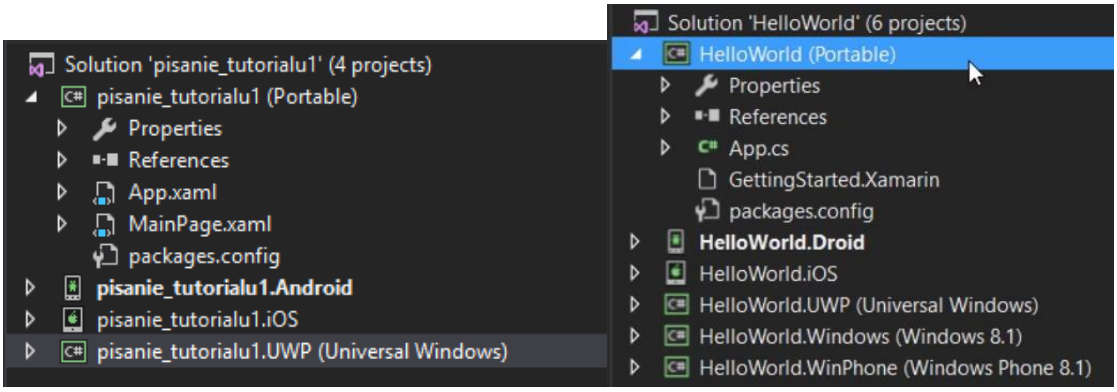
1. Vytvoríme nový projekt vo Visual Studio: V menu **File – New - Project**. Vyberieme umiestnenie a názov projektu podľa vlastného výberu. **Ak už postup vytvárania projektu poznáme, začíname bodom 10.**
2. Vyberieme **Cross-Platform – Cross-Platform App (Xamarin.Forms).** V inej verzii to môže byť **Blank App (Xamarin.Forms Portable).**
3. Na ďalšej obrazovke (ak sa objaví) vyberieme **Blank App** a **UI Technology – Xamarin.Forms** a **Code Sharing Strategy – Portable Class Library (PCL)**
4. Zatlačíme OK a vytvorí sa projekt. Zavrieme okno, kde nám ponúka Mac Agenta a odsúhlasíme target verzie UWP podprojektu.
5. Mali by sa nám vytvoriť minimálne 4 podprojekty (Portable, Android, iOS a UWP). Prípadne tam môžeme mať ešte Windows 8.1 a Windows Phone 8.1. Existujú 2 varianty, ako sa nám to môže vytvoriť, po novom by to mal byť ten naľavo už s nachystanou MainPage.



6. Klikneme pravým tlačidlom na **Solution,** ktorý je umiestnený nad podprojektami a vyberieme **Manage NuGet Packages for Solution.**
7. Vyberieme položku Update a updatneme knižnice, aby sme boli aktuálni. **POZOR – niekedy toto môže byť skôr kontraproduktívne...**
8. Ak sa nám projekt vytvoril bez MainPage.xaml (variant vpravo v bode 5), tak klikneme pravým tlačidlom na Portable podprojekt a vyberieme **Add – New item**. Tu vyberieme **Forms Blank Content Page Xaml.** Ak nie je napísané vyslovene **blank,** tak vyberieme **Forms Xaml Page.** Pomenujeme ju napríklad **GreetPage.**
9. Ak sa nám projekt vytvoril s **MainPage.xaml**, tak pracujeme rovno s ním. Ďalej už budeme používať v návode tento názov, hoci ho môžete mať nazvaný aj **GreetPage.xaml** z bodu 8.

10. Ideme urobiť jednoduchú aplikáciu, v ktorej si precvičíme zoznamy.
11. Do MainPage.xaml napíšeme kód (pred `</ContentPage>` a ak tam máme Label, tak ho zmažeme): `<ListView x:Name="listView" />`
12. Do kódu po InitializeComponent napíšeme náš zoznam a pošleme ho do komponentu zo XAML - listView:

```
var names = new List<string>
      {
          "Peter",
          "Robert",
          "Andrej"
      };

listView.ItemsSource = names;
```

13. Spustíme aplikáciu a vidíme štruktúrovaný zoznam.

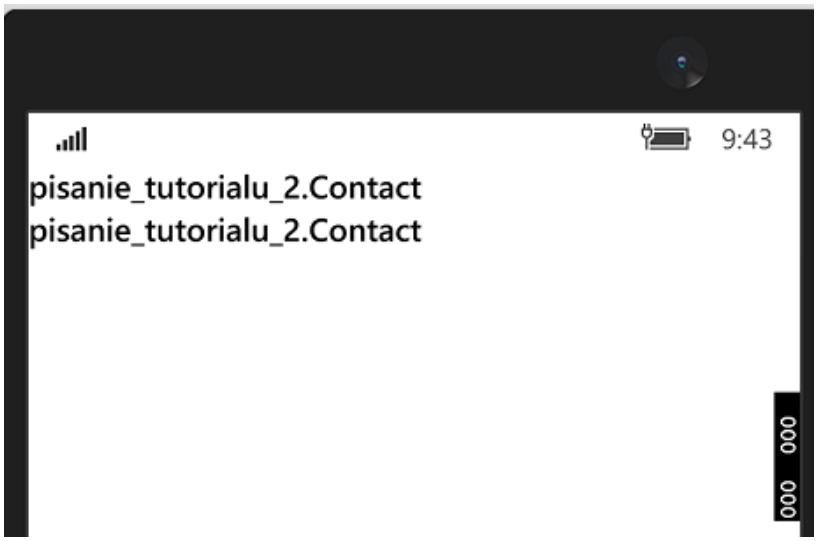
14. Môžeme sa pohrať s atribútmi komponentu `<ListView`, a to napríklad `SeparatorVisibility` a `SeparatorColor.` Vyzerá to, že na nových systémoch (minimálne Windows a Android) už separátor nie je podporovaný a nezobrazuje sa.
15. Ideme zobrazovať zoznam kontaktov. V **Portable podprojekte** vytvoríme nový priečinok nazvaný **Models**.
16. Do tohto priečinka pridáme novú triedu **Contact.cs** a naplníme ju obsahom (skontrolujeme si, či nám napísal rovnaký `namespace` ako v ostatných súboroch):

```
public class Contact
    {
        public string Name { get; set; }
        public string Status { get; set; }
        public string ImageUrl { get; set; }
    }
```

