



Katedra Systemów Geoinformatycznych

Systemy mobilne

Laboratorium

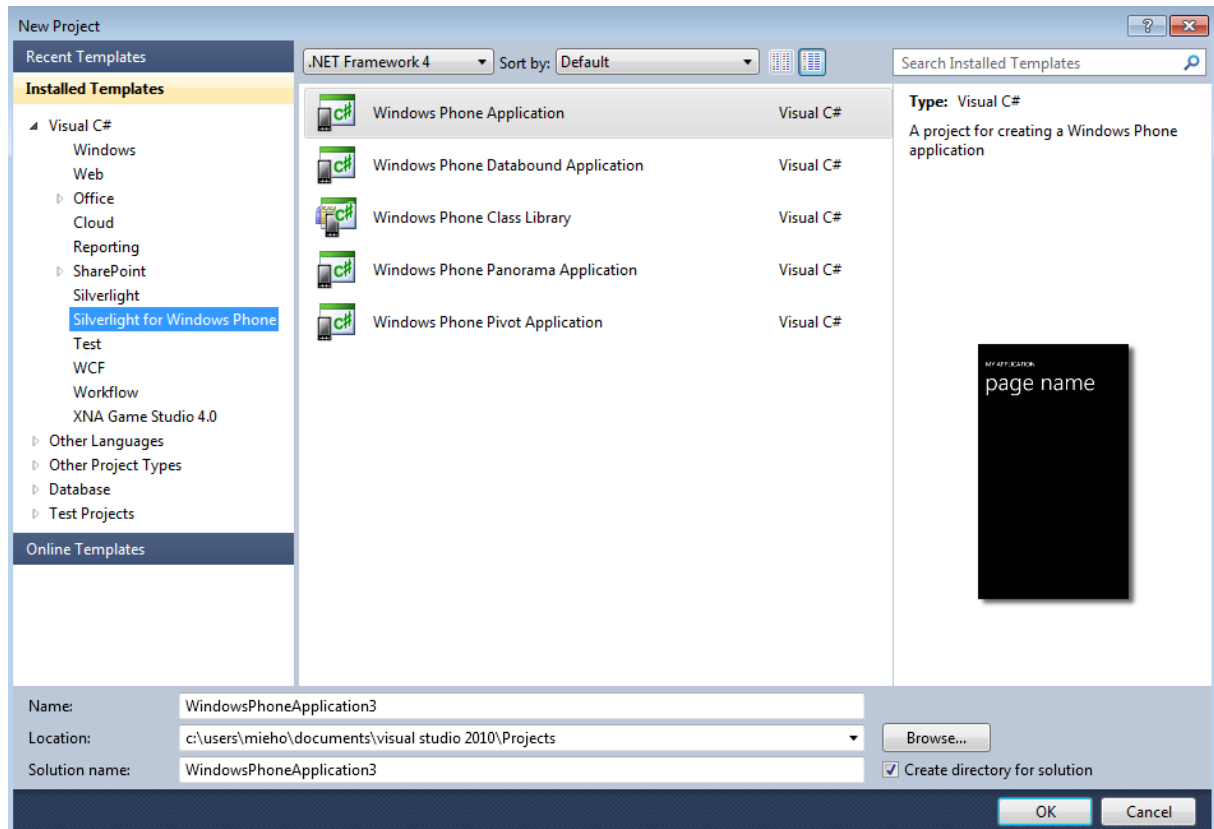
Windows Phone 7 tworzenie aplikacji Silverlight



Zadanie 1

Zadanie polega na zapoznaniu się ze środowiskiem Visual Studio 2010, stworzeniu projektu przeznaczonego na urządzenia mobilne oraz napisaniu prostej aplikacji GUI.

