**Ejercicio #1 – Implementar stack navigation**

1. Crear en la raíz una nueva **ContentPage**, llamada **MainPage**.
   1. En **XAML**, definir dentro de un **StackLayout** (con Padding), un label (centrado y con estilo general) y tres botones para las tres pantallas.
   2. En codebehind, generar los eventos anónimos para los cuatro botones; con **PushAsync** se agrega cada pantalla al navigation stack.
2. En **App.xaml.cs**, asignar la pantalla a **MainPage** (con **NavigationPage**).
3. Correr la app (navegación stack pero pantallas no tienen título).
4. Asignar a cada pantalla un título con atributo **Title**.
5. Correr la app (navegación stack, pantallas con título).
6. Continuar con la presentación.

**Ejercicio #2 – Implementar tab navigation**

1. \*Crear pantallas para utilizar mapas\*:
   1. En una capeta nueva (**MapsPages**), crear una **TabbedPage** llamada **MapsMainPage**. Dar un título a la pantalla (**Title**).
   2. En la misma carpeta, crear dos **ContentPage**: **MapPage** y **AccessMapPage**.
   3. En el codebehind de **MapsMainPage**, agregar las pantallas como hijas.
2. Incluir nuevo botón en **MainPage**, para acceder a **MapsMainPage**.
3. Correr la app (navegación tab - comentar espacio de tabs (recomendable 3 - 4)).
4. Arreglar padding de todas las pantallas (**Padding** de 30).
5. Incluir imágenes para las tabs:
   1. De la carpeta **Recursos**, agregar las imágenes en las carpetas correspondientes de **Android** y **iOS** (en **iOS** todas van en **Resources**).
   2. En cada pantalla, agregar el atributo **Icon** (o **IconImageSource**), con el nombre de la imagen correspondiente.
6. En **MapsMainPage.xaml**, agregar **BarBackgroundColor** y **BarTextColor** a **TabbedPage**, con los colores de los recursos en **App.xaml**.
7. Cambiar los colores de status bar y action bar:
   1. Abrir el archivo **styles.xml**:
      1. Usar el color hex azul de **Resources** en **App.xaml** para el action bar (**colorPrimary**).
      2. Usar el color azul claro que era de action bar en status bar (**colorPrimaryDark**).
8. Correr la app (tabs con imágenes y color distinto; status y action bar con colores distintos).
9. Continuar con la presentación.

**Ejercicio #3 – Crear drawer menu**

1. Crear una master page:
   1. Crear una **ContentPage** en la raíz de la aplicación, llamada **CdisMasterPage**.
   2. En **XAML**, agregar cinco botones, uno por cada pantalla de la app (incluir la pantalla **MainPage**).
   3. Darle un valor al atributo **Title** de **ContentPage**.
   4. Cambiar el estilo de los botones:
      1. Crear estilo **menuBtnStyle** en **App.xaml** (color de fondo transparente y color de texto azul); asignarlo a los botones del menú.
2. Crear una **MasterDetailPage**:
   1. Crear una clase (**.cs**) llamada **CdisMasterDetailPage**.
   2. Hacerla pública y que herede de **MasterDetailPage** (agregar **using** **Xamarin.Forms**).
   3. Hacer que **Master** sea **CdisMasterPage** y que **Detail** sea **MainPage**.
   4. Aplicar **MasterBehavior** de **Popover**.
3. En **App.xaml.cs**, asignar **CdisMasterDetailPage** a **MainPage**.
4. \***iOS** no proporciona un ícono para el drawer\*:
   1. De la carpeta Recursos, agregar las imágenes correspondientes a **Resources** del proyecto de **iOS**.
   2. En **CdisMasterPage**, agregar la etiqueta **Icon** (o **IconImageSource**) con los valores correspondientes.
5. Correr la app (se muestra un menú pop-up (sin navegación)).
6. Continuar al siguiente ejercicio.

