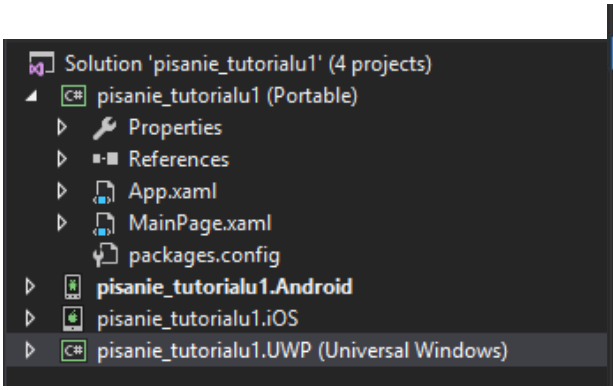
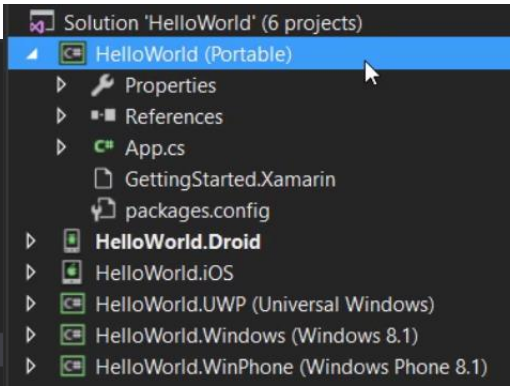
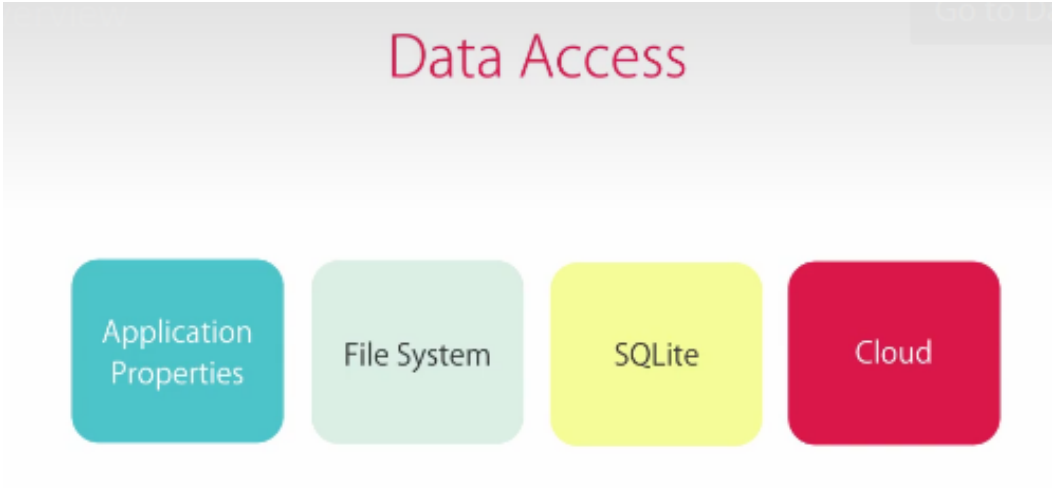
1. Vytvoríme nový projekt vo Visual Studio: V menu **File – New - Project**. Vyberieme umiestnenie a názov projektu podľa vlastného výberu.
2. Vyberieme **Cross-Platform – Cross-Platform App (Xamarin.Forms).** V inej verzii to môže byť **Blank App (Xamarin.Forms Portable).**
3. Na ďalšej obrazovke (ak 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. Ideme sa naučiť rôzne spôsoby ukladania dát v Xamarin.Forms.



