Da clic derecho en la carpeta **Modelos**, selecciona **Agregar → Clase** y agrega la clase **Prediction**, con el código siguiente:

```
namespace Pokedex.Modelos
{
    public class Prediction
    {
        public string TagId { get; set; }
        public string Tag { get; set; }
        public double Probability { get; set; }
    }
}
```

Paso 13. En la misma carpeta, agrega la clase **CustomVisionResult** con el código siguiente

```
using System;
using System.Collections.Generic;

namespace Pokedex.Modelos
{
    public class CustomVisionResult
    {
        public string Id { get; set; }
        public string Project { get; set; }
        public string Iteration { get; set; }
        public DateTime Created { get; set; }
        public List<Prediction> Predictions { get; set; }
    }
}
```

Paso 14. Ahora, en la carpeta **Helpers**, agrega una clase llamada **Constantes**. Asigna los valores **PredictionKey** y **PredictionURL** de acuerdo a la información mostrada en el **paso 25** de la **Parte 1. Custom Vision API** de la práctica. También crea una constante llamada **PokemonNoIdentificado**, que usaremos más adelante en caso de que el servicio no pueda identificar al Pokémon.

```
namespace Pokedex.Helpers
{
    public static class Constantes
    {
        public const string PredictionKey = "aqui-va-tu-llave";
        public const string PredictionURL = "aqui-va-tu-url";
        public const string PokemonNoIdentificado = "Pokémon no identificado en la base de datos";
    }
}
```

Paso 15. Agrega una nueva clase en la carpeta **Servicios** con el nombre **ServicioClasificador**. El código es el siguiente:

```
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Pokedex.Helpers;
using System;
using Pokedex.Modelos;
using Newtonsoft.Json;
using System.Linq;

namespace Pokedex.Servicios
{
    public static class ServicioClasificador
    {
        public async static Task<string> ClasificarImagen(MemoryStream stream)
        {
            try {
                var url = Constantes.PredictionURL;

                using (var cliente = new HttpClient())
                {
                    cliente.DefaultRequestHeaders.Add("Prediction-Key",
Constantes.PredictionKey);

                    using (var content = new ByteArrayContent(stream.ToArray()))
                    {
                        content.Headers.ContentType = new
MediaTypeHeaderValue("application/octet-stream");
                        var post = await cliente.PostAsync(url, content);
                        var resultado = await post.Content.ReadAsStringAsync();
                        var cv =
JsonConvert.DeserializeObject<CustomVisionResult>(resultado);

                        if (cv.Predictions.Count > 0)
                        {
                            var prediccion = ObtenerPrediccion(cv);
                            return prediccion.Probability > 0.5 ? prediccion.Tag :
Constantes.PokemonNoIdentificado;
                        }
                        else
                            return "Error al realizar la predicción. Intenta de nuevo.";
                    }
                }
            }
            catch (Exception ex) { return "Ocurrió una excepción: " + ex.Message; }
        }

        static Prediction ObtenerPrediccion(CustomVisionResult cv)
        {
            return cv.Predictions.OrderByDescending(x => x.Probability).Take(1).First();
        }
    }
}
```

Paso 16. Crea una nueva clase llamada **ServicioImágenes** en la carpeta **Servicios**. El código es el siguiente:

```
using Plugin.Media;
using Plugin.Media.Abstractions;
using System.Threading.Tasks;

namespace Pokedex.Servicios
{
    public class ServicioImágenes
    {
        public static async Task<MediaFile> TomarFoto(bool usarCamara)
        {
            await CrossMedia.Current.Initialize();

            if (usarCamara)
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakePhotoSupported)
                {
                    return null;
                }
            }

            var file = usarCamara
                ? await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions
                {
                    Directory = "Clasificador",
                    Name = "test.jpg"
                })
                : await CrossMedia.Current.PickPhotoAsync();

            return file;
        }
    }
}
```

Paso 17. Ahora, en la carpeta **Paginas** da clic derecho y selecciona **Agregar → Nuevo elemento**. De la lista, selecciona la categoría **Xamarin.Forms** y elige **Content Page**. El nombre de esta página será **PaginaPokedex**.

PaginaPokedex.xaml (diseño)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="Pokedex.Paginas.PaginaPokedex"
              BackgroundColor="#DC0A2D">
    <ContentPage.Content>
        <StackLayout Padding="10">
            <StackLayout Orientation="Horizontal" HorizontalOptions="CenterAndExpand">
                <Button x:Name="btnCamara" x:Id="btnCamara" Text="Cámara"
                    Clicked="btnCamara_Clicked" Margin="10" WidthRequest="100" BackgroundColor="#000000"
                    TextColor="White"/>
                <Button x:Name="btnGaleria" x:Id="btnGaleria" Text="Galería"
                    Clicked="btnGaleria_Clicked" Margin="10" WidthRequest="100" BackgroundColor="#000000"
                    TextColor="White"/>
            </StackLayout>
            <Image x:Name="imgFoto" WidthRequest="150" HeightRequest="150"
                Aspect="AspectFit" HorizontalOptions="CenterAndExpand" Margin="5"/>
            <Button x:Name="btnClasificar" Text="¿Quién es ese Pokémon?"
                Clicked="btnClasificar_Clicked" HorizontalOptions="CenterAndExpand" Margin="5"
                BackgroundColor="#000000" TextColor="White"/>
            <ActivityIndicator x:Name="actBuscando" IsVisible="False" IsRunning="False"
                IsEnabled="False"/>

            <Label x:Name="lblResultado" Text="---" TextColor="#BAF73C" Margin="5"
                FontSize="Large"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