17. V kóde zmažeme všetko okrem InitializeComponent() a napíšeme:

```
listView.ItemsSource = new List<Contact>
            {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };
```

18. Vidíme, že výsledok je nežiaduci:



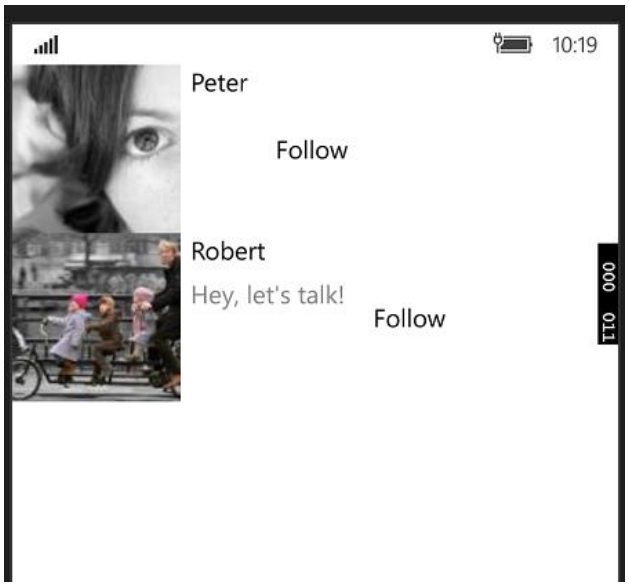
19. Keďže nemáme stringy, s ktorým listView defaultne pracuje, musíme zmeniť kód v XAML na:

```
<ListView x:Name="listView">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ImageCell Text="{Binding Name}" Detail="{Binding
Status}" ImageSource="{Binding ImageUrl}"/>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
```

20. Pozrieme si výsledok a vyzerá omnoho lepšie. Pri texte máme aj obrázky. Chceme si však pre takýto zoznam urobiť vlastnú šablónu – template.
21. Zo XAML zmažeme riadok `<ImageCell` a nahradíme ho:

```
<ViewCell>
<StackLayout Orientation="Horizontal">
                    <Image Source="{Binding ImageUrl}"/>
                    <StackLayout>
                        <Label Text="{Binding Name}" />
                        <Label Text="{Binding Status}"
TextColor="Gray" />
                    </StackLayout>
                   <Button Text="Follow" />
 </StackLayout>
<ViewCell>
```

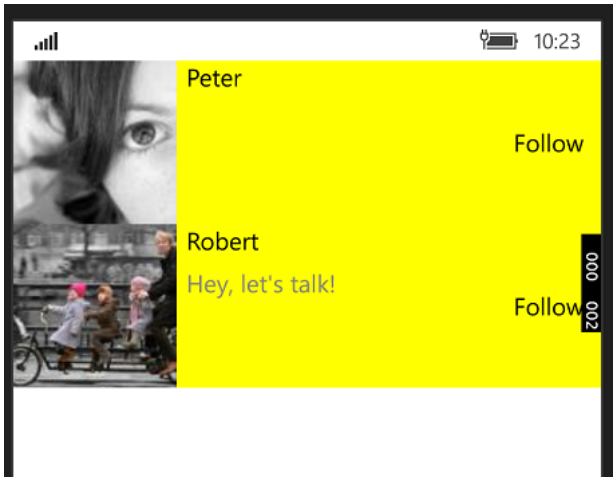
22. Dostaneme toto, čo nie je úplne pekné:



23. Medzi `<ViewCell> </ViewCell>` dáme teda iný kód:

```
<StackLayout Orientation="Horizontal" BackgroundColor="Yellow">
                    <Image Source="{Binding ImageUrl}"/>
                    <StackLayout HorizontalOptions="StartAndExpand">
                        <Label Text="{Binding Name}" />
                        <Label Text="{Binding Status}" TextColor="Gray" />
                    </StackLayout>
                    <Button Text="Follow" />
</StackLayout>
```

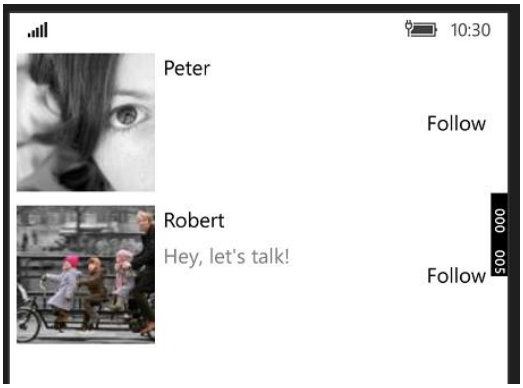
24. Teraz to vyzerá lepšie:



25. Zišiel by sa nám nejaký padding a farba je škaredá. Vymeníme teda kód za:

```
<ListView x:Name="listView" HasUnevenRows="True">
        <ListView.ItemTemplate>
            <DataTemplate>
            <ViewCell>
                <StackLayout Orientation="Horizontal" Padding="5">
                    <Image Source="{Binding ImageUrl}"/>
                    <StackLayout HorizontalOptions="StartAndExpand">
                        <Label Text="{Binding Name}" />
                        <Label Text="{Binding Status}" TextColor="Gray" />
                    </StackLayout>
                    <Button Text="Follow" />
                </StackLayout>
            </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

26. HasUnevenRows oznamuje zhruba to, že každý riadok môžu mať inú výšku, ktorá sa nastaví podľa obsahu bunku. Aplikácia vyzerá dobre:



27. Ak by sme chceli zoskupiť kontakty podľa začiatočného písmena, tak postupujeme nasledovne. Máme na mysli niečo takéto:

28. Do priečinka **Models** pridáme novú triedu **ContactGroup.cs** a naplníme ju obsahom (skontrolujeme si, či nám napísal rovnaký `namespace` ako v ostatných súboroch):

```
public class ContactGroup : List<Contact>
    {
        public string Title { get; set; }
        public string ShortTitle { get; set; }
```