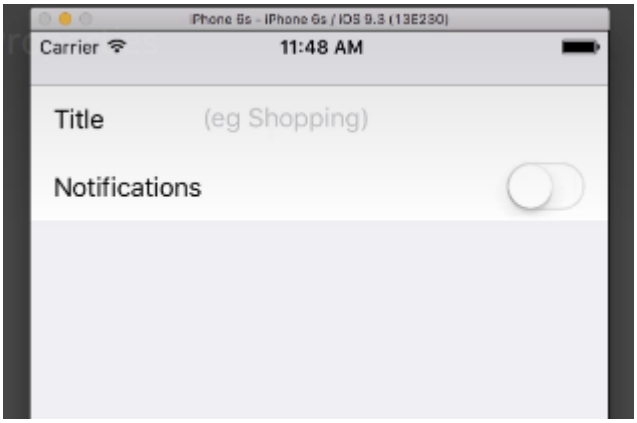
9. Začneme s **Application Properties**. Tieto fungujú ako dátový typ „dictionary“:

```
public class App : Application
{
    App()
    {
        Properties["Username"] = "mosh";
    }
}
```

10. Do XAML pred </ContentPage> dáme:

```
<TableView Intent="Form">
            <TableRoot>
                  <TableSection>
                        <EntryCell x:Name="title" Text="{Binding
Title}" Label="Title" Placeholder="(eg Shopping)" />
                        <SwitchCell x:Name="notificationsEnabled"
Text="Notifications" On="{Binding NotificationsEnabled}" />
                  </TableSection>
            </TableRoot>
      </TableView>
```

11. Vidíme toto:



12. Čo ak by sme chceli hodnoty týchto vstupných prvkov uložiť pre neskoršie použitie?
13. Do XAML prvku EntryCell doplníme Completed="title_Completed", pričom dbáme o vytvorenie handleru, ktorý bude vyzerať:

```
private void title_Completed(object sender, EventArgs e)
        {
        }
```

14. Do XAML prvku SwitchCell doplníme OnChanged="notificationsEnabled_OnChanged" , pričom dbáme o vytvorenie handleru, ktorý bude vyzerať:
15.

```
private void notificationsEnabled_OnChanged(object sender,
ToggledEventArgs e)
        {
        }
```

16. Keďže je to takto zbytočné a vzniká veľa handlerov, nazveme jeden OnChange a druhý zmažeme. Zmeníme to aj v XAML.
17. Do tejto novej metódy OnChange dáme kód:

```
Application.Current.Properties["Name"] = title.Text;
Application.Current.Properties["NotificationsEnabled"] =
notificationsEnabled.On;
```

18. Pod InitializeComponent dáme kód, ktorý zabezpečí načítanie propertov po spustení:

```
if (Application.Current.Properties.ContainsKey("Name"))
                title.Text = Application.Current.Properties["Name"].ToString();

            if
(Application.Current.Properties.ContainsKey("NotificationsEnabled"))
                notificationsEnabled.On =
(bool)Application.Current.Properties["NotificationsEnabled"];
```

19. Skúsime a vidíme, že aj po skončení aplikácie sa tieto vlastnosti uložia. Malo by to fungovať na iOS, ostatné platformy môžu mať problém. Ukážeme si teda krajšiu implementáciu.
20. Pod kód `public partial class MainPage : ContentPage {` pridáme:
```
private const string TitleKey = "Name";
private const string NotificationsEnabledKey = "NotificationsEnabled";
```
21. Vymeníme metódu OnChange a konštruktor MainPage za:
```
private void OnChange(object sender, EventArgs e)
        {
            Application.Current.Properties[TitleKey] = title.Text;
            Application.Current.Properties[NotificationsEnabledKey] =
notificationsEnabled.On;

        }

        public MainPage()
        {
            InitializeComponent();

            if (Application.Current.Properties.ContainsKey(TitleKey))
                title.Text =
Application.Current.Properties[TitleKey].ToString();

            if
(Application.Current.Properties.ContainsKey(NotificationsEnabledKey))
                notificationsEnabled.On =
(bool)Application.Current.Properties[NotificationsEnabledKey];

        }
```
22. Je to univerzálnejšie, ale stále pracujeme len v rámci jednej stránky. Čo ak máme stránok viac? Využijeme zavádzaciu triedu App.
23. Konštanty z bodu 20 **premiestnime** do **App.xaml.cs**. Pridáme tam tiež:
24.
```
public string Title
        {
            get
            {
                if (Properties.ContainsKey(TitleKey))
                    return Properties[TitleKey].ToString();
                return "";
            }

            set
            {
                Properties[TitleKey] = value;
            }
        }

        public bool NotificationsEnabled
        {
            get
            {
                if (Properties.ContainsKey(NotificationsEnabledKey))
                    return (bool)Properties[NotificationsEnabledKey];

                return false;
            }

            set
            {
```

