SPRAWOZDANIE

Zawansowane aplikacje WWW

Ekstrakcja danych z sieci WWW

16.12.2015

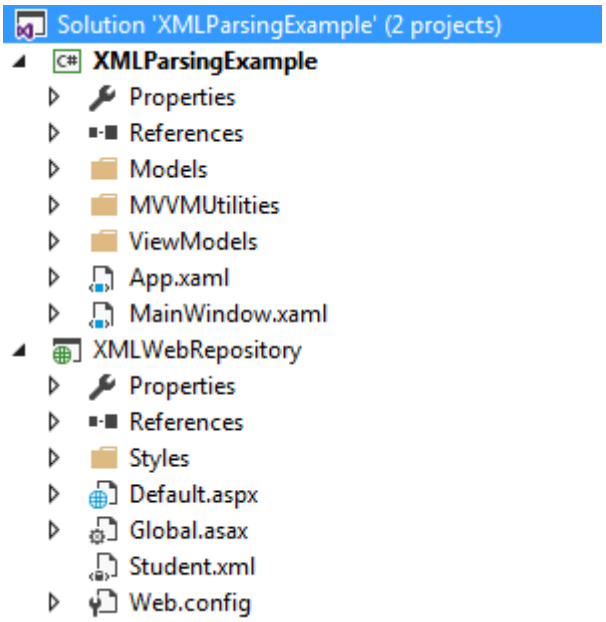
Karol Suwalski 125NCI B

https://github.com/SuwalskiKarol/orw.git

Parsowanie pliku XML

Do ekstrakcji stwórzmy dwa projekty, które zamknąłem w jednej solucji.

- `XMLWebRepository` – jest to projekt ASP.NET. Jest nam on potrzebny do przedstawienia samego pliku XML orz odpalenia go w przeglądarce. Nic Więcej.
- `XMLParsingExample` – Jest to projekt WPF. To tutj dochodzi do parsowani dokumentu. Jako iż jest to projekt wymagający od aplikacji pobrania czegoś z przeglądarki to zastosowałem wzorzec MVVM, który idealnie się sprawdza z WPF. Jest to moje pierwsze podejście do MVVM.



1. Do pracy jest nam oczywiście potrzebny sam plik XML. Od jego stworzenia zacznijmy, więc cała zabawę.

Element kodu przedstawiający same dane.

```
<Studentlist>
<Studentid="1"pkt="100"rekrutacja="15.07.2015"
opis="to ta żółta">Gumiś Sani</Student>
<Studentid="2"pkt="100"rekrutacja="16.07.2015"
opis="Najmłodszy, różowy(?)">GumiśKabi</Student>
<Studentid="3"pkt="99"rekrutacja="14.06.2015"
opis="O! to ja! gruby, zaspany i ciągle je!">GumićTami</Student>
<Studentid="4"pkt="98"rekrutacja="12.05.2015"
opis="paker z wadą wymowy">GumiśGrafi</Student>
<Studentid="5"pkt="99"rekrutacja="15.07.2015"opis=
"to ten czarny">Barack Obama</Student>
<Studentid="6"pkt="35"rekrutacja="09.09.2015"
opis="ten mądry co ciągle w książkach siedział">GumiśZami</Student>
</Studentlist>
```

2. Jak to zazwyczaj bywa przy wzorcach, zacznijmy od Modelu. Mamy tu dwa pliki

ApplicationModel.cs – W nim się znajduje się typowa reprezentacja danych.

```
publicclassStudent
    {
publicint id { get; set; }
publicstringimię { get; set; }
publicintpkt { get; set; }
publicstringrekrutacja { get; set; }
publicstringopis { get; set; }
    }

publicclassStudentsInformation
    {
publicstringSzkoła { get; set; }
publicstringWydział { get; set; }
publicList<Student>Studentlist { get; set; }

publicStudentsInformation()
        {
Szkoła = "N/A";
Wydział = "N/A";
Studentlist = newList<Student>();
        }
       }
```

XMLParser.cs - w nim przedstawione są dwie metody parsowania dokumentów xml. Klasa **XMLDocument** oraz klasa **XDocument**. Czym one się różnią? Na ogół są one bardzo podobne. XDocument jest bardziej czytelny i pozwala zaoszczędzić kilka linii kodu w porównaniu do XMLDocument.