PaginaClasificador (code-behind)

```
using System;
using Xamarin.Forms;
using System.IO;
using Pokedex.Servicios;
using Xamarin.Forms.Xaml;
using System.Threading.Tasks;

namespace Pokedex.Paginas {
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PaginaPokedex : ContentPage {
        static MemoryStream streamCopy;

        public PaginaPokedex(){
            InitializeComponent();
        }

        void MostrarIndicador(bool mostrar) {
            actBuscando.IsVisible = mostrar;
            actBuscando.IsRunning = mostrar;
            actBuscando.IsEnabled = mostrar;
        }

        async Task ObtenerImagen(bool camara) {
            var archivo = await ServicioImagenes.TomarFoto(camara);
            lblResultado.Text = "---";
            imgFoto.Source = ImageSource.FromStream(() => {
                var stream = archivo.GetStream();
                streamCopy = new MemoryStream();
                stream.CopyTo(streamCopy);
                stream.Seek(0, SeekOrigin.Begin);
                archivo.Dispose();
                return stream;
            });
        }

        private async void btnCamara_Clicked(object sender, EventArgs e) {
            await ObtenerImagen(true);
        }

        private async void btnGaleria_Clicked(object sender, EventArgs e) {
            await ObtenerImagen(false);
        }

        private async void btnClasificar_Clicked(object sender, EventArgs e) {
            if (streamCopy != null) {
                MostrarIndicador(true);
                streamCopy.Seek(0, SeekOrigin.Begin);
                var resultado = await ServicioClasificador.ClasificarImagen(streamCopy);
                lblResultado.Text = $"Es... {resultado}";
            }
            else lblResultado.Text = "---No has seleccionado una imagen---";

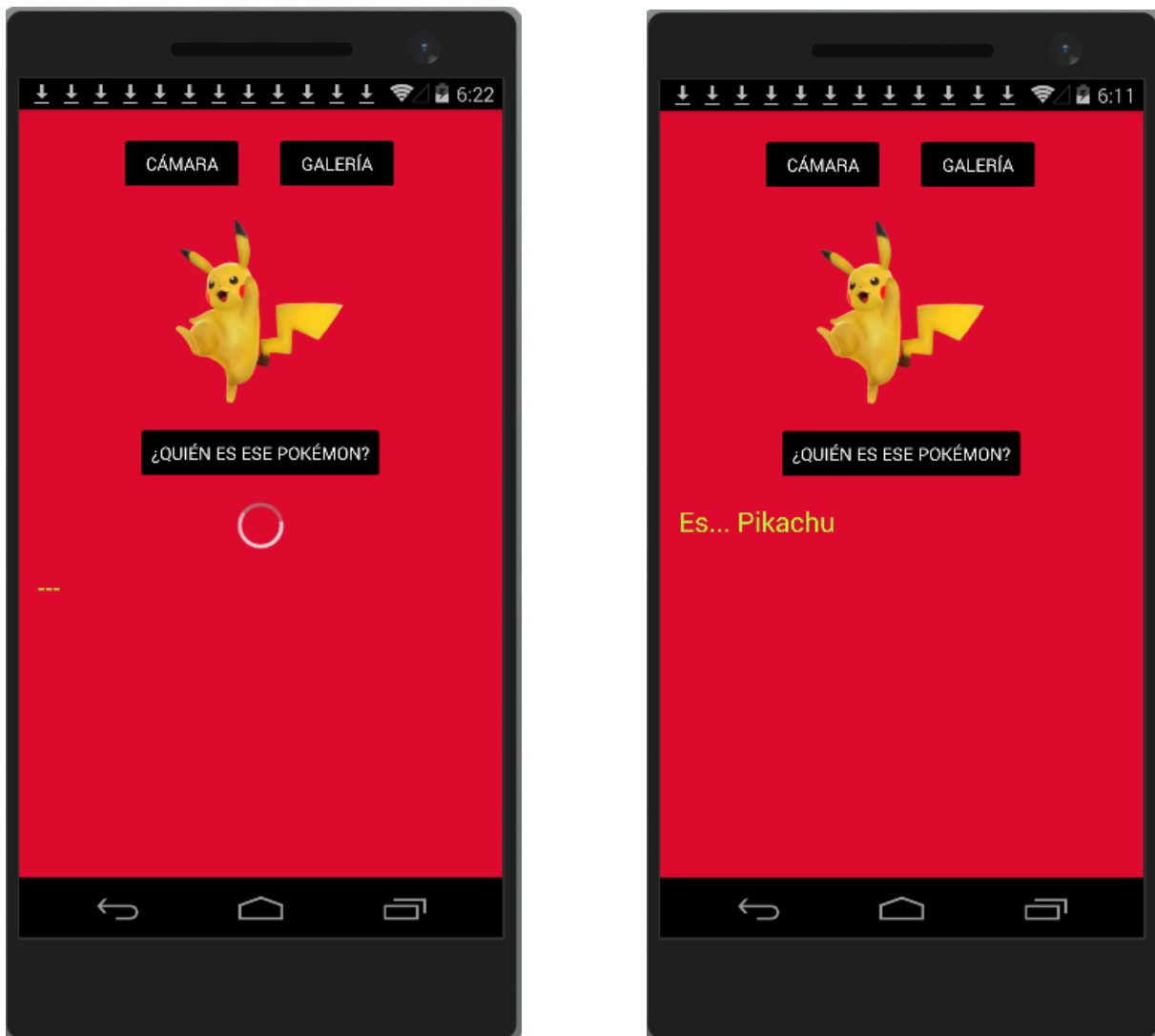
            MostrarIndicador(true);
        }
    }
}
```

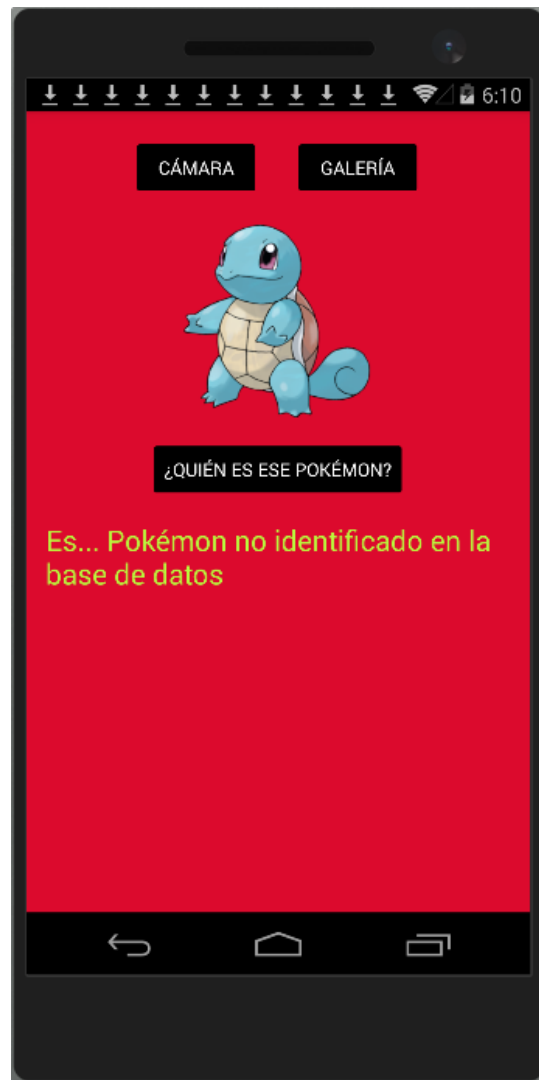
Paso 18. Modifica el constructor de la clase **App** para que la página inicial sea **PaginaPokedex**:

```
public App()
{
    InitializeComponent();

    MainPage = new Pokedex.Paginas.PaginaPokedex();
}
```

