Vyšší odborná škola a Střední škola Varnsdorf,

příspěvková organizace

Crock

aplikace na generování prezentací

Vypracoval: Zdeněk Bárta

Třída: I4B

Konzultant: Škoda Jan Mgr. Bc.

Oponent: Hroza Pavel Ing.

Školní rok: 2013/2014

Datum zadání: 7. ledna 2014

Datum odevzdání: 7. března 2014

Čestné prohlášení

Já Zdeněk Bárta, datum narození 13.1.1995

bydliště Rumburk, Poštovní 272/1,

tímto čestně prohlašuji,

že jsem praktickou maturitní zkoušku zpracoval samostatně.

V Rumburku dne 3.2.2014

Anotace

Tato maturitní práce se zabývá návrhem a vytvořením programu v jazyce c#, který bude stahovat zdrojové kódy webových stránek Wikipedia, parsovat je a nadéle z vyparsovaných dat které se zkombinují s daty z nastavení aplikace bude vytvářet automaticky prezentace. Celý tento cyklus bude vyžadovat co nejmenší účast uživatele a vytváření prezentací bude vice jak 1000x rychlejší než kdyby to vytvářel sám člověk.

Abstract

This graduation project deals with the desing and creation of a program in c#, which will download the source code of web site Wikipedia, then parse it. Data be combined with application settings, then start the whole process of creating presentations. This whole cycle will require as little user participation and creating presentations will include more than 1,000 times faster than if it created the man himself.

Obsah

1 Úvod 5
1.1 Očekávané znalosti 5
2 HTML a XML 6
2.1 HTML 6
2.2 XML 6
2.3 Rozložení stránky Wikipedia 7
2.4 Výběr parseru 7
3 Popis aplikace 8
3.1 Obecný návrh 8
3.2 Rozložení souborů 9
3.3 Řešení jazyků v aplikaci 12
3.4 Nastavení aplikace 13
3.5 Formuláře 14
3.5.1 Message formulář 14
3.5.2 Settings formulář 14
3.5.3 Update formulář 14
3.5.4 User formulář 14
3.5.5 Main formulář 14
3.6 Parser 15
3.7 Třída na generování prezentace 18
4 Závěr 19
4.1 Spokojenost s prací 19
4.2 Grafika a testování 19
4.3 Představy do budoucna 19
5 Seznam použité literatury 20

1 Úvod

Do vytváření prezentací musí vkládat hodně času nejej studenti ale také zaměstnanci různých firem. Většinou jsou to nudné informace, které jsou nejčastěji čerpány z pár serverů s informacemi. Hromady textu které se k nim dostane, většinou kopírují a poté je čtou z projektorů nebo vytištěných svazků papírů. Mým cílem je tedy zjednodušit a hlavně zrychlit tuto práci a dát lidem program co jim dá více volného času na věci, které je opravdu zajímají. Zaměřil jsem se na server wikipedia.org, protože se umístil na 6. místě nejnavštěvovanějších webových stránek za rok 2013 a před sebou měl jen vyhledávače a sociální sítě.[1]

1.1 Očekávané znalosti

Ke čtení zdrojových kódů v této práci jsou nutné alespoň základní znalosti objektového programování v jazyce C#, nebo nějakém jiném objektovém orientovaném jazyce.

K vývoji tohoto programu sem využívam framework .Net 4.5 který je už obsažen v nových verzích Microsoft Windows, nebo se dá doinstalovat.[2]

[1] http://sprava-dokumentu.cz/magazin/top/1000-nejnavstevovanejsich-webu-na-svete
[2] http://www.microsoft.com/cs-cz/download/details.aspx?id=30653

2 HTML a XML

2.1 HTML

Jazyk HTML je charakterizován množinou značek (tagů) a jejich vlastností (atributů) definovaných pro danou verzi. Tedy jazyk HTML je určen pouze pro webové stránky, protože mají předem definovaný styl zápisu.

Vlasní tagy není možné přidávat protože uživatel nemá možnost upravit browser tak aby to podporoval.

Základní konstrukce webové stránky:

```
<!DOCTYPE html>
<html>
 <head>
  <title> </title>
 </head>
 <body>
 </body>
</html>
```

2.2 XML

Používá se pro serializaci dat. A je velmi hodně používán k ukládání dat, jako je například nastavení, ale I jiné. Například předávání dat mezi aplikacemi. Vezměme si například Microsoft Power Point 2002 a 2012 ve kterém můžeme souboru uložit v XML, power point 2012 má jistě mnoho věcí navíc, které nepůjdou ve starší verzi vykreslit ale většina prezentace se otevře I ve starší verzi. Je to vlastně převod objektu a jejich vlastností do xml které si můžeme normálně otevřít v textovém editoru. V souborech na CD je přiložen textový soubor s xml kódem pro jednoduchou jedno slidovou prezentaci.

Xml nemá dané přesnné jména elementů a jeho vlastností jako Html, záléží na tom jaký si napíšeme program a jestli to bude podporovat.

Obecný příklad Xml:

```
<jméno_elementu vlastnost=“hodnota“ ....>hodnota</jméno_elementu>
```

2.3 Rozložení stránky Wikipedia

Všechny stránky na wikipedii kde se popisuje nějaké téma maji stejné rozložení. Na CD je přiložen celý zdrojový kód stránky s tématem, po odstranění všech nepotřebných dat jako načítání css souboru nebo skriptu vyjde jednoduchá kostra celé stránky.