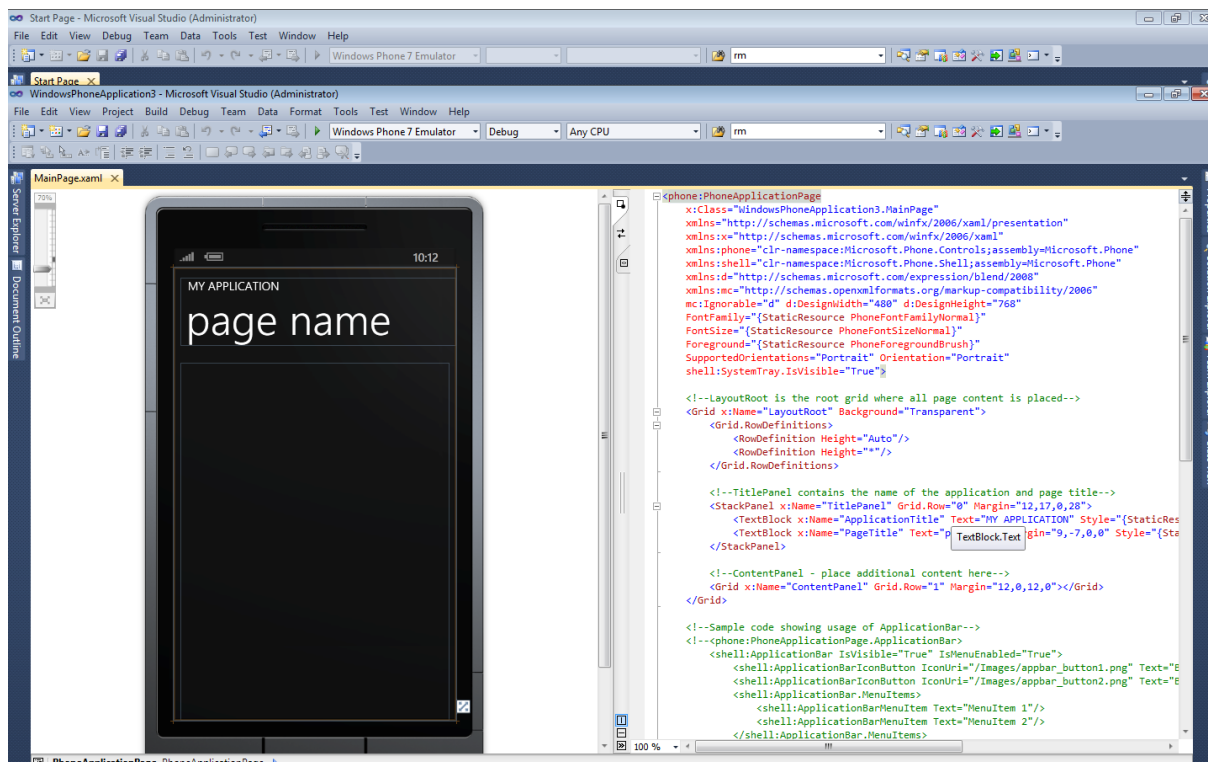
Aby stworzyć nowy projekt należy wybrać opcję File->New->Project, pojawi się okno wyboru typu tworzonego projektu:



Z dostępnych opcji z listy po lewej stronie wybieramy „Silverlight for Windows Phone”, a następnie „Windows Phone Application”.

Podajemy nazwę projektu w polu Name i zatwierdzamy przyciskiem OK.

Po chwili nasz projekt zostaje utworzony.



Po lewej stronie widzimy graficzną reprezentację kodu pliku .xaml (prawa strona). W programowaniu w Silverlight'cie pliki z rozszerzeniem .xaml odpowiadają za część wizualną projektu, natomiast korespondujące z nimi pliki .xaml.cs za logikę aplikacji.

Aby przetestować działanie aplikacji wystarczy nacisnąć F5. Aplikacja zostanie wrzucona domyślnie na emulator. Jeżeli mamy podłączony telefon do komputera możemy z listy po prawej stronie od przycisku „Start Debugging” wybrać „Windows Phone 7 Device” co spowoduje wrzucenie naszej aplikacji na telefon. Jeżeli zdecydujemy się jednak na testowanie aplikacji przy użyciu emulatora, należy pamiętać, że jest on osobnym programem, symulującym działanie telefonu komórkowego. Nie należy również przy każdym zastopowaniu aplikacji wyłączać emulatora.