Paso 19. Compila y ejecuta la aplicación, verificando el correcto funcionamiento de la misma





Parte 3. Mejorando la aplicación:

- Obteniendo información adicional del Pokémon consultando un servicio en la nube (pasos 1-3)
- El Pokédex “habla”, mencionando la descripción del Pokémon (pasos 4-5)
- El Pokédex reproduce el sonido característico de cada Pokémon (pasos 6-10)

Paso 1. Agrega la clase **Pokemon** en la carpeta **Modelos**, con el siguiente código

```
namespace Pokedex.Modelos
{
    public class Pokemon
    {
        public string ID { get; set; }
        public string Nombre { get; set; }
        public string Tipo1 { get; set; }
        public string Tipo2 { get; set; }
        public string Especie { get; set; }
        public string Descripcion { get; set; }
    }
}
```

Paso 2. Agrega la siguiente línea dentro de la clase **Constantes** (en la carpeta **Helpers**)

```
public const string PokedexFunctionURL = "https://pokedex.azurewebsites.net/api/pokedex-function?code=EDbXA2CYF7DUh6DKY4ogrELCqSKwoh24NrUZs07T0RFk5G9AGlyb6A==";
```

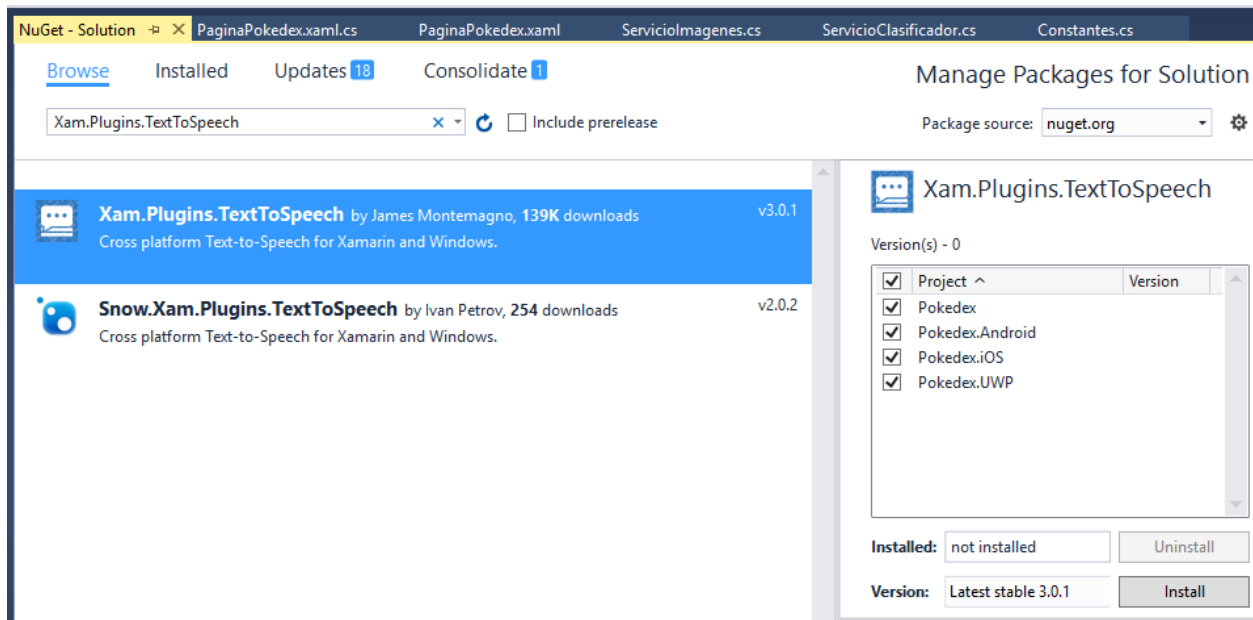
Esta URL es una **Azure Function** en la cual se consulta el contenido de una Google Spreadsheet compartida en Google Drive. El nombre del Pokémon en cuestión se pasa a través del body al hacer el request. **En los anexos (al final de esta práctica), puedes encontrar el contenido de la Azure Function, información importante para configurarlo y el contenido del archivo de Google Drive**

Paso 3. Agrega la clase **ServicioPokedexInfo** dentro de la clase **Servicios**. Su código es el siguiente;

