**Ejercicio #1 – Mostrar información de archivo json**

1. Instalar el paquete **Nuget Newtonsoft.Json** (sólo en proyecto **.NET Standard**).
2. Definir los recursos para pantalla nueva:
   1. Crear en la raíz dos carpetas: **Clients** e **Images**.
   2. De la carpeta **Recursos**, copiar las imágenes de logos en **Images** y los archivos **clients.json**, **Client.cs**, **ClientRepository.cs y ClientExtensions.cs** en **Clients**.
      1. Hacer que la propiedad **BuildAction** de las imágenes y el archivo json sea **EmbeddedResource**.
   3. Comentar cada archivo de carpeta **Clients**, en especial **ClientRepository** (genera una lista de objetos **Client**, a partir del archivo **json**).
3. En carpeta **Clients**, crear pantalla **ClientsPage**:
   1. En XAML:
      1. Dar un título a la pantalla.
      2. En etiqueta **ToolbarItems**, agregar dos elementos (botones **Previous** y **Next**).
      3. Dentro de un **ScrollView**, un **StackLayout** (con **Spacing** y **Padding**).
      4. Dentro de Stack, agregar controles requeridos (asignar **Style** correspondiente a los labels).
         1. Sin las propiedades de **Binding**.
   2. En codebehind:
      1. Crear variables globales **repository** y **currentClient**.
      2. En constructor, inicializar variables globales.
      3. Crear atributo **CurrentClient** (con **get** de objeto en lista **Clients** de **repository**).
      4. Crear método **InitializeData**: asignar valores de **CurrentClient** a cada control.
      5. En constructor, llamar a método **InitializeData** y generar handlers de botones de **Toolbar**.
      6. Dar el funcionamiento correcto a los handlers de botones (disminuir/aumentar valor de **currentClient** y llamar a **InitializeData**).
4. Agregar pantalla a la master page.
5. Correr la app (se muestra la información de los clientes).
6. Continuar al siguiente ejercicio.

**Ejercicio #2 - Reemplazar C# con bindings**

1. En codebehind de **ClientsPage**:
   1. Quitar (comentar) el contenido de **InitializeData**, excepto la línea que asigna el valor de la imagen (con el método de **ClientExtensions**).
   2. Asignar el valor del cliente actual (**CurrentClient**) a la propiedad **BindingContext**.
2. En XAML de **ClientsPage**:
   1. Asignar el valor de **Text** de los labels y el valor de **Date** de DatePicker al atributo adecuado del objeto **Binding**.
3. Correr la app (mismo funcionamiento).
4. Continuar con la presentación.

**Ejercicio #3 – Generar evento para actualizar objeto target**

1. Agregar nuevo botón en Toolbar, para mostrar una alerta con info:
   1. En XAML, agregar nuevo elemento a etiqueta **ToolbarItems** (botón **Show**).
   2. En codebehind, generar evento del botón en el constructor; en el evento, mostrar alerta (**DisplayAlert**) con info del nombre y fecha del cliente.
2. Correr la app (alerta muestra info).
3. \*Simular actualización del modelo de datos en tiempo de ejecución:\*
   1. En el evento de la alerta, modificar la fecha del objeto **CurrentClient**.
4. Correr la app (el evento actualiza la fecha, pero no se refleja en automático en el control **Datepicker**; si se cambia la pantalla sí se actualiza, pero es porque **InitializeData** vuelve a asignar la propiedad **BindingContext**).
5. Llamar evento en setter de **CompletionDate**; en clase **Client**:
   1. Incluir **using System.ComponentModel**.
   2. Implementar interfaz **INotifyPropertyChanged** (se implementa un atributo tipo evento).
   3. Crear variable global **completionDate**:
      1. En getter, devolver el valor de **completionDate**.
      2. En setter, revisar que **completionDate** sea diferente a **value**, si lo es, asignar a **value** e invocar el handler del evento **PropertyChanged**.
6. Correr la app (el evento de la alerta modifica la fecha y se refleja en automático en el control **Datepicker**).
7. Continuar con la presentación.