```
        public ContactGroup(string title, string shortTitle)
        {
            Title = title;
            ShortTitle = shortTitle;
        }
    }
```

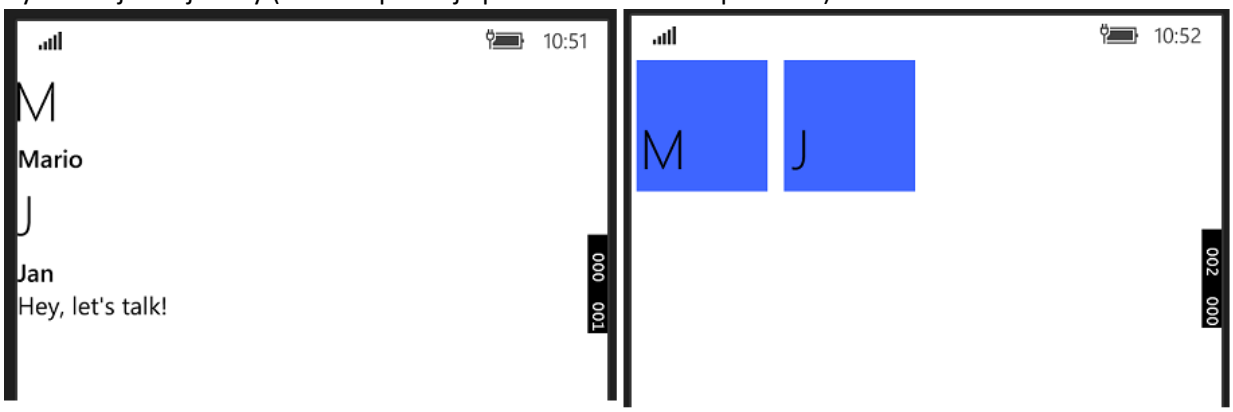
29. ShortTitle bude použité pre renderovanie malých modrých písmen v pravom dolnom rohu na obrázku v bode 27. V Androide a Windowse takýto prvok nebudeme mať.
30. V kóde pod InitializeComponent musíme opraviť kód:

```
listView.ItemsSource = new List<ContactGroup>
            {
                new ContactGroup ("M" , "M" ) {
                    new Contact { Name = "Mario", ImageUrl =
"http://lorempixel.com/100/100/people/1"}
                },
                 new ContactGroup ("J" , "J" ) {
                    new Contact { Name = "Jan", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
                },
            };
```

31. Zjednodušíme a zmeníme XAML kód:

```
<ListView x:Name="listView" IsGroupingEnabled="True" GroupDisplayBinding="{Binding
Title}" GroupShortNameBinding="{Binding ShortTitle}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <TextCell Text="{Binding Name}" Detail="{Binding Status}" />
            </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

32. Výsledok je zaujímavý (screen vpravo je po kliknutí na veľké písmeno):



33. Ideme sa teraz venovať výberu položiek v zozname. Pre tieto účely sme náš ListView zjednodušili:

```
<ListView x:Name="listView">
        <ListView.ItemTemplate>
            <DataTemplate>
                <TextCell Text="{Binding Name}" Detail="{Binding Status}" />
            </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

34. Potrebujeme pridať do prvého tagu prvky pre spracovanie kliknutí na položky. Tieto tagy odporúčame písať ručne, aby sa nám hneď v kóde vytvorila príslušná metóda na spracovávanie týchto udalostí:

```
<ListView x:Name="listView" ItemTapped="listView_ItemTapped">
```

35. Ak sa nám v kóde metóda nevytvorila, tak vyzerá takto:

```
private void listView_ItemTapped(object sender, ItemTappedEventArgs e)
        {

        }
```

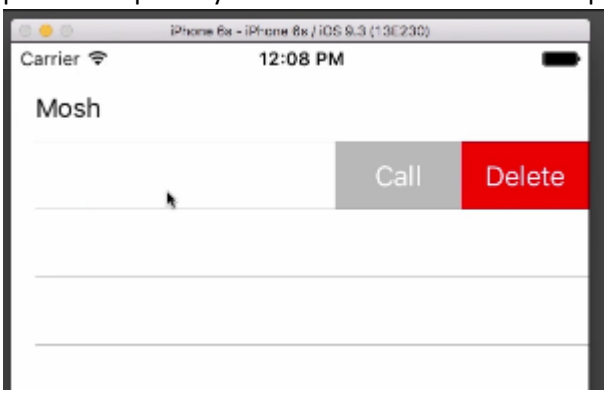
36. Metódu `listView_ItemTapped` naplníme obsahom:

```
var contact = e.Item as Contact;
DisplayAlert("Tapped", contact.Name, "OK");
```

37. V hlavnej triede našej stránky zmažeme všetko okrem InitializeComponent() a napíšeme:

```
listView.ItemsSource = new List<Contact>
            {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };
```

38. Odskúšame a vidíme, že funguje.
39. Teraz chceme urobiť kontextové akcie, ako napríklad **Call** a **Delete**, ktoré sa zobrazia po potiahnutí položky zoznamu doľava na iOS alebo podržaním na Windows 10:



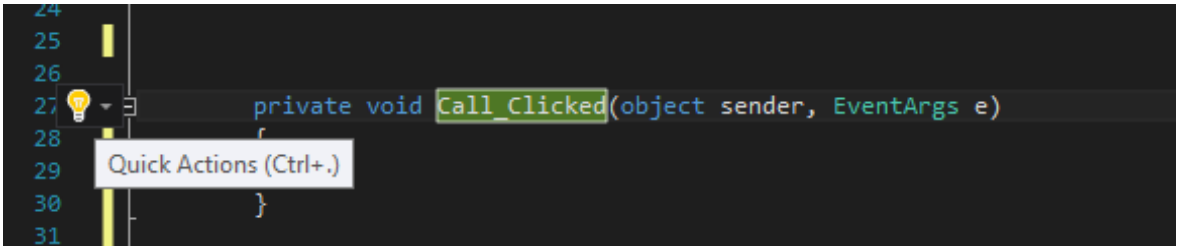
40. Z kódu zmažeme metódu `listView_ItemTapped`.
41. Do XAML pred `</ContentPage>` napíšeme, pričom metódu Clicked opäť píšeme ručne:

```
<ListView x:Name="listView">
        <ListView.ItemTemplate>
            <DataTemplate>
                <TextCell Text="{Binding Name}" Detail="{Binding Status}">
                    <TextCell.ContextActions>
                        <MenuItem Text="Call" Clicked="MenuItem_Clicked" />
                    </TextCell.ContextActions>
                </TextCell>
            </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

42. Ak sa nám v kóde metóda nevytvorila, tak vyzerá takto:

```
private void MenuItem_Clicked(object sender, EventArgs e)
        {

        }
```

43. V kóde metódu premenujeme na `Call_Clicked` a zatlačením žiarovky sa nám premenuje aj v XAML:



44. Do XAML pridáme na miesto pod MenuItem ďalší riadok a rovnako zabezpečíme vytvorenie príslušnej metódy v kóde ako v predošlom bode:
```
<MenuItem Text="Delete" Clicked="Delete_Clicked" IsDestructive="True"/>
```
45. Do metódy Call_Clicked napíšeme kód:
```
var menuItem = sender as MenuItem;
var contact = menuItem.CommandParameter as Contact;
DisplayAlert("Call", contact.Name, "OK");
```
46. Potrebujeme ešte upraviť XAML (bodka znamená, že berieme všetky property – v tomto prípade triedy Contact):
```
<MenuItem Text="Call" Clicked="Call_Clicked" CommandParameter="{Binding .}" />
<MenuItem Text="Delete" Clicked="Delete_Clicked" IsDestructive="True"
CommandParameter="{Binding .}"/>
```
47. Odskúšame funkčnosť. Malo by všetko fungovať, avšak chceme implementovať reálne fungujúce mazanie.
48. Do metódy Delete_Clicked napíšeme kód:
```
var contact = (sender as MenuItem).CommandParameter as Contact;
_contacts.Remove(contact);
```
49. Po kóde „public partial class MainPage : ContentPage {“ napíšeme:
```
private List<Contact> _contacts;
```
50. Po InitializeComponent zmažeme kód a napíšeme:
```
_contacts = new List<Contact>
            {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };

listView.ItemsSource = _contacts;
```
51. Vidíme, že po spustení aplikácia síce kontakty maže, ale nie je to vidno. Potrebujeme použiť iný typ zoznamu.
52. Hore do **using** zóny napíšeme: `using System.Collections.ObjectModel;`
53. Kód „`private List<Contact> _contacts;`“ nahradíme:
```
private ObservableCollection<Contact> _contacts;
```
54. ObservableCollection tiež použijeme v hlavnej triede pri deklarovaní zoznamu.
55. Všetko teraz funguje, ako má.
56. Teraz sa ideme naučiť **Pull to Refresh**. Z kódu zmažeme všetko okrem InitializeComponent a pod neho napíšeme:
```
listView.ItemsSource = GetContacts();
```
57. Pre tieto účely sme náš ListView zjednodušili. Metódu Refreshing píšeme ručne, aby sa nám vytvoril nový Event Handler v kóde. Tento postup už poznáme:
```
<ListView x:Name="listView" IsPullToRefreshEnabled="True"
Refreshing="listView_Refreshing">
        <ListView.ItemTemplate>
            <DataTemplate>
```

```
                    <TextCell Text="{Binding Name}" Detail="{Binding Status}" />
                </DataTemplate>
            </ListView.ItemTemplate>
    </ListView>
```

58. Vytvorená metóda by mala vyzerať takto:

```
private void listView_Refreshing(object sender, EventArgs e)
        {

        }
```

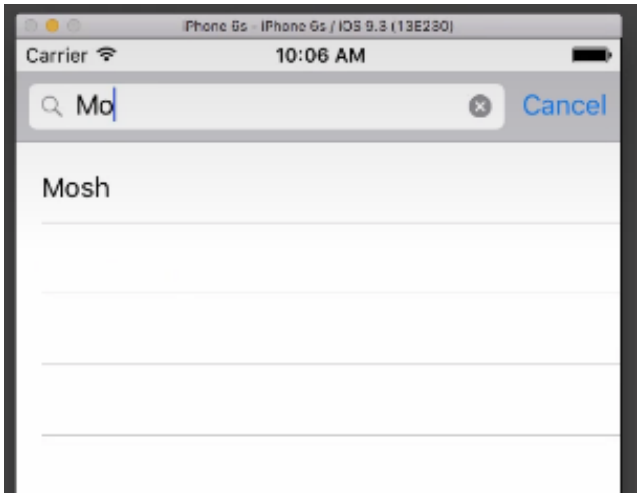
59. Nakoľko sme webservisy, ktoré by nám v tomto prípade mali zabezpečiť stiahnutie nového zoznamu kontaktov po obnovení zoznamu, ešte nepreberali, tak budeme takúto funkcionalitu emulovať.
60. Po kóde „`public partial class MainPage : ContentPage {`“ napíšeme:

```
List<Contact> GetContacts()
 {
     return new List<Contact>  {
         new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
         new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
 };
 }
```

61. Do metódy listView_Refreshing napíšeme:

```
listView.ItemsSource = GetContacts();
listView.EndRefresh();
```

62. Aplikáciu odskúšame a vidíme, že funguje. **Nefunguje však na Windows (oficiálne)...**
63. Ideme teraz pridať **Search Bar.** Bude vyzerať takto:



64. Ideme modifikovať súčasný projekt. Kód v XAML medzi tagmi ContentPage vymeníme za:

```
<StackLayout>
        <SearchBar Placeholder="Search..." />
        <ListView x:Name="listView">
            <ListView.ItemTemplate>
                <DataTemplate>
                   <TextCell Text="{Binding Name}" Detail="{Binding Status}" />
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
</StackLayout>
```

65. V kóde modifikujeme metódu GetContacts():

```
List<Contact> GetContacts()
        {
            var contacts = new List<Contact>  {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };

            return contacts;
}
```

