**Ejercicio #1 – Utilizar API de cámara**

1. En carpeta nueva, **CloudOperations**, crear tres pantallas: **CloudMainPage**, **ImageUploadPage** e **ImageDownloadPage**.
   1. **CloudMainPage** será **TabbedPage**; las dos pantallas serán hijas.
   2. Agregar **CloudMainPage** a la master**.**
2. En los tres proyectos, descargar el paquete **Nuget** **Xam.Plugin.Media**.
3. Seguir las instrucciones de configuración del archivo **readme**:
   1. En **MainActivity** de Android:
      1. Utilizar método de **Plugin.Permissions** en el evento **OnRequestPermissionsResult** (con las condiciones de los tres permisos)**.**
      2. Agregar en **OnCreate** el método **CrossCurrentActivity.Current.Init**; agregar la librería …
   2. En **AndroidManifest.xml** (ya se agregaron permisos de read y write external storage \*ver el archivo oculto\*) agregar la etiqueta **provider** dentro de **application**; agregar también el permiso de la cámara (desde ventana gráfica preferentemente).
   3. En el proyecto de Android, agregar un folder llamado **xml** en la carpeta **Resources** y crear el archivo **file\_paths.xml** con el código proporcionado …
   4. Agregar las cuatro llaves en el archivo **Info.plist** de iOS; abrir el archivo con XML …
4. Crear la UI de **ImageUploadPage**: una imagen, un label, dos botones, un ActivityIndicator y un label. Irán dentro de un **StackLayout**, dentro de un **ScrollView**.
5. En el codebehind de **ImageUploadPage**, agregar las librerías necesarias (de **Media** y **Permissions**) y:
   1. Crear variable global **imageFile** (tipo **MediaFile**).
   2. En el constructor, asignar un handler al evento **Clicked** del botón para tomar la foto. En el handler:
      1. Revisar que la cámara esté disponible o que haya cámara …
      2. Revisar que se tengan los permisos para la cámara y almacenamiento …
         1. Si se tienen, tomar la foto y guardarla (en carpeta privada y pública); mostrar su ubicación en el label y mostrarla en la imagen; habilitar el botón de carga …
         2. Si no, mostrar mensaje adecuado …
6. \*Poner breakpoints en el handler y en **MainActivity**, para ver los permisos\*.
7. Correr la app, modificando valores de **StoreCameraMediaOptions**, para el almacenamiento privado y público (se toma la foto, y se muestra).
8. Continuar con la presentación.

**Ejercicio #2 – Subir imagen a la nube (Azure)**

1. Asignar un handler al evento **Clicked** del botón de carga.
2. En el handler, determinar si hay acceso a Internet; si no hay conexión, mostrar una alerta. Usar clase **Connectivity** de **Xamarin.Essentials**.
3. Crear el método **GetByteArray**, para generar un arreglo de bytes a partir de un archivo **MediaFile**: se recorre el archivo y se llena un arreglo de bytes con su información.
4. En el evento del botón de carga, crear un arreglo de bytes a partir de la foto; mostrar al indicador de actividad corriendo.
5. Crear la clase estática **AzureStorage**:
   1. Instalar en el proyecto (NET Standard) el paquete **Nuget** **WindowsAzure.Storage**; agregar las referencias using necesarias.
   2. Crear la conexión a una cuenta de Azure (mostar mi cuenta (dashboard) de Azure).
   3. Crear el método **GetContainer**: devuelve un contenedor de archivos blob con un nombre específico.
   4. Crear el método **UploadFileAsync**: llama a **GetContainer** y obtiene un container o lo crea si no existe; crea un archivo blob en el contenedor y le proporciona la información correspondiente.
6. En el evento de botón de carga, llamar al método **UploadFileAsync** de **AzureStorage**, proporcionando al arreglo de bytes como un stream de datos (con clase **MemoryStream**).
7. Detener al indicador de actividad y mostrar el texto de éxito en el mensaje de carga.
8. Correr la app(se sube la foto a la nube).
9. Continuar al siguiente ejercicio.

**Ejercicio #3 – Descargar imagen de la nube (Azure)**

1. Crear la UI de **ImageDownloadPage**: un label, un Entry, un botón, una imagen, un botón, un ActivityIndicator y un label. Irán dentro de un **StackLayout**, dentro de un **ScrollView**.
2. En **AzureStorage**:
   1. Crear el método **DownloadFileAsync**: llama a **GetContainer** y , a partir de un parámetro, obtiene un blob de ese container. Si el blob existe, el método devuelve un arreglo de bytes con la información del blob, si no existe se devuelve un null.
3. En codebehind de **ImageDownloadPage**:
   1. En constructor, asignar un handler al evento **Clicked** del botón para mostrar la imagen. En el handler:
      1. Validar que el valor del control **Entry** sea correcto.
      2. Mostrar un indicador de actividad corriendo.
      3. Guardar en una variable global el resultado del método **DownloadFileAsync** de **AzureStorage**, proporcionando el nombre de la imagen (valor de **Entry**).
      4. Proporcionar la variable como la fuente de datos del control **Image**.
      5. Detener al indicador de actividad.
      6. Habilitar el botón para descargar la imagen.
4. Correr la app (se muestra la imagen de la nube).
5. \*Ahora se guardará la imagen en el dispositivo\*
6. En codebehind de **ImageDownloadPage**:
   1. En constructor, asignar un handler al evento **Clicked** del botón para descargar la imagen. En el handler:
      1. Obtener carpeta Pictures en almacenamiento interno.
      2. Asignar una ruta a esa carpeta con el nombre de la imagen.
      3. Crear la imagen en el dispositivo con método WriteAllBytes de File.
      4. Mostrar la ruta de la imagen.
7. Correr la app (se guarda la imagen en el dispositivo ¿?).