**Ejercicio #1 – Determinar la conectividad**

1. En carpeta nueva **Books**, crear pantalla **BooksListPage**:
   1. En XAML: Crear dentro de **StackLayout**, un botón (que mostrará una lista de libros); dar título a la pantalla.
   2. En codebehind:
      1. En el constructor, generar handler de evento **Clicked** del botón.
      2. Incluir **using Xamarin.Essentials** (En Android, se necesita el permiso **ACCESS\_NETWORK\_STATE** \*ya lo tenemos\*).
      3. En el handler, mostrar una alerta si no hay acceso a una red (**NetworkAccess.None**).
      4. No hacer nada si hay acceso a una red.
2. Incluir la pantalla en la master.
3. Correr la app (si no hay Internet, se muestra una alerta).
4. Continuar con la presentación.

**Ejercicio #2 – Probar un servicio web REST**

1. En **Sandbox** de **Azure**, crear app web temporal, con un servicio web de ejemplo (Books – link: <https://docs.microsoft.com/en-us/learn/modules/consume-rest-services/4-exercise-consume-rest-service-with-httpclient>).
2. Buscar e instalar aplicación **Postman** (iniciar sesión con Google).
3. Abrir aplicación y probar el servicio web con la URI de la app web temporal, con los métodos:
   1. **GET**: Agregar palabra **login** a la URI; se genera una llave que autoriza el uso del servicio.
   2. **GET**: Sin palabra **login**, y agregando dos requests headers: **Authorization** con el valor de la llave y **Accept** con valor de application/json (esos headers también los usan los demás verbos).
   3. **POST**: Incluyendo el cuerpo del mensaje en pestaña **Body**; en formato **JSON**, un objeto con atributo **isbn** vacío y sin atributo **href**.
   4. **PUT**: Agregar valor de **isbn** en la URI e incluir el cuerpo del mensaje en pestaña **Body**; en formato **JSON**, un objeto Book.
   5. **DELETE**: Agregar valor de isbn en la URI.
4. Continuar con la presentación.