Uruchomienie aplikacji na emulatorze

Aby sprawdzić poprawność napisanego kodu należy nacisnąć F6, co spowoduje przekompilowanie projektu. Po naciśnięciu F5 program zostaje załadowany na emulator.

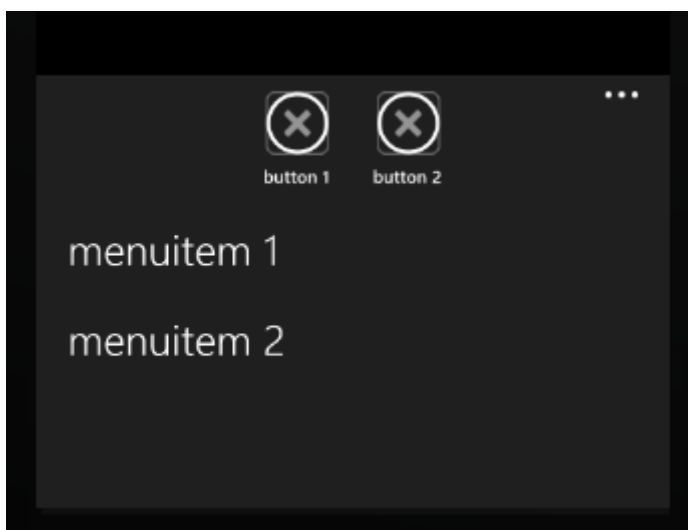
Aplikacja napisana na platformę Windows Phone 7 może korzystać z menu aplikacji, czego rolę pełni ApplicationBar. Jego działanie można zobaczyć po odkomentowaniu kodu w pliku MainPage.xaml:

```

<!--Sample code showing usage of ApplicationBar-->
<!--<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
    <shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png" Text="Button 1"/>
    <shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png" Text="Button 2"/>
    <shell:ApplicationBar.MenuItems>
      <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
      <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
    </shell:ApplicationBar.MenuItems>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar-->

```

Do odkomentowania bloku kodu służy skrót klawiszowy ctrl+k+u, analogicznie do zakomentowania ctrl+k+c. Po usunięciu komentarza widzimy zmiany w oknie designera, a po uruchomieniu aplikacji widzimy czym w rzeczywistości jest ApplicationBar.