66. Za InitializeComponent máme: `listView.ItemsSource = GetContacts();`
67. Zmažeme metódu `listView_Refreshing`.
68. V XAML do prvku `SearchBar` pridáme prvok `TextChanged` a dbáme, aby sa vytvorila metóda:

```
private void SearchBar_TextChanged(object sender, TextChangedEventArgs e)
        {

        }
```

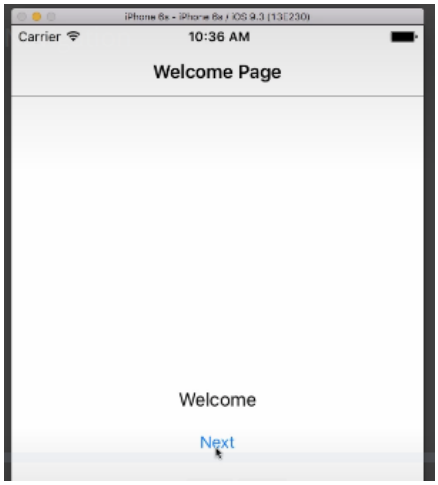
69. Metódu GetContacts zmeníme na:

```
IEnumerable<Contact> GetContacts(string searchText = null)
        {
            var contacts = new List<Contact>  {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };

            if (String.IsNullOrWhiteSpace(searchText))
                return contacts;

            return contacts.Where(c => c.Name.StartsWith(searchText));
}
```

70. Metódu SearchBar_TextChanged naplníme:
    `listView.ItemsSource = GetContacts(e.NewTextValue);`
71. Aplikáciu vyskúšame a vidíme, že vyhľadávanie funguje, hoci je case sensitive.

---

72. Ďalšou tematikou na precvičenie je **navigácia medzi stránkami mobilnej aplikácie.**
73. Prvým typom je hierarchická navigácia.



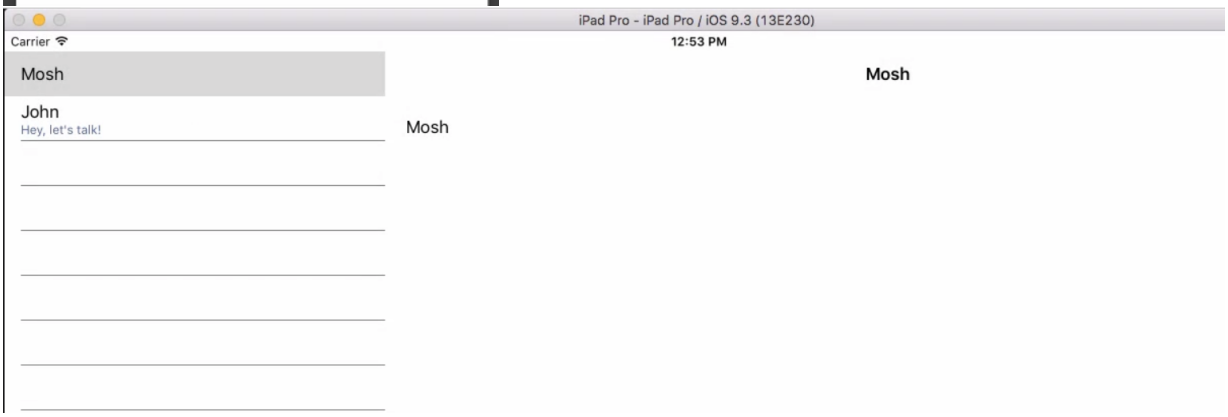
74. V takomto type navigácie sa posúvame medzi jednotlivým stránkami pomocou tlačidla Next a na predošlú stránku sa vieme dostať pomocou tlačidla Back. Toto sa používa napríklad pri rôznych uvítacích obrazovkách mobilných aplikácii, kde sa človek dozvie o rôznych funkcionalitách aplikácie.
75. Pokračujeme v súčasnom projekte (kontakty budeme ešte potrebovať) a vytvoríme do Portable podprojektu 2 stránky (Forms Blank Content Page XAML) s názvami **WelcomePage** a **IntroductionPage.**
76. Do **WelcomePage.xaml** napíšeme pred </ContentPage> (pri Clicked dávame pozor, aby sa vytvorila príslušná metóda v kóde):
```
<StackLayout HorizontalOptions="Center" VerticalOptions="Center">
        <Label Text="Welcome" HorizontalOptions="Center"/>
        <Button Text="Next" Clicked="Button_Clicked"/>
</StackLayout>
```
77. Za **<ContentPage** napíšeme:
`Title=”Welcome”`
78. Do **IntroductionPage.xaml** napíšeme pred </ContentPage> (pri Clicked dávame pozor, aby sa vytvorila príslušná metóda v kóde):
```
<StackLayout HorizontalOptions="Center" VerticalOptions="Center">
        <Label Text="This is how the app works." />
        <Button Text="Back" Clicked="Button_Clicked"/>
</StackLayout>
```
79. Za **<ContentPage** napíšeme:
`Title=”Introduction”`
80. Našej **WelcomePage** ideme upravovať kód. Do **Button_Clicked** metódy napíšeme kód:
```
async void Button_Clicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new IntroductionPage());
        }
```
81. Našej **IntroductionPage** ideme upravovať kód. Do **Button_Clicked** metódy napíšeme kód:
```
async void Button_Clicked(object sender, EventArgs e)
        {
            await Navigation.PopAsync();
        }
```
82. Jasne vidíme, že tu funguje princíp zásobníka (stack). Keď sa chceme vrátiť na predošlú stránku, stačí z vrchnej časti zásobníka odstrániť súčasnú stránku (operácia Pop).

83. V **App.xaml.cs** (alebo App.cs) zmeníme riadok určujúci MainPage na:
`MainPage = new NavigationPage (new WelcomePage());`
84. Spustíme aplikáciu a vidíme, že funguje.
85. Ak by sme chceli skryť hornú lištu (navigation bar) s názvom stránky, tak to elementu ContentPage pridáme tag `NavigationPage.HasNavigationBar="False"`
86. Teraz si ideme odskúšať Master Detail.