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Pokedex.Helpers;
using System;
using Pokedex.Modelos;
using Newtonsoft.Json;

namespace Pokedex.Servicios
{
    public static class ServicioPokedexInfo
    {
        public async static Task<Pokemon> ObtenerInfo(string nombrePokemon)
        {
            try {
                var url = Constantes.PokedexFunctionURL;
                using (var cliente = new HttpClient()) {
                    string json = "{\\"NombrePokemon\\":\\" + nombrePokemon + "\\"}";
                    HttpContent content = new StringContent(json);
                    content.Headers.ContentType = new
MediaTyHeaderValue("application/json");
                    var post = await cliente.PostAsync(url, content);
                    var resultado = await post.Content.ReadAsStringAsync();
                    var pokemon = JsonConvert.DeserializeObject<Pokemon>(resultado);
                    return pokemon;
                }
            }
            catch (Exception ex) { return null; }
        }
    }
}
```

Paso 4. Agrega el paquete **Xam.Plugins.TextToSpeech** a la solución

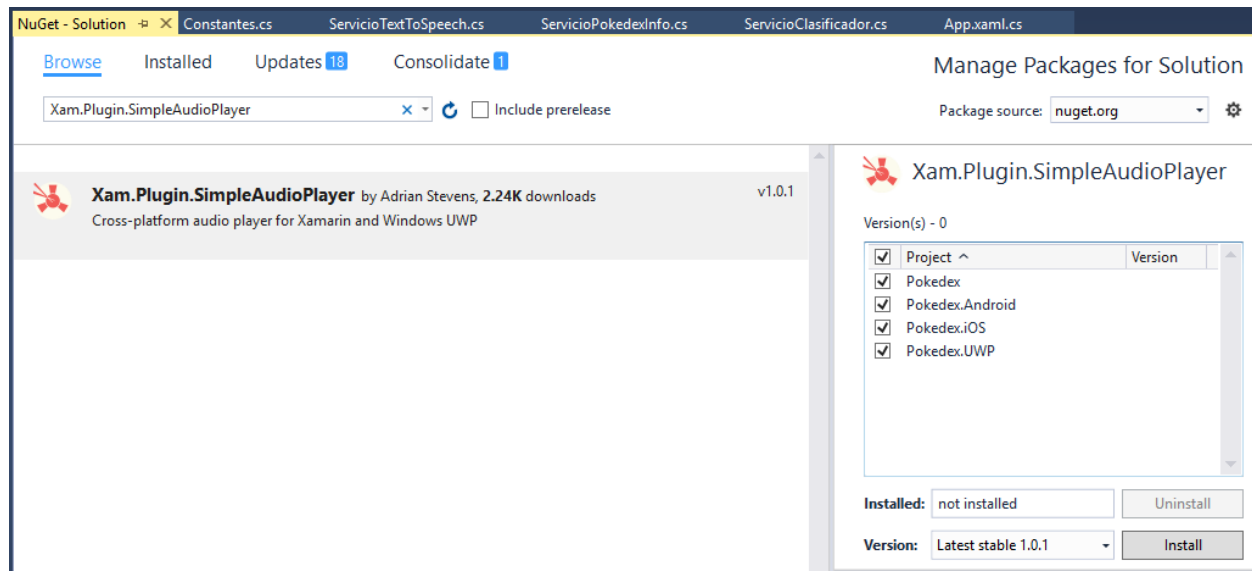


Paso 5. Agrega la clase **ServicioTextToSpeech** en la carpeta de **Servicios**. Su código es:

```
using Plugin.TextToSpeech;
using Plugin.TextToSpeech.Abstractions;
using System.Threading.Tasks;