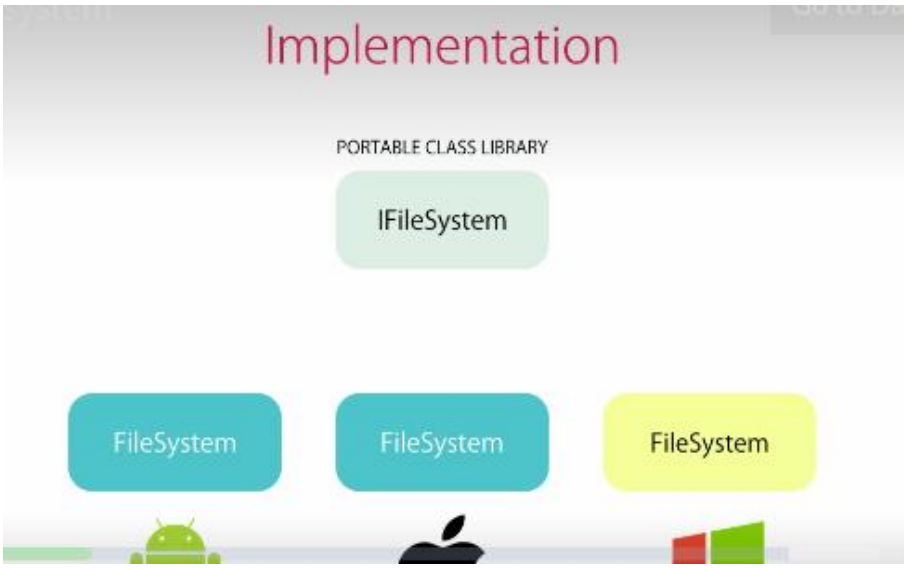
```
                Properties[NotificationsEnabledKey] = value;
            }
        }
```

25. Ideme do **MainPage.xaml.cs** a zmeníme kód v metóde OnChange na:
```
var app = Application.Current as App;
app.Title = title.Text;
app.NotificationsEnabled = notificationsEnabled.On;
```
26. Za InitializeComponent dáme iba kód:
```
BindingContext = Application.Current;
```
27. V XAML vymažeme všetky naše x:Name a tagy metód `Completed` a OnChanged. ***Zmažeme aj samotnú metódu OnChange.***
28. **Malo by fungovať.** Ak nefunguje v Androide, v Android projekte je potrebné nastaviť v Properties - Android Options - Advanced - HttpClient implementation hodnotu default a prepnúť na Android.
29. Ideme teraz na ukladanie na **súborový systém.** Z C# poznáme:

```
namespace System.IO {

    File
    FileInfo

    Directory
    DirectoryInfo

    Environment

}
```

30. Toto však vieme používať len na Android a iOS. Windows má iné metódy, ktoré sú asynchrónne:

31. Ako pracovať s 2 rôznymi metodikami, ak máme náš portable projekt? Pomocou interfejsov, tak ako to bolo uvedené na prednáške z praxe:




```
interface IFileSystem {

    string[] GetFiles(string path);

    void WriteText(string fileName, string text);
}
```