Zadanie 2

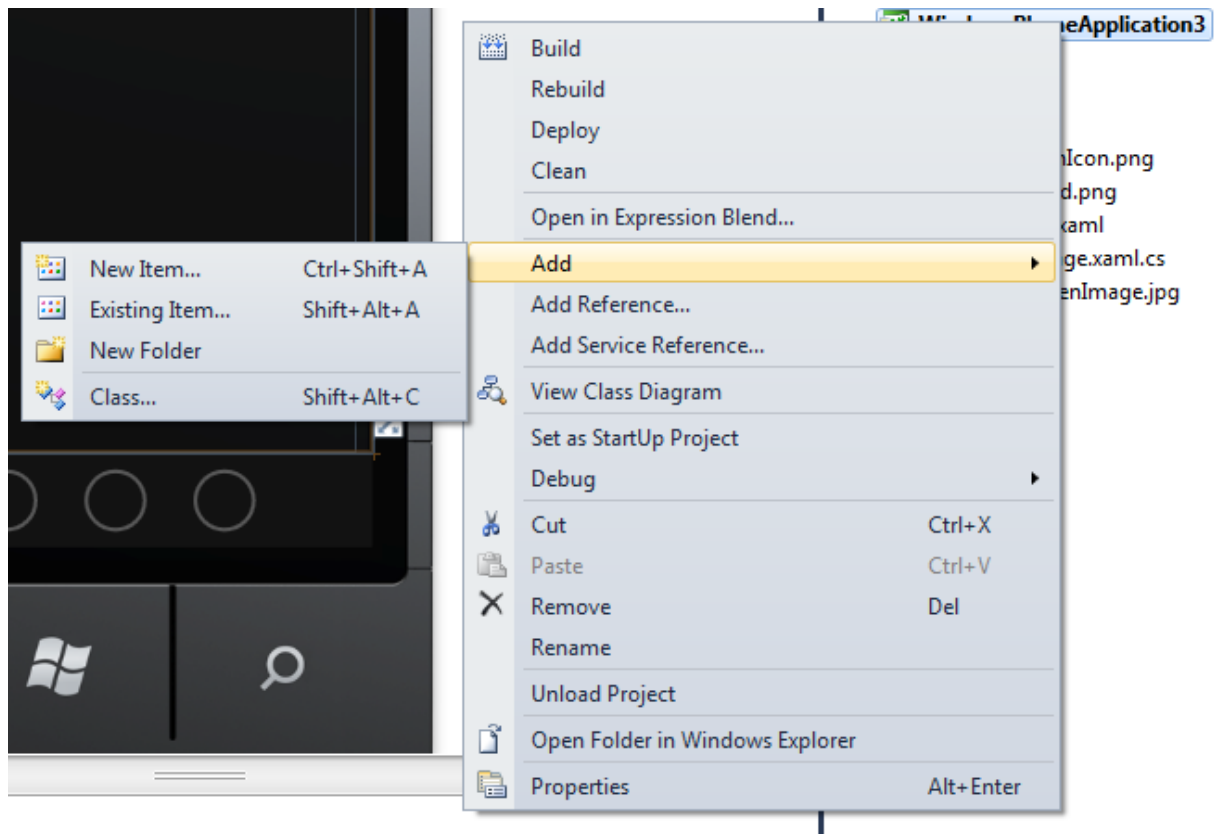
Zadanie polega na stworzeniu prostego interfejsu programu. Zadanie to realizujemy tworząc prostą aplikację wyświetlającą posty danego użytkownika portalu Twitter.com.

Z okna Toolbox'a przeciągamy do Designer'a interesujące nas elementy : TextBox do wprowadzenia nazwy użytkownika, Button do zatwierdzenia wyboru i pobrania postów, oraz ListBox do wyświetlenia pobranych postów. Końcowe okno powinno mieć postać :