namespace Pokedex.Servicios
{
    public static class ServicioTextToSpeech
    {
        public static async Task Speak(string texto)
        {
            var locale = new CrossLocale();
            locale.Country = "ES";
            locale.Language = "es";
            await CrossTextToSpeech.Current.Speak(texto, locale);
        }
    }
}
```

Paso 6. Agrega el paquete **Xam.Plugin.SimpleAudioPlayer** a la solución



Paso 7. En este paso vamos a agregar los sonidos característicos de cada Pokémon. Sin embargo, esto se hace de manera específica por cada proyecto/plataforma. Por ello, dependiendo el tipo de proyecto que quieras implementar, crea la carpeta **Pokesonidos** en la ubicación mostrada en la tabla. **Por lo tanto, si tu app estará disponible en las 3 plataformas, hay que crear 3 carpetas.**

Proyecto	Ubicación
Windows/UWP	Dentro de la carpeta Assets
Android	Dentro de la carpeta Assets
iOS, macOS, tvOS	Dentro de la carpeta Resources

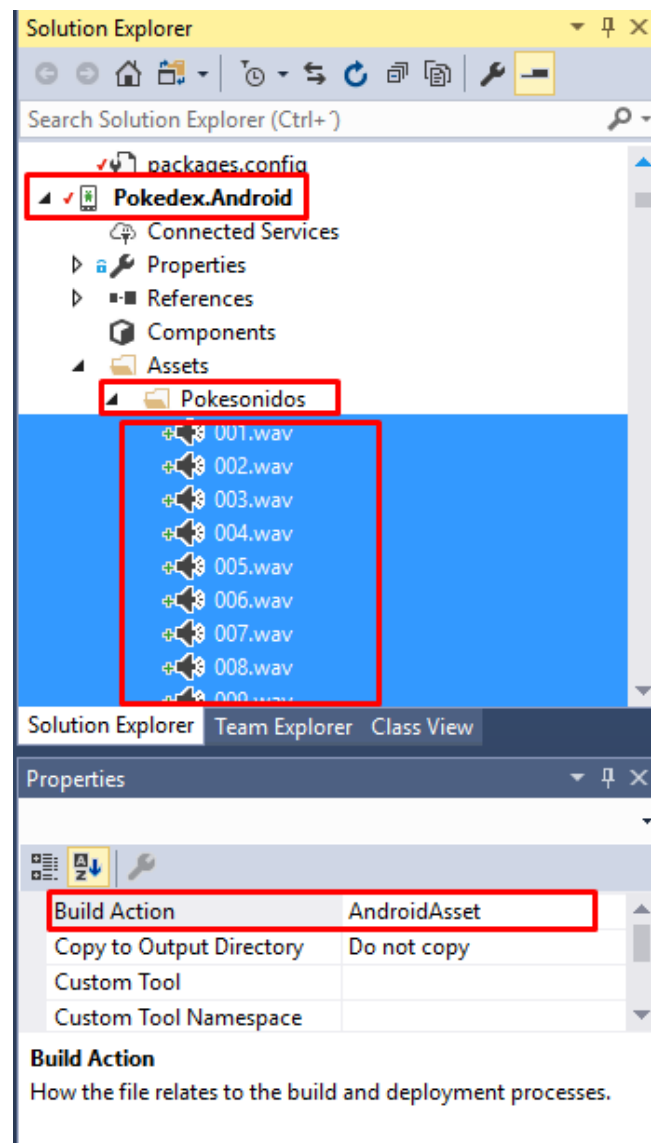
Paso 8. Da clic derecho en la carpeta **Pokesonidos** y elige **Agregar → Elemento existente**. Selecciona todos los audios disponibles en la carpeta **Pokesonidos** incluida en la descarga del proyecto del repositorio. **NOTA.** Al igual que el paso anterior, hay que agregar los audios en cada proyecto que desees implementar.

Paso 9. Selecciona todos los sonidos recién agregados y utilizando el menú de propiedades, establece la acción de compilación (Build Action) dependiendo el tipo de proyecto que estás implementando. **Al igual que los pasos anteriores, este paso se repite por cada plataforma que vayas a probar.**

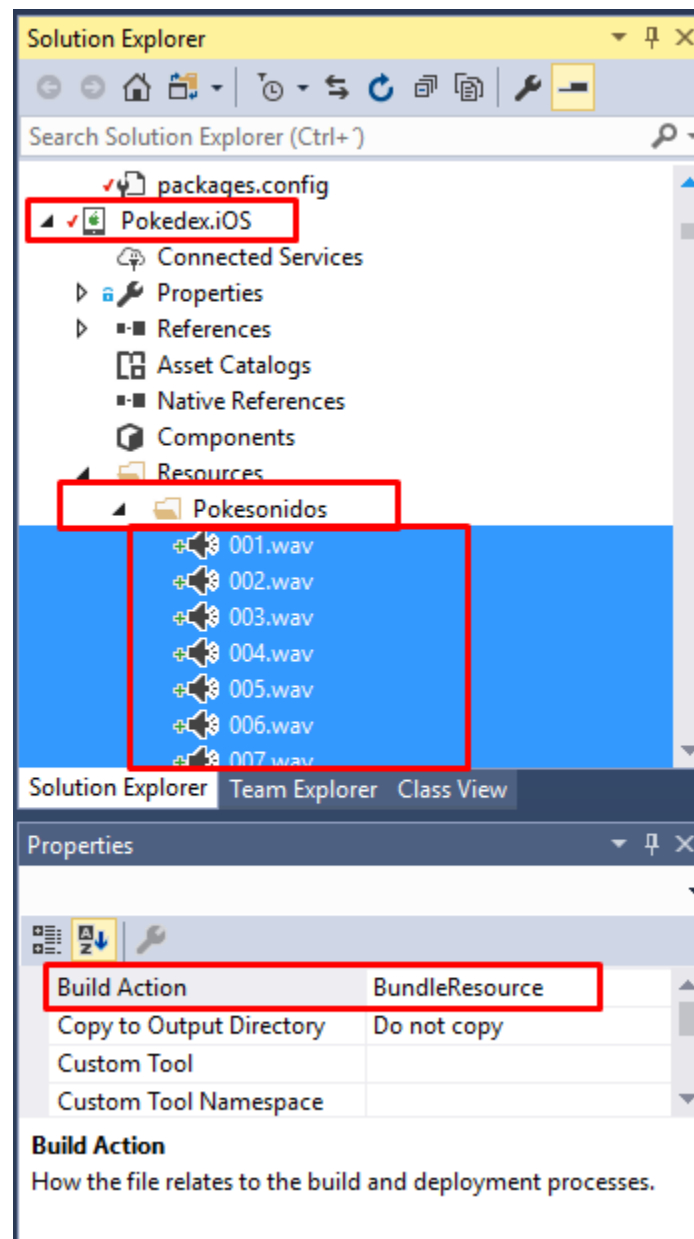
Proyecto	Build Action
Windows/UWP	Content
Android	Android Asset
iOS, macOS, tvOS	BundleResource

A continuación, se muestra el resumen por plataforma de los pasos 7 al 9:

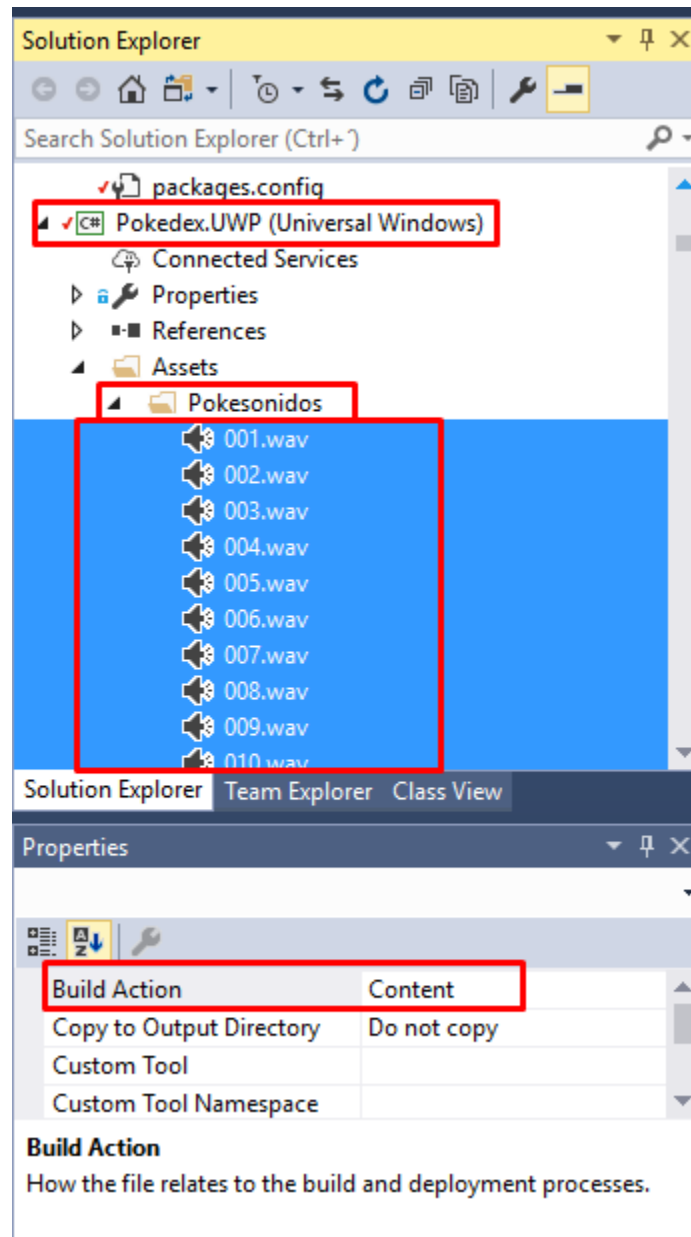
a) Android:



b) iOS:



c) Windows/UWP:



Paso 10. Agrega la clase **ServicioAudio** en la carpeta de **Servicios**. Su código es el siguiente:

```
using Plugin.SimpleAudioPlayer;

namespace Pokedex.Servicios
{
    public static class ServicioAudio
    {
        public static void PlayAudio(string ID)
        {
            var player = CrossSimpleAudioPlayer.Current;
            player.Load($"Pokesonidos/{ID}.wav");
            player.Play();
        }
    }
}
```

Paso 11. Modifica el archivo **PaginaPokedex.xaml**. Simplemente agrega debajo de **lblResultado** el siguiente **Label**:

```
<Label x:Name="lblDetalle" TextColor="White" Margin="5" FontSize="Medium"/>
```

Paso 12. Modifica el archivo **PaginaPokedex.xaml.cs** con el siguiente código:

- a) Primero agrega los dos espacios de nombres mostrados a continuación en la sección superior del código:

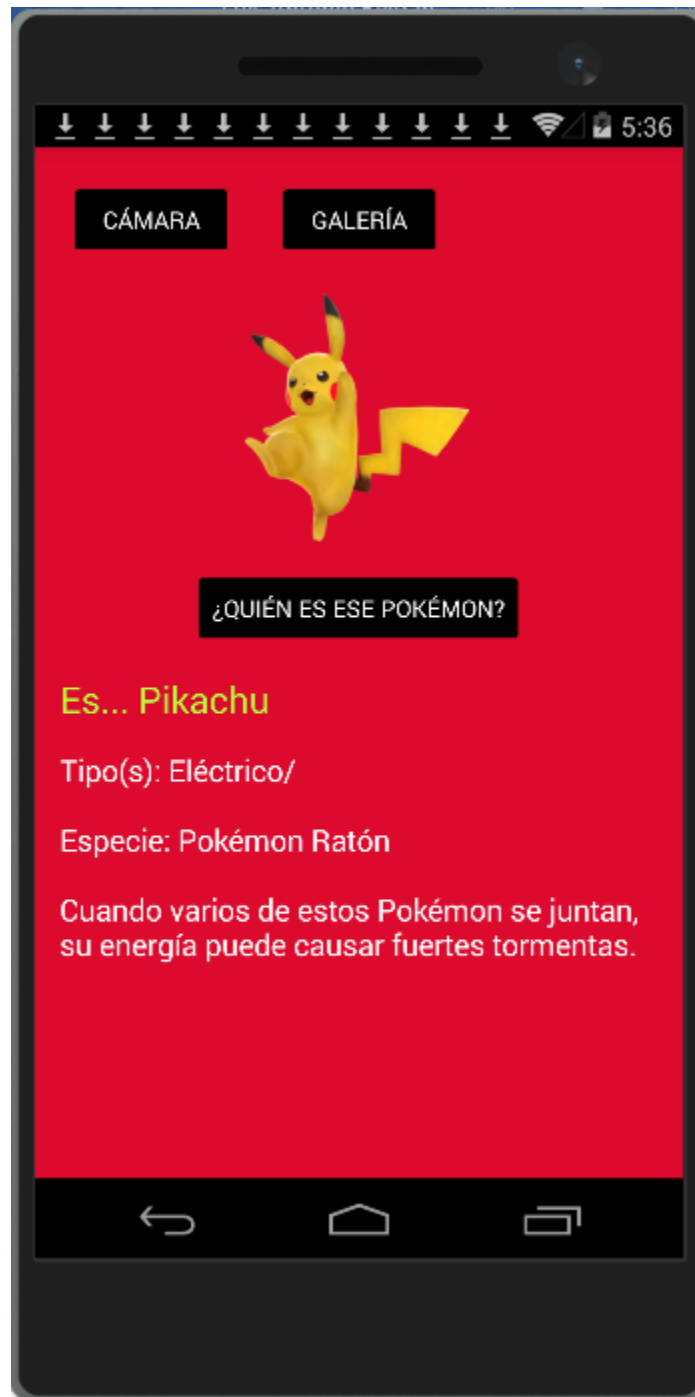
```
using Pokedex.Modelos;
using Pokedex.Helpers;
```

- b) Dentro del método **btnClasificar_Clicked**, agrega el siguiente código debajo de la asignación del texto de **lblResultado** y antes de la instrucción **MostrarIndicador(false)**:

```
if (resultado != Constantes.PokemonNoIdentificado)
{
    Pokemon pokemon = await ServicioPokedexInfo.ObtenerInfo(resultado);