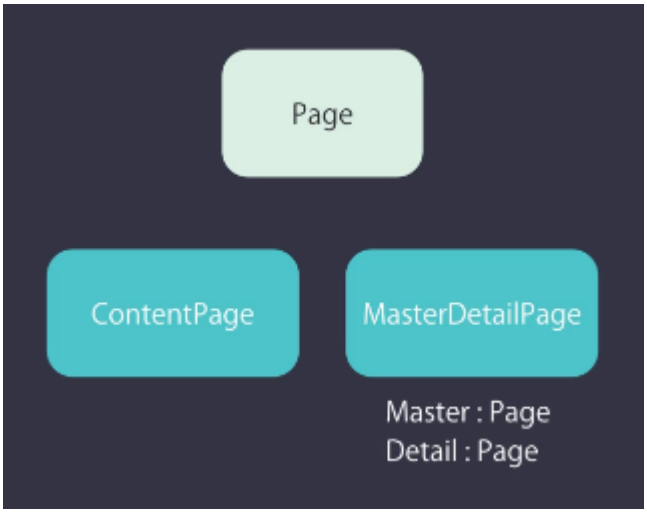
87. Vytvoríme do Portable podprojektu 2 stránky (Forms Blank Content Page XAML) s názvami **ContactsPage** a **ContactDetailPage.**
88. V kóde našej stránky **ContactsPage** zmažeme všetko okrem InitializeComponent() a napíšeme:
```
listView.ItemsSource = new List<Contact>
            {
                new Contact { Name = "Peter", ImageUrl =
"http://lorempixel.com/100/100/people/1"},
                new Contact { Name = "Robert", ImageUrl =
"http://lorempixel.com/100/100/people/2", Status="Hey, let's talk!"}
            };
```
89. Do **ContactsPage.xaml** napíšeme pred </ContentPage>:
```
<ListView x:Name="listView">
        <ListView.ItemTemplate>
            <DataTemplate>
                <TextCell Text="{Binding Name}" Detail="{Binding
Status}" />
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
```
90. Za **<ContentPage** napíšeme:
```
Padding="0,20,0,0" Title="Contacts"
```
91. Do **ContactDetailPage.xaml** napíšeme pred </ContentPage>:
```
<Label Text="" />
```
92. Za **<ContentPage** napíšeme:
```
Padding="20"
```
93. Vrátime sa do **ContactsPage.xaml**. Do komponentu ListView pridáme tag `ItemSelected="listView_ItemSelected"` , pričom dbáme, aby sa vytvorila aj priradená metóda v kóde, do ktorej napíšeme:
```
void listView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
  {
    var contact = e.SelectedItem as Contact;
    Navigation.PushAsync(new ContactDetailPage() { BindingContext = contact });
  }
```
94. Všimnime si, ako sa odovzdáva referencia na konkrétny kontakt.
95. Do **ContactDetailPage.xaml** napíšeme pred </ContentPage>:
```
<Label Text="{Binding Name}" />
```
96. Za **<ContentPage** napíšeme:
```
Title="{Binding Name}"
```
97. V **App.xaml.cs** (alebo App.cs) zmeníme riadok určujúci MainPage na:
```
MainPage = new NavigationPage(new ContactsPage());
```
98. Odskúšame aplikáciu a vidíme, že funguje. Avšak po návrate na hlavnú stránku zostáva naša položka stále označená, že je vybraná. Vyriešime zmenou kódu v ContactsPage.xaml.cs:
```
async void listView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
        {
            if (e.SelectedItem == null)
                return;
            var contact = e.SelectedItem as Contact;
            await Navigation.PushAsync(new ContactDetailPage() { BindingContext
= contact });
            listView.SelectedItem = null;
        }
```
99. Vidíme, že všetko je OK.

100. Ideme urobiť podobnú funkcionalitu, avšak so vstavaným typom stránky **MasterDetailPage.**

101. Vyzerá nejak takto, avšak jej hlavná výhoda je vidieť na tabletoch:



102. MasterDetailPage má 2 property: Master a Detail:



103. Našu **ContactsPage** ideme prerobiť z typu ContentPage na MasterDetailPage. V ContactsPage.xaml necháme len „`<?xml version="1.0" encoding="utf-8" ?>`“ a napíšeme tam nasledovný kód, pričom `pisanie_tutorialu_2` nahradíme názvom nášho projektu:

```
<MasterDetailPage  xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="pisanie_tutorialu_2.ContactsPage" IsPresented="true">
    <MasterDetailPage.Master>
        <ContentPage Padding="0,20,0,0" Title="Contacts">
            <ListView x:Name="listView" ItemSelected="listView_ItemSelected">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextCell Text="{Binding Name}" Detail="{Binding Status}" />
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </ContentPage>
    </MasterDetailPage.Master>
    <MasterDetailPage.Detail>
    </MasterDetailPage.Detail>
</MasterDetailPage>
```

104. Všimneme si, že MasterDetailPage obsahuje 2 podstránky a to **Master** a **Detail**, čo je zjavné zo štruktúry XAML.
105. V kóde zameníme `public partial class ContactsPage : ContentPage` za:
     `public partial class ContactsPage : MasterDetailPage`
106. K jednotlivým podstránkam vieme potom pristupovať pomocou **this.Master** a **this.Detail**.
107. Musíme sa ešte vrátiť do ContactDetailPage.xaml.cs a napísať tam:

```
public ContactDetailPage(Contact contact)
        {
            if (contact == null)
                throw new ArgumentNullException();

            BindingContext = contact;

            InitializeComponent();
        }
```

108. V kóde ContactsPage.xaml.cs zmažeme kód:

```
if (e.SelectedItem == null)
                return;
await Navigation.PushAsync(new ContactDetailPage() { BindingContext = contact });
listView.SelectedItem = null;
```

109. Za `var contact = e.SelectedItem as Contact;` napíšeme:

```
Detail = new ContactDetailPage(contact);
IsPresented = false;
```

110. Keďže odovzdávame parameter **contact**, preto sme v bode 107 museli robiť úpravu.
111. Metóde `listView_ItemSelected` dáme preč modifikátor **async.**
112. Do XAML za `<MasterDetailPage.Detail>` napíšeme:

```
<ContentPage />
```