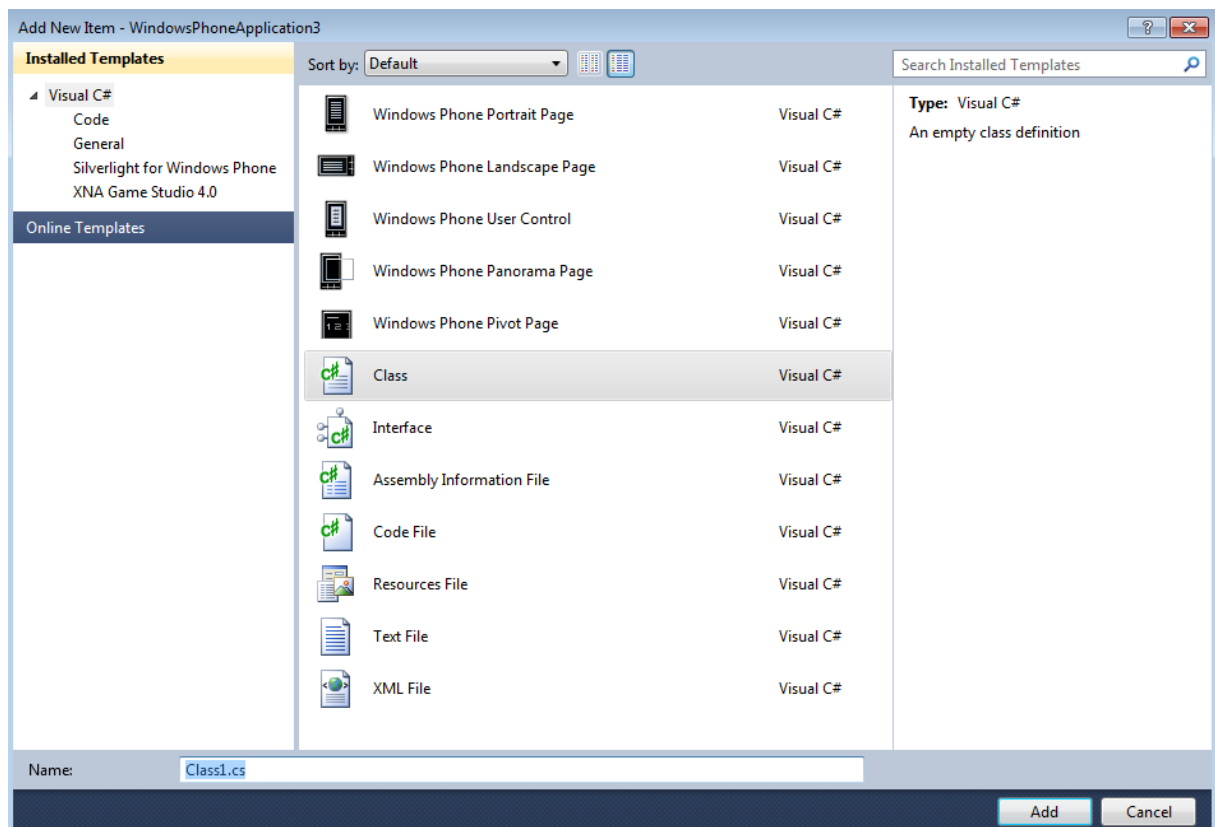


Warto zauważyć, że przeciągając elementy z ToolBox'a zostaje automatycznie wygenerowany kod w pliku .xaml. Możemy dowolnie przełączać się między widokiem Designera a kodem pliku .xaml, poprzez podwójne kliknięcie na przycisk „Design”, bądź „XAML”.

Chcąc wypełnić listę pojedynczymi postami będziemy potrzebowali dodatkowej klasy odzwierciedlającej pojedynczego posta. Aby dodać plik z nową klasą wystarczy kliknąć prawym przyciskiem myszy na nazwę projektu w Solution Explorer, następnie Add->Class.



W powstałym oknie podajemy w polu Name nazwę klasy i zatwierdzamy przyciskiem Add



Następnie dodajemy pola, odpowiadające postom:

```
private string _userName;

public string UserName
{
    get { return _userName; }
    set { _userName = value; }
}

private string _message;

public string Message
{
    get { return _message; }
    set { _message = value; }
}

private string _imageSource;

public string ImageSource
{
    get { return _imageSource; }
    set { _imageSource = value; }
}
```

Następnie musimy dodać metodę odpowiedzialną za pobranie postów danego użytkownika.

Posty będą pobierane po naciśnięciu uprzednio dodanego przycisku. Aby dodać obsługę zdarzenia, należy wejść w Designera, następnie kliknąć prawym przyciskiem myszy na interesujący nas przycisk i z listy wybrać „Properties”. Widzimy jak rozwinęła się nam lista właściwości obiektu. Możemy tu bezpośrednio edytować wszystkie właściwości odpowiadające za wygląd danego przycisku. Jednak chcąc obsłużyć zdarzenie kliknięcia myszką musimy przejść do drugiej zakładki „Events” i kliknąć podwójnie w puste pole po prawej stronie od napisu Click.

Zostanie automatycznie wygenerowana metoda obsługująca kliknięcie danego przycisku.

Do pobrania postów posłużymy się obiektem klasy WebClient, która ma metody pozwalające na pobieranie danych z określonej przez Uri lokalizacji. Po utworzeniu instancji klasy WebClient po naciśnięciu „.” przy obiekcie tej klasy widzimy listę wszystkich właściwości oraz metod które udostępnia. Z listy wybieramy DownloadStringAsync, a jako parametr wejściowy podajemy nowy obiekt klasy Uri:

```
new Uri("http://api.twitter.com/1/statuses/user_timeline.xml?screen_name=" +
textBoxUsername.Text)
```

Pisząc programy w Silverlight'cie spotyka się asynchroniczne wywołanie metod. Chcąc obsłużyć zdarzenie zakończenia wywołania metody musimy dodać EventHandlera. Przykładowo dla metody obiektu „twitter” DownloadStringAsync, mamy DownloadStringCompleted.