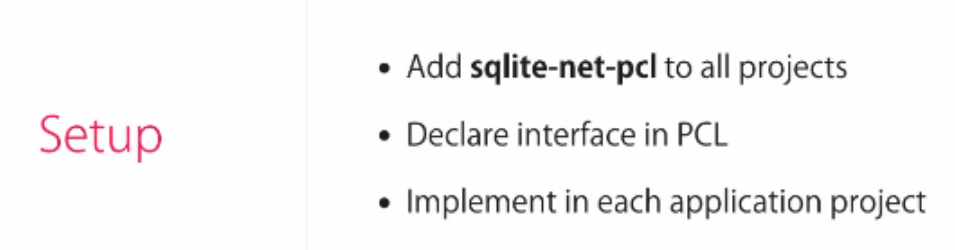
32. Zjednodušením je NuGet package PCLStorage, ktorý zabezpečuje vyššie uvedenú logiku.
33. Nájdeme ho na stránke: https://github.com/dsplaisted/PCLStorage

PCL Storage provides a consistent, portable set of local file IO APIs for .NET, Windows Phone, Windows Store, Xamarin.iOS, Xamarin.Android, and Silverlight. This makes it easier to create cross-platform .NET libraries and apps.

Here is a sample showing how you can use PCL Storage to create a folder and write to a text file in that folder:

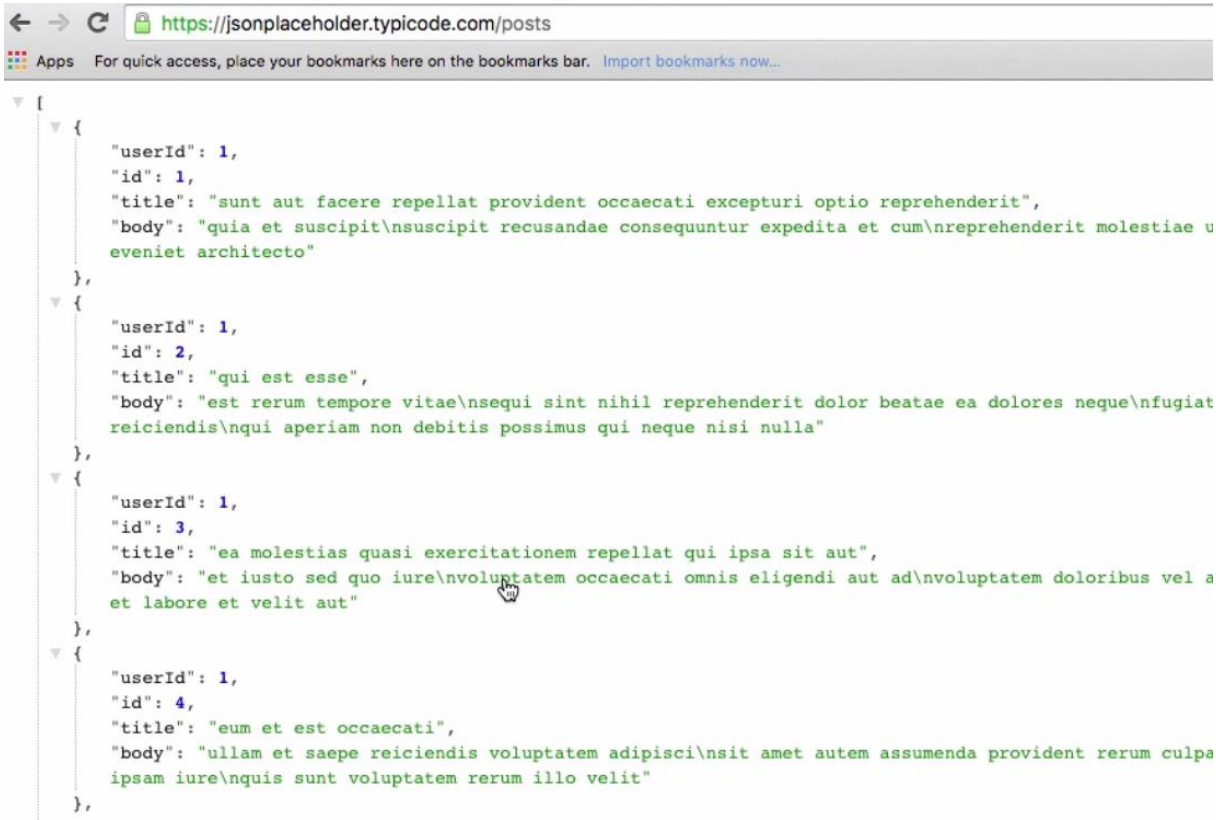
```
public async Task PCLStorageSample()
{
    IFolder rootFolder = FileSystem.Current.LocalStorage;
    IFolder folder = await rootFolder.CreateFolderAsync("MySubFolder",
        CreationCollisionOption.OpenIfExists);
    IFile file = await folder.CreateFileAsync("answer.txt",
        CreationCollisionOption.ReplaceExisting);
    await file.WriteAllTextAsync("42");
}
```