    lblDetalle.Text = $"Tipo(s): {pokemon.Tipo1}/{pokemon.Tipo2}\n\nEspecie:
{pokemon.Especie}\n\n{pokemon.Descripcion}";
    ServicioAudio.PlayAudio(pokemon.ID);
    await ServicioTextToSpeech.Speak($"{pokemon.Nombre}, {pokemon.Especie},
{pokemon.Descripcion}");
}
```

Paso 13. Compila y ejecuta la aplicación nuevamente:



También se debe escuchar el sonido del Pokémon, así como una voz que menciona el nombre, especie y descripción del Pokémon.

Anexo 1. Archivo de Google Drive (información del Pokédex)

Pokedex Information - G X

Es seguro | <https://docs.google.com/spreadsheets/d/14Clz2CCKDa6DLUpSqFK5V1B6SCXLgE58cpYg31gxiTc/edit#gid=0>

Pokedex Information ☆

File Edit View Insert Format Data Tools Add-ons Help Last edit was 4 hours ago

100% \$ % .0 .00 123 Arial 10 B I S A

ID	ID	A	B	C	D	E	F
1	ID	Nombre	Tipo1	Tipo2	Especie	Descripcion	
2	001	Bulbasaur	Planta	Veneno	Pokémon Semilla	Una rara semilla fue plantada en su espalda al nacer. La planta brota y crece con este Pokémon.	
3	003	Venasaur	Planta	Veneno	Pokémon Semilla	La planta florece cuando absorbe energía solar. Ésta le obliga a ponerse en busca de la luz solar.	
4	004	Charmander	Fuego		Pokémon Lagartija	Prefiere los sitios calientes. Dicen que cuando llueve sale vapor de la punta de su cola.	
5	006	Charizard	Fuego	Volador	Pokémon Llama	Escupe fuego tan caliente que funde las rocas. Causa incendios forestales sin querer.	
6	007	Squirtle	Agua		Pokémon Tortuguila	Tras nacer, su espalda se hincha y endurece como una concha. Echa potente espuma por la boca.	
7	009	Blastoise	Agua		Pokémon Marisco	Un brutal Pokémon con reactores de agua en su caparazón. Éstos son usados para rápidos placajes.	
8	025	Pikachu	Eléctrico		Pokémon Ratón	Cuando varios de estos Pokémon se juntan, su energía puede causar fuertes tormentas.	
9	144	Articuno	Hielo	Volador	Pokémon Congelar	Un legendario pájaro Pokémon. Se aparece a la gente que se ha perdido en las heladas montañas.	
10	172	Pichu	Eléctrico		Pokémon Ratoncito	Todavía no domina el almacenamiento de electricidad, y descargará un rayo si se divierte o asusta.	
11	384	Rayquaza	Dragón	Volador	Pokémon Cielo	Rayquaza vivió durante cientos de millones de años en la capa de ozono sin bajar a la tierra ni una vez. Si	
12							

+ Pokemon

Share with others Get shareable link

Link sharing on [Learn more](#)

Anyone with the link can view Copy link

<https://docs.google.com/spreadsheets/d/14Clz2CCKDa6DLUpSqFK5V1B6SCXLgE58cpYg31gxiTc/edit#gid=0>

People

Enter names or email addresses...

Done Advanced

NOTAS:

- El **ID del archivo** (el que viene después de d/ en la URL) lo vas a utilizar en la Azure Function
- El **primer renglón** debe tener el **nombre de las columnas** (requerido por Azure Functions)
- Necesitas colocar un **nombre a la hoja** (en este caso es Pokemon, requerido por Azure Functions). **El nombre del archivo no puede coincidir con el nombre de la hoja.**
- En la Azure Function vas a crear un conector a Google Drive utilizando tu cuenta de Gmail.
 - Si el **archivo de Google Drive** está en la misma cuenta, solo necesitas permisos de **lectura** sobre el archivo.
 - Si el **archivo de Google Drive** está en otra cuenta, requieres contar con permisos de **escritura** sobre el archivo.

Anexo 2. Azure Function

a) Creación y configuración de la Azure Function

Choose a template below or [go to the quickstart](#)

Language: All ▼

Scenario: Experimental ▼

ExternalFileTrigger - Batch (Experimental) A Batch function that will be run whenever a file is added to a External File provider.	ExternalFileTrigger - C# (Preview) A C# function that will be run whenever a file is added to a External File provider.	ExternalFileTrigger - F# (Preview) An F# function that will be run whenever a file is added to a External File provider.	ExternalFileTrigger - JavaScript (Preview) A JavaScript function that will be run whenever a file is added to a External File provider.
ExternalTable - C# (Experimental) A C# function that fetches entities from a External Table when it receives an HTTP request.	ExternalTable - F# (Experimental) An F# function that fetches entities from a External Table when it receives an HTTP request.	EventGridTrigger - C# A C# function that will be run whenever an event grid receives a new event	EventGridTrigger - JavaScript A JavaScript function that will be run whenever an event grid receives a new event

This template is experimental and does not yet have full support. If you run into issues, please file a bug on our [GitHub repository](#).

Name your function

pokedex_function

External Table (Experimental) input

Table Name ⓘ

Pokemon

External Table connection ⓘ

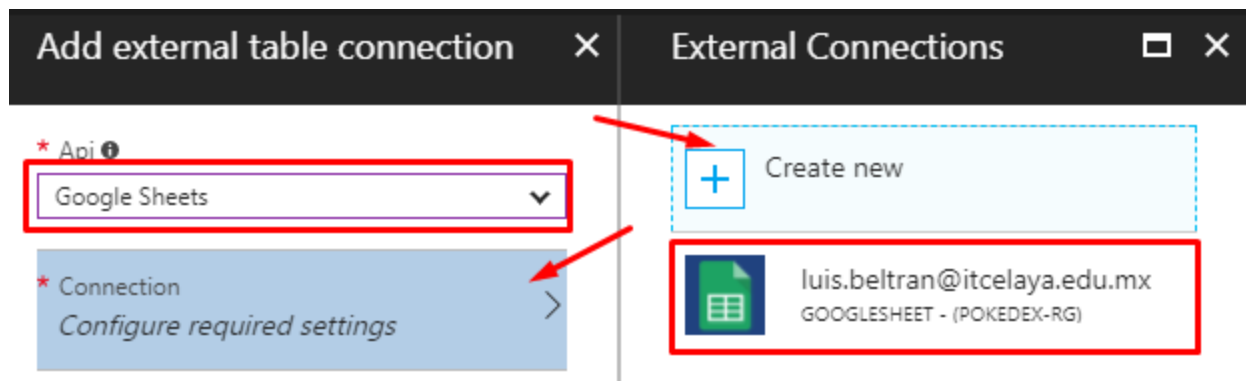
[show value](#)

googlesheet_GOOGLESHEET

[new](#)

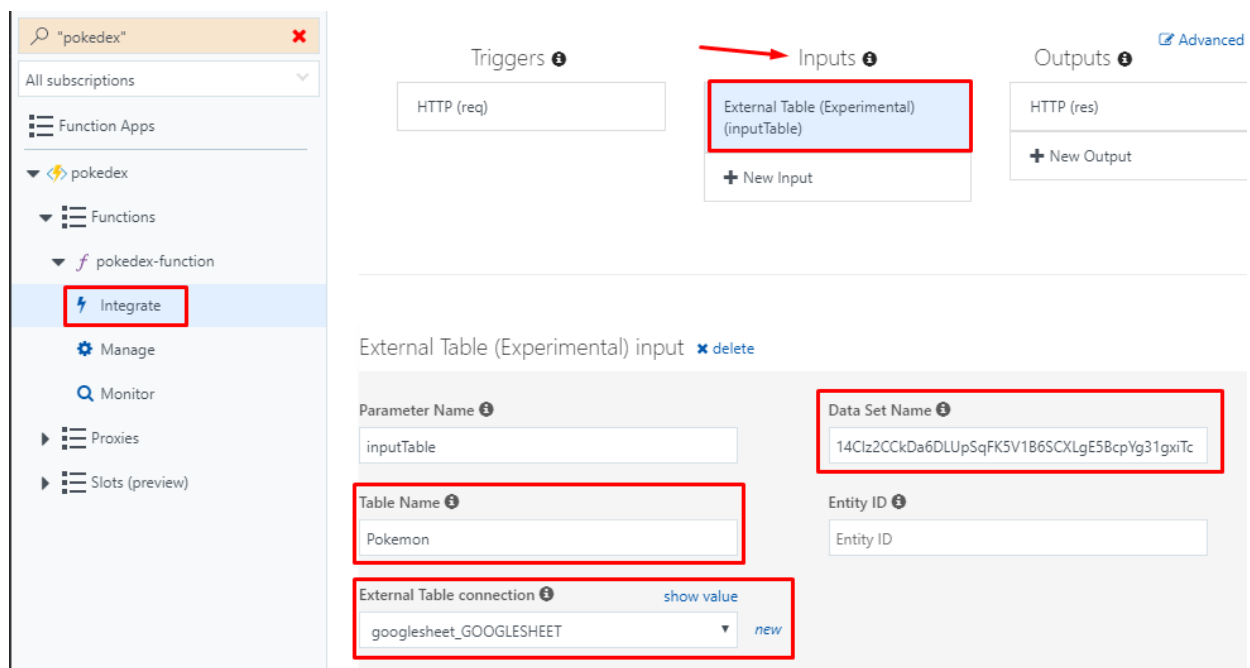
NOTAS:

- El template utilizado es **ExternalTable C#**, el cual está dentro de los escenarios de tipo **Experimental**.
- **TableName** corresponde al **nombre de la hoja** en el archivo de GoogleDrive (Pokemon)
- La conexión a tabla externa es de tipo **Google Sheets**. Para crearlo, se da clic en **new**.



NOTAS:

- Elige el Api **Google Sheets**.
- En la conexión, configura los parámetros requeridos, dando clic en Create new.
- Posteriormente, se solicitarán tus credenciales de acceso a Google Drive.



NOTAS:

- En la sección **Integrate**, configura la **External Table** que aparece en **Inputs**.
- El **Data Set Name** es el **identificador del archivo de Google Drive**.
- El resto de parámetros ya está configurado (Table Name y Connection).

b) Código de la Azure Function:

```
#r "Microsoft.Azure.ApiHub.Sdk"
#r "Newtonsoft.Json"
using System;
using System.Net;
using Microsoft.Azure.ApiHub;
using System.Text;
using Newtonsoft.Json;

public class Pokemon
{
    public string ID { get; set; }
    public string Nombre { get; set; }
    public string Tipo1 { get; set; }
    public string Tipo2 { get; set; }
    public string Especie { get; set; }
    public string Descripcion { get; set; }
}

public class Consulta
{
    public string NombrePokemon { get; set; }
}

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
ITable<Pokemon> inputTable, TraceWriter log)
{
    var datos = await req.Content.ReadAsAsync<Consulta>();
    var pokemon = String.Empty;
    var pokemonEncontrado = false;
    ContinuationToken token = null;

    do {
        var pagina = await inputTable.ListEntitiesAsync(continuationToken:
token);
        var pokemons = pagina.Items.ToList();
        var busqueda = pokemons.Where(x => x.Nombre ==
datos.NombrePokemon).FirstOrDefault();

        if (busqueda != null) {
            pokemon = JsonConvert.SerializeObject(busqueda, Formatting.Indented);
            pokemonEncontrado = true;
        }
        token = pagina.ContinuationToken;
    } while (token != null && !pokemonEncontrado);

    return new HttpResponseMessage(HttpStatusCode.OK) {
        Content = new StringContent(pokemon, Encoding.UTF8, "application/json")
    };
}
```

c) Prueba de la Azure Function

The screenshot displays the 'Test' tab of the Azure Functions portal. The 'HTTP method' is set to 'POST'. The 'Request body' is a JSON object: `{ "NombrePokemon": "Pichu" }`. The 'Output' section shows a successful response with status '200 OK' and a JSON body: `{ "ID": "172", "Nombre": "Pichu", "Tipo1": "Eléctrico", "Tipo2": "", "Especie": "Pokémon Ratoncito", "Descripcion": "Todavía no domina el almacenamiento de" }`. A red arrow points from the 'Nombre' field in the output to the 'NombrePokemon' field in the request body. The 'Run' button is highlighted with a red box.

View files Test

HTTP method
POST

Query
There are no query parameters
[+ Add parameter](#)

Headers
There are no headers
[+ Add header](#)

Request body

```
1 {  
2   "NombrePokemon": "Pichu"  
3 }  
4
```

Output ✔ Status: 200 OK

```
{  
  "ID": "172",  
  "Nombre": "Pichu",  
  "Tipo1": "Eléctrico",  
  "Tipo2": "",  
  "Especie": "Pokémon Ratoncito",  
  "Descripcion": "Todavía no domina el almacenamiento de"  
}
```

Run

NOTAS:

- El **método HTTP** de la petición es **POST**.
- En el **Request Body**, se coloca en formato **Json** un objeto anónimo que tiene una propiedad **NombrePokemon**, seguida del valor del Pokémon que se desea consultar.
- Al dar clic en el botón **Run**, aparecerá el resultado en la sección **Output**, mostrando un **objeto Pokemon** en formato **Json** que contiene la información obtenida del **archivo de Google Drive**.

d) Obtener la dirección URL de la Azure Function para su consumo