113. V **App.xaml.cs** (alebo App.cs) zmeníme riadok určujúci MainPage na:

```
MainPage = new ContactsPage();
```

114. Aplikáciu odskúšame a vidíme, že funguje. Je však možné, že na niektorých platformách sa nevieme vrátiť z detailu na master page.
115. Namiesto riadka `Detail = new ContactDetailPage(contact);` napíšeme:

```
Detail = new NavigationPage (new ContactDetailPage(contact));
```

116. Ideme robiť aplikáciu, kde budú navigáciu zabezpečovať **taby**. Na iOS sú v dolnej časti displeja, na Androide a Windowse sú hore.
117. Vytvoríme nový projekt, ako je to uvedené v prvých bodoch tak, aby sme mali **MainPage.xaml**. Budeme tiež potrebovať ikony, ktoré sme vkladali na minulom cvičení do projektu. Ide o body 78 až 86 v návode v minulom cvičení. Ikony teda podľa tých bodov vložíme.
118. Ideme do **MainPage.xaml** zmažeme tag Label, ak ho tam máme:
119. Uzatvárací a otvárací tag **ContentPage** premenujeme na **TabbedPage**.
120. V MainPage.xaml.cs zmeníme kód `public partial class MainPage : ContentPage` na: `public partial class MainPage : TabbedPage`
121. Do XAML medzi tagy TabbedPage dáme:

```
<ContentPage Title="Page1" Icon="clock.png">
        <Label Text="Content of Page 1" HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
<ContentPage Title="Page2" Icon="clock.png">
        <Label Text="Content of Page 2" HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
```

122. Spustíme aplikáciu a vidíme, že funguje:

123. V reálnej situácii nebudete dávať do podstránok iba Label, ale je predpoklad, že pôjde o inú stránku. **Referencovanie** na napríklad našu **ContactsPage**, čo sme už robili, by vyzeralo takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<TabbedPage
    xmlns:local="clr-namespace:HelloWorld"
    xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.cc
    <local:ContactsPage Title="Contacts" Icon="clock.png" />
    <ContentPage Title="Page1" Icon="clock.png">
        <Label Text="Content of Page1" HorizontalOptions="Center" VerticalOptions="Cer
    </ContentPage>
    <ContentPage Title="Page2" Icon="clock.png">
        <Label Text="Content of Page2" HorizontalOptions="Center" VerticalOptions="Cer
    </ContentPage>
</TabbedPage>
```

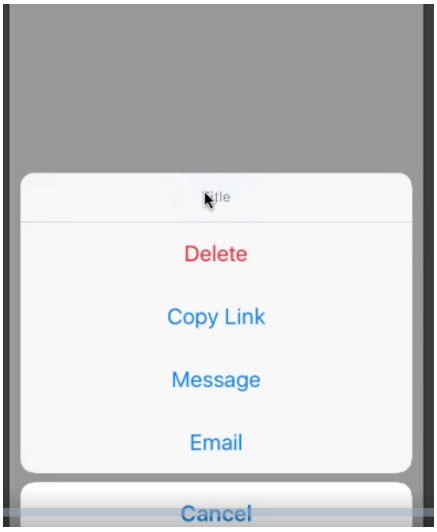
124. Čo ak by sme chceli na jednu z podstránok dať hore navigačnú lištu? Urobíme to takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<TabbedPage
    xmlns:local="clr-namespace:HelloWorld"
    xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.cc
    <local:ContactsPage Title="Contacts" Icon="clock.png" />

    <NavigationPage Title="Page1" Icon="clock.png">
        <x:Arguments>
            <ContentPage>
                <Label Text="Content of Page1" HorizontalOptions="Center" VerticalOpti
            </ContentPage>
        </x:Arguments>
    </NavigationPage>

    <ContentPage Title="Page2" Icon="clock.png">
        <Label Text="Content of Page2" HorizontalOptions="Center" VerticalOptions="Cer
    </ContentPage>
</TabbedPage>
```

125. Podobným spôsobom ako TabbedPage sa robí aj **CarouselPage**.

---

126. Ideme sa teraz naučiť robiť **pop-up menu a podobne**. Toto sa volá action sheet:



127. Vytvoríme nový projekt, ako je to uvedené v prvých bodoch tak, aby sme mali **MainPage.xaml**.

128. Do XAML pred </ContentPage> napíšeme, pričom dbáme o vytvorenie metódy ku kliknutiu:

```
<Button Text="Click" Clicked="Button_Clicked" />
```

129. Metóda v kóde by mala vyzerať:

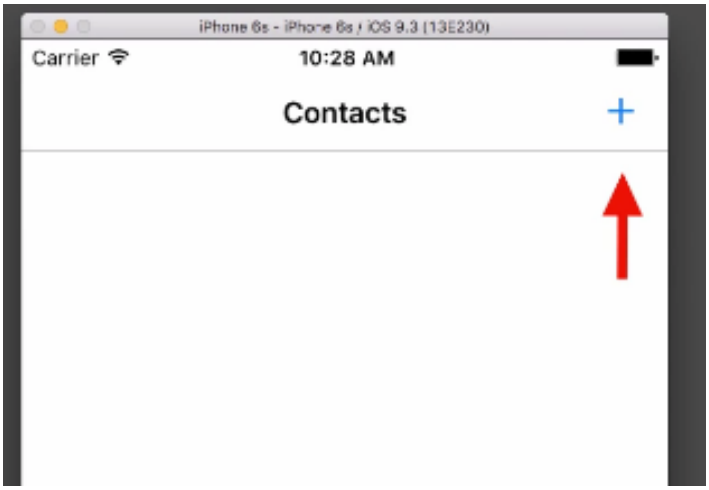
```
async void Button_Clicked(object sender, EventArgs e)
        {
            var response = await DisplayAlert("Warning", "Are you sure?",
"Yes", "No");
            if (response)
                await DisplayAlert("Done", "Your response will be saved!",
"OK");
        }
```

130. Vyskúšame aplikáciu. Ideme na spomínaný **action sheet**. Do vyššie spomenutej metódy dáme kód:

```
var response = await DisplayActionSheet("Title", "Cancel", "Delete", "Copy
Link", "Message", "Email");
await DisplayAlert("Response", response, "OK");
```

131. Vyskúšame aplikáciu. Funguje.

132. Ideme teraz robiť **nástroje toolbaru**, takže z projektu zmažeme náš button a handler k nemu. Jeden z nástrojov toolbaru je toto:



133. Stiahneme si ikonku „plus.png“ – http://uamt.fei.stuba.sk/MVI/sites/default/files/Resources_cv5.zip a známym postupom dáme ikonku do jednotlivých podprojektov našich platforiem.