**Ejercicio #4 – Implementar drawer navigation**

1. En la raíz del proyecto, crear la clase **PageTypes.cs**; definir un **enum** llamado **PageType**, con los nombres de las cinco pantallas de la app.
2. Crear un evento en la master page, para los botones; en el codebehind de **CdisMasterPage**:
   1. Crear el evento **PageSelected**, con el **enum** **PageType** como parámetro.
   2. Asignar el evento **PageSelected** al evento **Clicked** de cada botón, con el parámetro de **PageType** correspondiente.
3. Llamar al evento en **CdisMasterDetailPage**, para cambiar la pantalla de Detail:
   1. En el constructor, generar el handler para el evento **PageSelected** de **master**.
   2. En el handler:
      1. Definir un objeto **Page**.
      2. En un **switch**, asignar la pantalla correspondiente al objeto **Page**, de acuerdo con el valor del parámetro **PageType**.
      3. Asignar el objeto **Page** al stack de navegación de **Detail**.
      4. Cambiar la propiedad **IsPresented** a false (antes, alentar un poco la main thread con método **Delay** de **Task**).
4. Correr la app (drawer con navegación).
5. Continuar al siguiente ejercicio.

**Ejercicio #5 – Incluir un mapa en la app**

1. En los tres proyectos, descargar el paquete **Nuget** **Xamarin.Forms.Maps,** que coincida con la versión de **Xamarin.Forms** (actualizar el paquete, si se requiere).
2. Configuraciones para iOS:
   1. En **AppDelegate**, llamar al método **Init** de **FormsMaps** (después de **Forms.Init**).
   2. En **Info.plist**, agregar las llaves **NSLocationAlwaysUsageDescription**, **NSLocationWhenInUsageDescription y NSLocationAlwaysAndWhenInUseUsageDescription**.
3. Configuraciones para Android:
   1. En **MainActivity**, llamar al método **Init** de **FormsMaps** (después de **Essentials.Init**).
   2. En **AndroidManifest**, agregar los permisos **AccessCoarseLocation**, **AccessFineLocation**, **AccessLocationExtraCommands**, **AccessMockLocation**, **AccessNetworkState**, **AccessWifiState** e **Internet.**
   3. En **AndroidManifest.xml**, agregar la llave para utilizar la **API** de Google Maps; en **value** va la llave que se generará.
   4. Para generar la llave:
      1. Se necesita conocer la clave **SHA-1** del archivo **keystore** que genera Android (seguir las instrucciones del artículo de Microsoft):
         1. Se debe correr el comando de ejemplo (modificando la ruta del **keystore**) en la carpeta **bin** del **jdk** de carpeta Java (o Android) …
         2. Copiar la clave **SHA-1**.
      2. Crear un proyecto en Google APIs:
         1. Seguir las instrucciones del artículo de Microsoft (al final, copiar la clave de la API) …
      3. Pegar la clave de la API en **AndroidManifest.xml**.
4. Crear la UI de **MapPage**: dentro de un **StackLayout** (sin **Padding**), un mapa (incluir la etiqueta **maps** con **xmlns** a la librería de **Forms.Maps**), y dentro de otro **StackLayout** (con **Padding**), tres botones; darle un título a la pantalla …
5. En el codebehind de **MapPage**:
   1. Incluir **Xamarin.Forms.Maps**.
   2. En el constructor, mostrar el mapa en una ubicación en particular y asignar el atributo **Clicked** a cada botón …
   3. Asignar el tipo de mapa correcto en cada evento …
6. Correr la app; asegurarse que haya acceso a Internet (se muestra el mapa).
7. Continuar al siguiente ejercicio.

**Ejercicio #6 – Acceder a una aplicación de mapas**

1. \*Se utilizará la API de **Xamarin.Essentials**\*.
2. Crear la UI de **AccessMapPage**: **StackLayout** (con **Padding**) con tres labels, tres cajas de texto y un botón (estilos para labels y botón).
3. En el codebehind de **AccessMapPage**:
   1. Incluir la librería **Xamarin.Essentials**.
   2. En el constructor, crear un handler para el evento **Clicked** del botón. En el handler:
      1. Validar que los valores de latitud y longitud sean correctos (hacer que el handler sea **async**).
      2. Generar un objeto **Location** y un **MapLaunchOptions** y llamar al método **OpenAsync** de **Xamarin.Essentials.Map**.
4. Correr la app (se abre una aplicación de mapas).
   1. Validar los valores de lat y long y ver los diferentes modos de navegación; comentar el nombre de la ubicación (sólo aparece sin Internet)\*.
5. Incluir un **ScrollView** en todas las pantallas (**EXCEPTO** en la pantalla del mapa).
6. Correr la app (se abre una app de mapas).