Napisanie samego kodu w tych klasach jest bardzo proste, więc nie będę się na nim skupiał za bardzo. Podam kilka linii kodu jednej oraz drugiej klasy aby je porównać

XMLDOCUMENT

```
publicstaticStudentsInformationParseByXMLDocument()
        {
var students = newStudentsInformation();

XmlDocument doc = newXmlDocument();
doc.Load(xmlUrl);

XmlNodeGeneralInformationNode =
doc.SelectSingleNode("/StudentsInformation/GeneralInformation");
students.Szkoła =
GeneralInformationNode.SelectSingleNode("Szkoła").InnerText;
students.Wydział =
GeneralInformationNode.SelectSingleNode("Wydział").InnerText;

XmlNodeStudentListNode =
doc.SelectSingleNode("/StudentsInformation/Studentlist");
XmlNodeListStudentNodeList =
       StudentListNode.SelectNodes("Student");
```

XDOCUMENT

```
publicstaticStudentsInformationParseByXDocument()
        {
var students = newStudentsInformation();

XDocument doc = XDocument.Load(xmlUrl);
XElementgeneralElement = doc
                    .Element("StudentsInformation")
.Element("GeneralInformation");
students.Szkoła = generalElement.Element("Szkoła").Value;
       students.Wydział = generalElement.Element("Wydział").Value;
```

Już na pierwszy rzut oka widać, że XDocument jest prostszy i potrzeba mniejszej ilości kodu do jego napisania.

3. Teraz zaczną się rzeczy dla mnie nowe. Elementy związane z MVVM.

   Stwórzmy plik MVVMUtilities, zawierający dwie pomocne klasy.

   ViewModelBase–Jest podstawową klasą dla wszystkich viewmodels w aplikacji.

```
publicabstractclassViewModelBase
        : DependencyObject, INotifyPropertyChanged
    {
publiceventPropertyChangedEventHandlerPropertyChanged;

protectedvoidNotifyPropertyChanged(stringpropertyName)
        {
if (PropertyChanged != null)
            {
PropertyChanged(this, newPropertyChangedEventArgs(propertyName));
}
}
```

RelayCommand – będzie używana do implementacji commandów w viewmodels.

…

```
publicboolIsEnabled
        {
get { returnisEnabled; }
set
            {
if (value != isEnabled)
                {
isEnabled = value;
if (CanExecuteChanged != null)
                    {
CanExecuteChanged(this, EventArgs.Empty);
                    }
                }
            }
        }
publicboolCanExecute(object parameter)
        {
returnIsEnabled;
        }
publiceventEventHandlerCanExecuteChanged;
```

```
publicvoid Execute(object parameter)
{
handler();
        }
```

4. Czas na stworzenie pliku view model. Klasa ta zawiera trzy commandy i jedną publiczną właściwość

    - PropertyStudentInformationObject– służy do zachowywania informacji z parsowaniaXMLa.
    - CommandXMLDocumentLoadCommand – parsuje nasz plik używając do tego XMLDocument wywołując metodę ParseByXMLDocument, którą zapisaliśmy w klasie XMLParser.
    - CommandXDocumentLoadCommand- parsujenasz plik używając do tego XDocument wywołując metodę ParseByXMLDocument, którą zapisaliśmy w klasie XMLParser.
    - Command ClearResultCommand– czyściobiektStudentInformationObject

```
        // Commands

        publicRelayCommandClearResultCommand { get; privateset; }
        privatevoidClearResult()
                {
        StudentInformationObject = newStudentsInformation();
                }

        publicRelayCommandXMLDocumentLoadCommand { get; privateset; }
        privatevoidXMLDocumentLoad()
                {
        StudentInformationObject = XMLParsers.ParseByXMLDocument();
                }

        publicRelayCommandXDocumentLoadCommand { get; privateset; }
        privatevoidXDocumentLoad()
                {
        StudentInformationObject = XMLParsers.ParseByXDocument();
                }

privatevoidWireCommands()
        {
ClearResultCommand = newRelayCommand(ClearResult);
ClearResultCommand.IsEnabled = true;

XMLDocumentLoadCommand = newRelayCommand(XMLDocumentLoad);
XMLDocumentLoadCommand.IsEnabled = true;

XDocumentLoadCommand = newRelayCommand(XDocumentLoad);
XDocumentLoadCommand.IsEnabled = true;
        }

// Constructor
publicMainWindowViewModel()
{
InitiateState();
WireCommands();
        }
```