34. Teraz sa budeme zaoberať databázou **SQLite**. Čo je lightweight databázový systém.



35. Stiahneme si projekt http://uamt.fei.stuba.sk/MVI/sites/default/files/Todo.zip. Otvoríme si ho. Ak by niečo nefungovalo, tak nainštalujeme NuGet package **sqlite-net-pcl**.
    Opis projektu máme na stránke: https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/databases/
36. Projekt otvoríme a analyzujeme jeho štruktúru.

37. Ideme sa teraz naučiť pracovať s webservismi (REST). Teoretický úvod môžete nájsť tu: https://drive.google.com/file/d/0ByqSuaFYkqNwbm5WNk95ZG1iYmM/view?usp=sharing
38. Budeme pracovať s **fejkovými webservismi** na stránke https://jsonplaceholder.typicode.com/ , pričom budeme pracovať s postami.



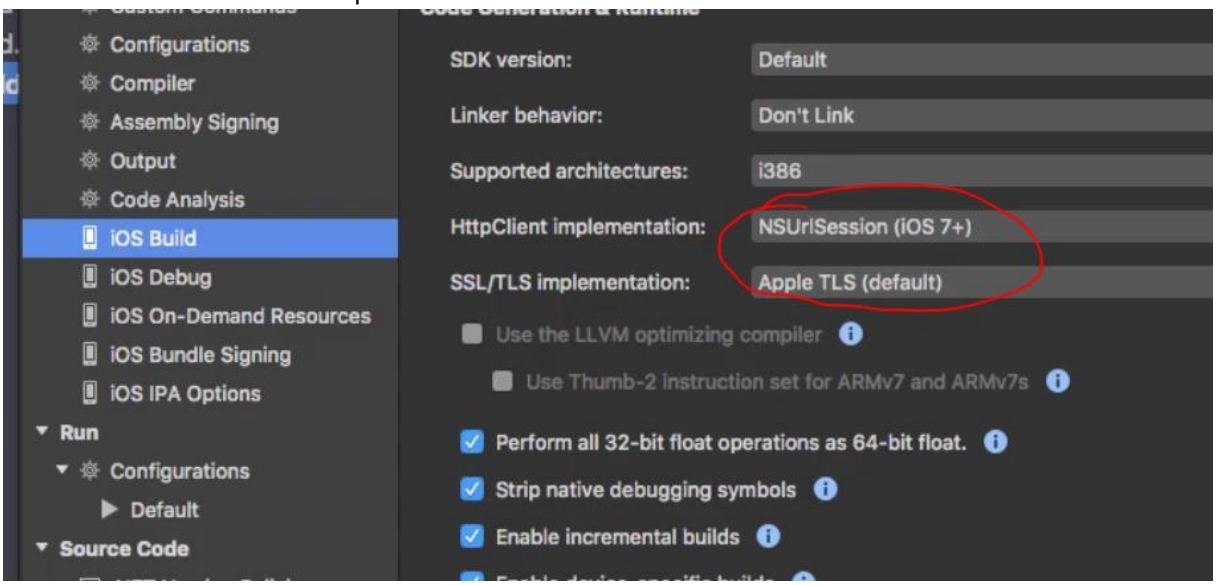
39. Vytvoríme si nový projekt. Do **MainPage.xaml.cs** dáme kód:

```
public class Post
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Body { get; set; }
    }

    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
        }
        void OnAdd(object sender, System.EventArgs e)
        {
        }
        void OnUpdate(object sender, System.EventArgs e)
        {
        }
        void OnDelete(object sender, System.EventArgs e)
        {
        }
    }
```