**Ejercicio #3 – Consumir un servicio web REST en la aplicación**

1. Incluir clases **Book** y **BookManager** de carpeta Recursos. Examinarlas.
2. \* Para consumir el servicio web se requiere un token\*.
3. En **BookManager**:
   1. Agregar la constante **Url** del servicio …
   2. Agregar el atributo **authorizationKey** …
   3. Crear el método privado **GetClient** (incluir **using System.Net.Http** y **using Newtonsoft.Json**)…
      1. Si es la primera vez que se utiliza la clase, se debe obtener el token y asignarlo a **authorizationKey** …
      2. Agregar los headers al objeto de **HttpClient** …
      3. Devolver el objeto **HttpClient**.
4. \*Ahora se realizará la operación **GET**, para llenar una lista en **BooksListPage**\*.
5. En **BookManager**:
   1. En el método **GetAll**:
      1. Llamar a **GetClient** para crear un **HttpClient** …
      2. Usar **GetStringAsync** para obtener un arreglo de libros …
      3. Convertir el objeto JSON en un arreglo **Book**.
6. En **BooksListPage**:
   1. En codebehind:
      1. Crear variable global **manager (BookManager)**.
      2. En constructor, instanciar **manager**.
      3. En handler del botón, hacer que **BindingContext** sea método **GetAll** de objeto **manager**.
   2. En XAML:
      1. Crear **ListView** con un nombre, con **ItemsSource** con valor de **Binding**.
      2. Dentro de **TextCell**, mostrar los atributos Title (en **Text**) y Genre (en **Detail**) de cada elemento.
7. Correr la app (se muestra a lista de libros al dar clic al botón).
8. \*Ahora se realizará la operación **POST**, para crear un objeto **Book**\*.
9. En **BookManager**:
   1. En el método **Add**:
      1. Crear un objeto **Book** con los datos de los parámetros …
      2. Llamar a **GetClient** para crear un **HttpClient** …
      3. Usar **PostAsync** para agregar un libro …
10. Crear nueva pantalla: **AddBookPage**:
    1. En XAML:
       1. Dentro de **StackLayout**:
          1. Crear control **TableView** de tipo **Form**. Crear etiquetas **TableRoot** y **TableSection** (con título).
             1. Crear tres controles **EntryCell** en **TableSection** para los tres atributos de objeto **Book**.
          2. Crear botón (con nombre).
       2. Dar título a pantalla.
    2. En codebehind:
       1. Agregar objeto **BookManager** como parámetro de constructor.
       2. Crear variable global **manager** (tipo **BookManager**).
       3. En constructor, asignar parámetro como valor de **manager**; y crear handler de evento **Clicked** del botón.
       4. En handler:
          1. Inhabilitar botón y hacer verdadero a **IsBusy**.
          2. Mostrar alerta si algún control **EntryCell** está vacío o nulo; si todos tienen valores, llamar método **Add** de manager y cerrar pantalla con método **PopModalAsync**.
          3. Habilitar botón y hacer false a **IsBusy**.
11. Modificar **BooksListPage**:
    1. En XAML:
       1. Agregar elemento en **ToolbarItems**: un botón para agregar objetos a la lista.
    2. En codebehind:
       1. En constructor, generar handler de evento **Clicked** del botón en **ToolbarItems**.
       2. En el handler, mostrar la pantalla **AddBookPage** con método **PushModalAsync**.
12. Correr la app (se agrega un libro).
13. \*Ahora se realizará la operación **PUT**, para actualizar un objeto **Book**\*.
14. En **BookManager**:
    1. En el método **Update**:
       1. Llamar a **GetClient** para crear un **HttpClient** …
       2. Usar **PutAsync** para actualizar un libro. La Url tendrá el ISBN del libro a modificar …
15. Crear nueva pantalla: **EditBookPage**:
    1. En XAML:
       1. Mismo diseño (**TableView**) que pantalla **AddBookPage**; quitar placeholders de controles **EntryCell**.
       2. Agregar atributos de **Binding** correspondientes para cada **EntryCell** (escribir **Authors[0]** como valor de **Binding** para Author).
       3. Cambiar info de botón.
       4. Dar título a la pantalla.
    2. En codebehind:
       1. Agregar objetos **BookManager** y **Book** como parámetros de constructor.
       2. Crear variable global **manager** (tipo **BookManager**).
       3. En constructor, asignar parámetro **BookManager** como valor de **manager**; hacer que parámetro **Book** sea el **BindingContext** y crear handler de evento **Clicked** del botón.
       4. En handler:
          1. \*Misma funcionalidad que handler de pantalla **AddBookPage**\*.
             1. Llamar a método **Update** de **BookManager**, enviando a objeto **BindingContext** como parámetro y cerrar la pantalla.
16. Modificar **BooksListPage**:
    1. En XAML:
       1. \*Sin cambio\*.
    2. En codebehind:
       1. En constructor, crear handler para evento **ItemTapped** de la lista.
       2. En el handler, mostrar la pantalla **EditBookPage** con método **PushModalAsync**.
17. Correr la app (se actualiza un libro).
18. \*Ahora se realizará la operación **DELETE** para eliminar un objeto Book.\*
19. En **BookManager**:
    1. En el método **Delete**:
       1. Llamar a **GetClient** para crear un **HttpClient** …
       2. Usar **DeleteAsync** para eliminar un libro. La Url tendrá el ISBN del libro a eliminar …
20. Modificar **BooksListPage**:
    1. En XAML:
       1. Dentro de nueva etiqueta **TextCell.ContextActions** agregar elemento **MenuItem**, con atributos **Name**, **Text**, **Clicked** (se genera un handler) y **CommandParameter**.
    2. En codebehind:
       1. En el handler generado, crear objeto **Book**; en una alerta, corroborar que se desea eliminar el objeto, y llamar a método **Delete** de manager.
21. Correr la app (se elimina un libro, pero no se refleja en la lista; hay que dar clic en botón show/refresh).
22. Modificar codebehind de **BooksListPage**:
    1. Crear nuevo método (**LoadListData**), pegar el contenido de handler del botón para cargar la lista.
    2. Llamar a método **LoadListData** en handler de botón de carga y en handler de botón de eliminar (después de la llamada al método **Delete**).
23. Correr la app (se elimina un libro y se refleja automáticamente).