5. Na koniec został nam plik view czyli XAML. Mamy tu trzy przyciski, które powiązane są z commandami w model view. Datagrid natomiast jest powiązany z obiektem `StudentInformationObject`

```
                    <StackPanelGrid.Row="0" Orientation="Horizontal"
HorizontalAlignment="Right" Margin="6">
<Button Content="Parse XML z XMLDocument"
 Command="{Binding Path=XMLDocumentLoadCommand}"
 Margin="0, 0, 5, 0" />
<Button Content="Parse XML z XDocument"
 Command="{Binding Path=XDocumentLoadCommand}"
 Margin="0, 0, 5, 0" />
<Button Content="Wyczyść"
 Command="{Binding Path=ClearResultCommand}" />
       </StackPanel>
```

Po odpaleniu aplikacja odczytuje plik XML odpalony w przeglądarce oraz wrzuca dane w nim zawarte do tabeli.

Parsowanie pliku HTML

Parsowanie HTML-a nie jest prostą sprawą. HTML nie ma jednolitej składni i nie może być potraktowany jak plik XML. Napisanie swojego parsera też nie jest łatwe. Analiza HTML przy użyciu wyrażeń regularnych jest jeszcze gorszym koszmarem.

Na szczęście jak to w programowaniu bywa ktoś już dużo wcześniej opracował rozwiązanie i się z nim podzielił. Tym rozwiązaniem jest biblioteka**HTMLAgilityPack**. Wystarczy jąpobrać za pomocą NuGetsa do naszego projektu.

Używając metody Load z HTMLAgilePack możemy uzyskać obiekt HtmlDocument, który określi na zawartość strony w podobny sposób do obiektu XMLDocument.

```
privatestaticHtmlDocumentRetrieveHtml(stringWebsiteUrl)
        {
HtmlWebhw = newHtmlWeb();
returnhw.Load(WebsiteUrl);
}
```

Mając już ten obiekt operujemy na nim przy pomocy metod LINQ dokładnie tak samo jak z obiektem XMLDocument. Poniższe wyrażenie LINQ wyrywa zawartość tekstową z dokumentu HTML.

Później zawartość tekstowa jest rozbijana na pojedyncze słowa. Przy pomocy odpowiedniego wyrażenia regularnego upewniam się ,że słowa nie zawierają w sobie niedozwolonych znaków.

```
privatevoidbutton_Click(objectsender, RoutedEventArgs e)
{
            d = RetrieveHtml(textBox.Text);

var text = d.DocumentNode.Descendants()
                .Where(x =>x.NodeType == HtmlNodeType.Text&&x.InnerText.Trim().Length
> 10
&& !(x.InnerHtml.Trim().Contains("function")) &&
!(x.InnerHtml.Trim().Contains("![CDATA["))
&& !(x.InnerHtml.Trim().Contains("var")) && !(x.InnerHtml.Trim().Contains("http://"))
&& !(x.InnerHtml.Trim().Contains("SyntaxHighlighter"))
&& !(x.InnerHtml.Trim().Contains("class"))
                    )
                .Select(x =>x.InnerText.Trim());


foreach (var a intext.Select(var2 => var2.Split(' ')))
            {
for (inti = 0; i<a.Length; i++)
{
a[i] = Regex.Replace(a[i], "[^a-zA-Zęńśćąóżźł]+", "");

if (a[i] != ""&& a[i].Length >= 5)
                    {
list.Add(a[i]);

                    }
                }
            }

listBox.ItemsSource = list;

}
```

Jeśli wyraz jest dłuższy bądź równy niż 5 znaków jest on wtedy dodawany do listy.