Chcąc dodać EventHandlera piszemy więc twitter.DownloadStringCompleted += i następnie naciskamy dwa razy klawisz TAB i kod zostaje dla nas automatycznie wygenerowany.

W ciele tej metody musimy sparsować otrzymane dane oraz dodać je do listboxa, za co odpowiada poniższy kod:

```
if (e.Error != null)
{
    return;
}
XElement TweetList = XElement.Parse(e.Result);

ParsedTweetList = from tweet in TweetList.Descendants("status")
                  select new TwitterItem
                  {
                      ImageSource = tweet.Element("user").Element("profile_image_url").Value,
                      Message = tweet.Element("text").Value,
                      UserName = tweet.Element("user").Element("screen_name").Value
                  };

listBoxTweets.ItemsSource = null;
listBoxTweets.ItemsSource = ParsedTweetList.ToList<TwitterItem>();
```

W e.Result mamy dane które zwróciła metoda DownloadString, a ponieważ są to dane w formacie xml, to posłużymy się obiektem klasy XElement do parsowania.

Następnie obiektowi ParsedTweetList przypiszemy listę nowo stworzonych obiektów klasy TwitterItem.

Obiekt ParsedTweetList jest obiektem typu IEnumerable<TwitterItem>:

```
private IEnumerable<TwitterItem> ParsedTweetList;
```

Formuła odpowiadająca za utworzenie listy postów z pliku xml, jest formułą LINQ, która swoją składnią przypomina zapytania SQL tylko, że w odwróconej kolejności.

listBoxTweets jest to listbox, w którym będziemy wyświetlać posty, stąd potrzebne jest ustawienie ItemsSource na nowo stworzoną listę. Ostatnią rzeczą przed uruchomieniem aplikacji jest zadbanie o prawidłowe wyświetlenie się postów, jak chociażby obrazków danego użytkownika (ImageSource).

Aby to zrobić należy w pliku MainPage.xaml, dla naszego listbox'a zdefiniować wygląd pojedynczego elementu listy. Robi się to przy pomocy zdefiniowania ItemTemplate, co ilustruje poniższy kod:

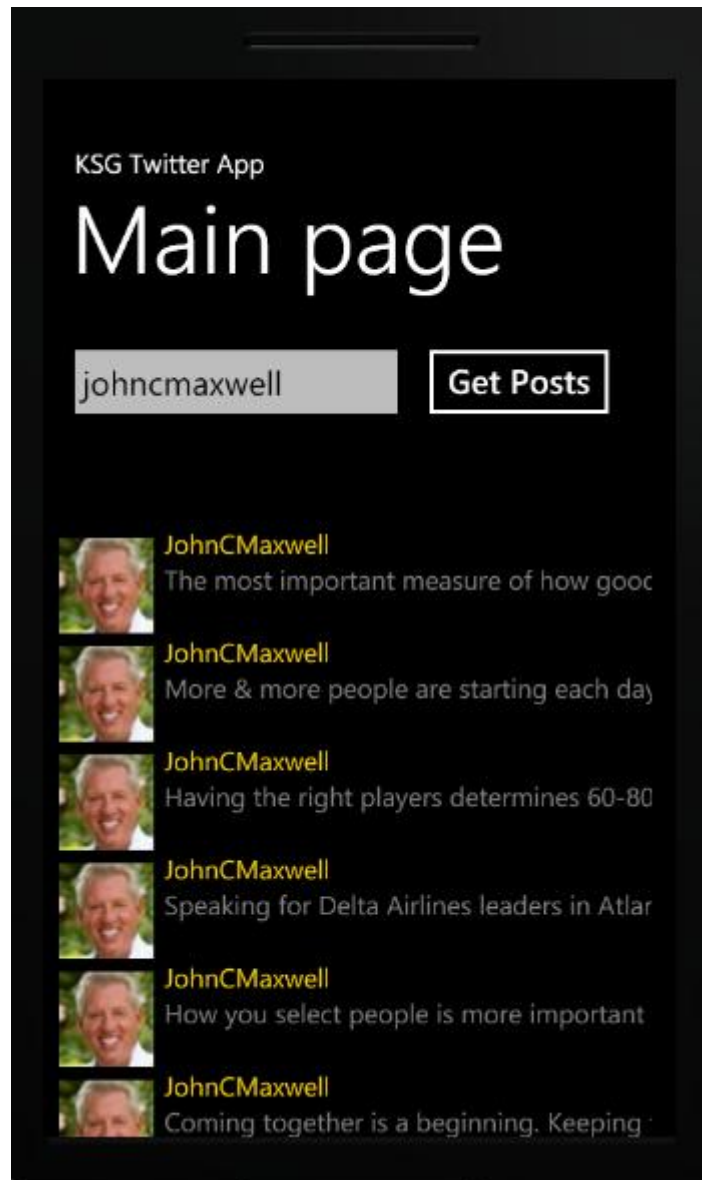
```
<ListBox Name="listBoxTweets" Height="521" HorizontalAlignment="Stretch">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal" Height="Auto">
        <Image Source="{Binding ImageSource}" Height="72" VerticalAlignment="Top" Margin="0,10,8,0"/>
        <StackPanel Width="370">
          <TextBlock Text="{Binding UserName}" Foreground="Gold"/>
          <TextBlock Text="{Binding Message}" Foreground="Gray"/>
        </StackPanel>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

W powyższym kodzie występuje również słowo Binding. Mechanizm bindingu jest bardzo wygodny, gdyż pozwala w łatwy i przejrzysty sposób zdefiniować miejsce, z którego będą pobierane dane dla konkretnego obiektu i tak np.

```
<TextBlock Text="{Binding UserName}" Foreground="Gold"/>
```

oznacza, że text textblock'a odnosi się do zmiennej UserName, która jest atrybutem klasy TwitterItem.

Po wykonaniu powyższych kroków nadszedł czas na przetestowanie działania aplikacji:



Zadanie 3

Zadanie polega na dodaniu możliwości zapisu postów znajdujących się aktualnie na liście do pliku .xml do IsolatedStorage. Dodatkowo dodanie możliwości usunięcia pliku z IsolatedStorage.

Aby wykonać to zadanie należy dodać do aplikacji dwa przyciski oraz obsłużyć ich zdarzenia Click.

Aby utworzyć plik w IsolatedStorage należy posłużyć się następującymi instrukcjami:

```
IsolatedStorageFile isoStorage = IsolatedStorageFile.GetUserStoreForApplication();
```

```
IsolatedStorageFileStream file = isoStorage.OpenFile(FILENAME, FileMode.Create);
```

Obiekt isoStorage odpowiada za dostęp do IsolatedStorage aplikacji, natomiast obiekt file umożliwia dostęp do pliku o nazwie zdefiniowanej w FILENAME.

Dla przycisku Zapisz skorzystamy z XmlSerializer'a:

```
XmlSerializer ser = new XmlSerializer(typeof(List<TwitterItem>));
```

Następnie przy pomocy metody Serialize() obiektu XmlSerializer zapiszemy dane do pliku .xml.

Dla przycisku Usuń wystarczy wywołać metodę DeleteFile() obiektu klasy IsolatedStorageFile.

Na sam koniec należy dodać uzupełnianie listBox'a postami z pliku .xml z IsolatedStorage, przy starcie aplikacji.

```
Loaded += new RoutedEventHandler(MainPage_Loaded);
```

Przy definicji ciała metody należy również posłużyć się obiektem klasy XmlSerializer do deserializacji danych z pliku .xml, a następnie ustawić ItemsSource listBoxa na nowo utworzoną listę obiektów klasy TwitterItem.