40. Vidíme, že sme si pripravili **triedu pre posty** a tiež prázdne metódy pre základné operácie s nimi.
41. Do XAML dáme:

```
<StackLayout>
            <StackLayout Orientation="Horizontal">
                  <Button Text="Add" Clicked="OnAdd" />
                  <Button Text="Update" Clicked="OnUpdate"
HorizontalOptions="CenterAndExpand" />
                  <Button Text="Delete" Clicked="OnDelete" />
            </StackLayout>
            <ListView x:Name="postsListView">
                  <ListView.ItemTemplate>
                        <DataTemplate>
                              <TextCell Text="{Binding Title}" />
                        </DataTemplate>
                  </ListView.ItemTemplate>
            </ListView>
      </StackLayout>
```

42. Potrebujeme nainštalovať NuGet Package **Newtonsoft.Json a Microsoft.Net.Http** do všetkých podprojektov.
43. Pre iOS bude možno treba sa pohrať s hodnotami:



44. Do kódu po „`public partial class MainPage : ContentPage {`“ pridáme kód:

```
private const string Url = "https://jsonplaceholder.typicode.com/posts";
private HttpClient _client = new HttpClient();
private ObservableCollection<Post> _posts;
```

45. Do hornej časti pridáme potrebný namespace:

```
using System.Net.Http;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
```

46. Pred kód `base.OnAppearing();` napíšeme:

```
protected override async void OnAppearing()
        {
            var content = await _client.GetStringAsync(Url);
            var posts = JsonConvert.DeserializeObject<List<Post>>(content);
```

```
            _posts = new ObservableCollection<Post>(posts);
            postsListView.ItemsSource = _posts;
```

47. Spustíme aplikáciu a vidíme, že po jej spustení (preto je to v metóde OnAppearing) sa zobrazia posty. Ideme teraz urobiť **pridávanie** postov.
48. Metódu OnAdd vymeníme za:
49.
```
async void OnAdd(object sender, System.EventArgs e)
        {
            var post = new Post { Title = "Title" + DateTime.Now.Ticks };

            var content = JsonConvert.SerializeObject(post);
            await _client.PostAsync(Url, new StringContent(content));

            _posts.Insert(0, post);
        }
```

50. Vyskúšame a vidíme, že nám pridáva posty s aktuálnym časom (respektíve tým, čo predstavuje `DateTime.Now.Ticks`).
51. Metódu OnUpdate vymeníme za:

```
async void OnUpdate(object sender, System.EventArgs e)
        {
            var post = _posts[0];
            post.Title += " UPDATED";

            var content = JsonConvert.SerializeObject(post);
            await _client.PutAsync(Url + "/" + post.Id, new
StringContent(content));

        }
```

52. Metódu OnDelete vymeníme za:

```
async void OnDelete(object sender, System.EventArgs e)
        {
            var post = _posts[0];
            await _client.DeleteAsync(Url + "/" + post.Id);
            _posts.Remove(post);
        }
```

53. Vidíme, že všetko funguje, ale update funguje spôsobom, že ho nevidíme. Keby sme ho chceli vidieť musíme implementovať interface `INotifyPropertyChange`. Príklad môžeme vidieť vo videu: https://drive.google.com/file/d/0ByqSuaFYkqNwelNGblJBVHBQbEk/view?usp=sharing
54. Užitočný ťahák k rôznemu typom prístupu dát: https://drive.google.com/file/d/0ByqSuaFYkqNwd3FfZ3ZyQVJna0U/view?usp=sharing

CVIČENIE TÝŽDEŇ Č.7 – BONUSOVÁ ÚLOHA

Úloha (1 bonusový bod)

Doplňte aplikáciu z bodov 37 až 54 o funkčne sa zobrazujúci update. Bude treba využiť interface `INotifyPropertyChange`.