**Ejercicio #4 – Crear un binding entre tipos de datos distintos**

1. \*El control **Image** necesita un objeto **ImageSource**, atributo ImageUrl de Client es **string**\*.
2. En carpeta **Clients**, crear clase **ImageConverter**:
   1. Incluir **using Xamarin.Forms**.
   2. Implementar interfaz **IValueConverter** (dos métodos).
   3. En método **Convert**, cerciorarse de que parámetro **value** no sea null ni cadena vacía; devolver objeto **ImageSource** (misma línea que en **ClientExtensions**).
   4. En método **ConvertBack**, arrojar una excepción **NotSupportedException**.
3. Utilizar el converter en el binding, en XAML de **ClientsPage**:
   1. Definir prefijo (cvt) del **namespace** donde está la clase **ImageConverter**.
   2. Crear una instancia del converter (**ImageConverter**) en un diccionario de recursos.
   3. Asignar atributo **Source** a control **Image**, con dos parámetros: ImageUrl como **Binding** y la instancia del converter (con **StaticResource**) como **Converter**.
4. En codebehind, comentar la línea de **InitializeData** que asigna **Source** a la imagen.
5. Correr la aplicación (mismo funcionamiento).
6. Continuar con la presentación.

**Ejercicio #5 – Mostrar una colección de datos con ListView**

1. Crear pantalla **ClientsListPage**:
   1. En codebehind:
      1. Crear una variable **repository** (de tipo **ClientRepository**).
      2. En constructor, instanciar **repository**.
      3. Crear atributo **Clients**, que devuelva la lista de clientes de **repository**.
      4. En constructor, hacer que el **BindingContext** de la pantalla sea el atributo **Clients**.
   2. En XAML:
      1. Crear controles **Label** y **ListView** dentro de **StackLayout**.
      2. Asignar un texto, estilo y margen al **Label**.
      3. Al **ListView**, asignar un nombre y asignar el **BindingContext** al atributo **ItemsSource** (sólo se escribe el atributo **Binding**).
      4. Dar un título a la pantalla.
2. Modificar botón Clients de **master** page para que ahora muestre la pantalla **ClientsListPage**.
3. Correr la app (se muestra la lista, pero sin información relevante).
4. Modificar la clase **Client**, para sobre escribir el método **ToString**; que sólo muestre el atributo **Name**.
5. Correr la app (se muestra la lista, con el nombre de los clientes).
6. Continuar al siguiente ejercicio.

**Ejercicio #6 – Mostrar una pantalla al seleccionar elemento de ListView**

1. \*Ahora **ClientsPage** se mostrará con la información de un elemento de la lista\*.
2. Modificar **ClientsPage**:
   1. En codebehind:
      1. Comentar todo el contenido: las variables **repository** y **currentClient**, el atributo **CurrentClient**, el método **InitializeData** y los tres eventos.
      2. Incluir un parámetro tipo **Client** en el constructor, y asignarlo al **BindingContext**.
   2. En XAML:
      1. Comentar la etiqueta de **ToolbarItems**.
      2. \*Los atributos de **Binding** se quedan igual en todos los controles\*.
3. Modificar **ClientsListPage**:
   1. En codebehind:
      1. En constructor, generar handler de evento **ItemTapped** del control **ListView**.
      2. En handler, crear un objeto **Client** del parámetro **ItemTappedEventArgs** del handler; mostrar la pantalla **ClientsPage** enviando al objeto **Client** como parámetro (con **PushAsync**).
   2. En XAML:
      1. \*Sin cambios\*.
4. Correr la app (se muestra una pantalla con la información del cliente seleccionado en la lista).
5. Continuar al siguiente ejercicio.

**Ejercicio #7 – Personalizar apariencia de ListView**

1. Crear una **ImageCell** dentro de etiqueta **DataTemplate** dentro de etiqueta **ListView.ItemTemplate**.
   1. Dar valores de **Binding** a los atributos correspondientes.
   2. Utilizar el converter para **ImageSource**:
      1. Mover el recurso **ImageConverter** de **ClientsPage.xaml** a **App.xaml**.
2. Correr la app (no se muestra la imagen en las celdas de ListView).
3. \*Emular la apariencia de **ImageCell** con **ViewCell**\*:
   1. Crear una **ViewCell** dentro de etiqueta **DataTemplate** dentro de etiqueta **ListView.ItemTemplate**.
   2. Crear un **Grid** con dos filas y dos columnas (segunda columna 3 veces más ancha).
      1. Crear control **Image**: posicionarlo en **Grid**, asignar **Binding** correspondiente (con **Converter**), asignar Margin; ajustar con **AspectFit**.
      2. Crear control **Label**: posicionarlo en **Grid**, asignar **Binding** correspondiente, asignar un **Style**.
      3. Crear control Label: posicionarlo en **Grid**, asignar **Binding** correspondiente (con formato), crear un estilo en **App.xaml** y asignarlo.
4. Correr la app (se muestran las celdas de ListView correctamente).