134. V XAML za <ContentPage pridáme tag: `Title="Contacts"`

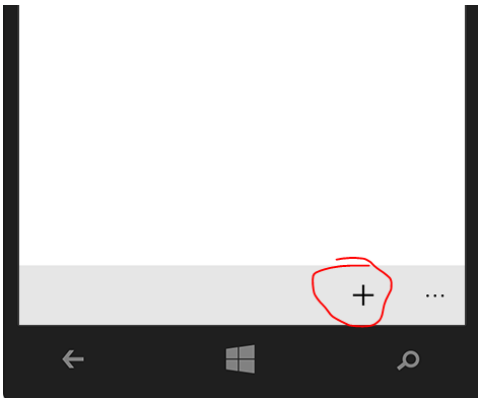
135. Pred </ContentPage> pridáme kód, pričom pri Activated dbáme o vytvorenie metódy:

```
<ContentPage.ToolbarItems>
        <ToolbarItem Icon="plus.png" Text="New"
Activated="ToolbarItem_Activated" />
    </ContentPage.ToolbarItems>
```

136. Metóda v kóde vyzerá takto:

```
void ToolbarItem_Activated(object sender, EventArgs e)
        {
            DisplayAlert("Activated", "ToolbarItem Activated", "OK");
        }
```

137. Po spustení by to malo vyzerať takto (na Androide mi to, žiaľ, nešlo správne):



138. K takémuto tlačidlu je vhodné vytvoriť tzv. modálnu stránku, kde sa očakáva od užívateľa nejaká akcia (napísanie novej správy po stlačení „+“).

139. Pred </ContentPage> pridáme kód:

```
<ContentPage.ToolbarItems>
        <ToolbarItem Icon="plus.png" Text="New"
Activated="ToolbarItem_Activated" Order="Primary" />
        <ToolbarItem Icon="plus.png" Text="New"
Activated="ToolbarItem_Activated" Order="Secondary" />
    </ContentPage.ToolbarItems>
```

140. V **App.xaml.cs** (alebo App.cs) zmeníme riadok určujúci MainPage na:

```
MainPage = new NavigationPage(new MainPage());
```

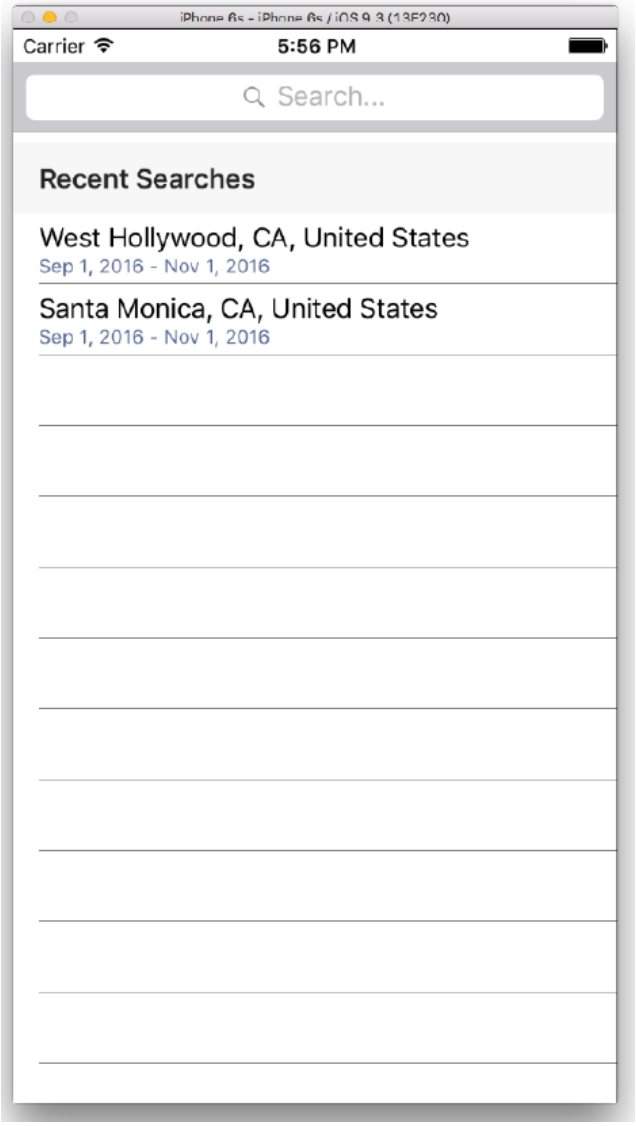
141. Aplikáciu spustíme (a už by mala ísť aj na Androide – nešla pre chýbajúcu NavigationPage).

CVIČENIE Č.5

Úloha (2 bonusové body)

Úloha slúži pre zopakovanie princípov tvorby vyhľadávacieho prvku a zoznamu v Xamarin.Forms.

Vytvorte aplikáciu (inšpirácia vpravo), kde si užívateľ bude môcť vyhľadávať medzi poslednými vyhľadávaniami letov (*Recent Searches*) do daných destinácii. Prípadne môžete *Recent Searches* poňať jednoducho ako jednotlivé lety. Vyhľadávanie medzi poslednými vyhľadávaniami letov môže znieť niekomu mätúco.



Pomôcky a pokyny:

- Vyhľadávanie nemá byť case-sensitive.
- Zoznam sa má dať refreshnúť potiahnutím dole.
- Položka zoznamu sa má dať zmazať.
- Položky zoznamu môžete mať napevno zadané v kóde.
- Každý let (*Recent Search*) má tieto properties:
  - Id (int)
  - Location (string)
  - CheckIn (DateTime)
  - CheckOut (DateTime)