Na začátku máme jen jeden nadpis typu h1:

```
<h1 id="firstHeading" class="firstHeading" lang="cs"><span
dir="auto">Včela</span></h1>
```

Dále úsek stránky ve kterém už nejsou žadné menu ani další upozornění.

`<div id="mw-content-text" lang="cs" dir="ltr" class="mw-content-ltr">` tady máme zbytek stránky kterou potřebujeme `</div>`

Všechny texty na stránce jsou v tagu <p>, nebo <li>, takže budeme vybírat pouze hodnoty v těchto úsekách. Také vynecháme všechny tabulky, protože v prostředí power pointu není snadné generovat tabulky a dost by nám to zkomlikovalo kód.

2.4 Výběr parseru

HtmlAgilityPack[3] je výborná knihovna na parsování xml kódu. Výborně napsaná ikdyž může být velmi těžké pochopit ze začátku jak s ní pracovat. Avšak pokud máte hodně času můžete si projít všechny zdrojové kódy které jsou poskytnuty na webové stránce.

Není to pouze na parsování HTML kódu, ale na parsování elementů které si zadáme, takže si v programu navolíme tagy, které obsahují nějaké texty na stránce wikipedia.

Licence – Microsoft Public License (Ms- PL)

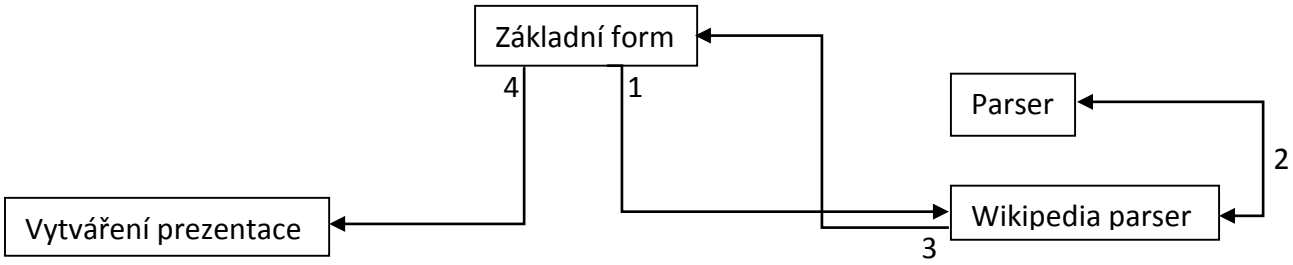
Tato licence se řídí používání doprovodného softwaru . Používáte-li software , můžete přijmout tuto licenci. Pokud nesouhlasíte licenci, nepoužívejte software.

[3] http://htmlagilitypack.codeplex.com/

3 Popis aplikace

3.1 Obecný návrh

Návrh aplikace sem zvolil tak aby byla co nejpřehlednější v kódu a aby se s ní dalo pěkně pracovat. Hlavní problém byla možnost rozvoje aplikace ať už o nové parsery nebo rozvoj vzhledu. Třídy a metody jsou co nejjednodduší aby byla možnost vidět co se opravdu děje. Největší problémy nastali s parsery a řešení jazyků a nastavení. Parsery jsou řešen tak, že až s další verzí přijdou nové budou snadno přepsány do xml kódu aby si uživatel mohl jednoduše vytvářet svoje. Bohužel teď je to řešeno pouze v kódu.



1. Základní form získá url adresu nějaké webové stránky, předá je vybranému parseru. V našem případě máme zatím poze jeden parser.

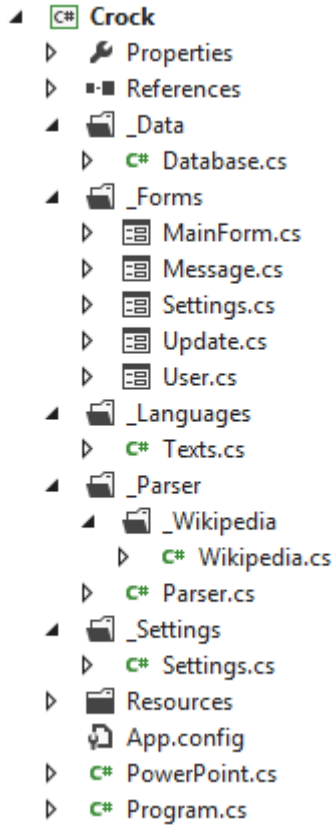
2. Wikipedia parser parsuje stránku za pomoci metod, které jsou v třídě parser ze které dědí.

3. Wikipedia parser předá list objektů zpět do formu.

4. Základní form je přepošle k vytvoření prezentace spolu s údaji o jméně uživatele a dnešnímu datumu.

5. Není vyznačena, ale jsou to ostatní obejkty které se využívají podle potřeby.

3.2 Rozložení souborů



Program začína v souboru Program.cs, ten automaticky jen otevře nové okno. Aplikace však obsahuje nastavení, takže tato metoda je mírně upravená.

```
[STAThread]
public static void Main()
{
	Application.EnableVisualStyles();
	Application.Run(new MainForm());
	Setting.SaveAll();
	Application.Exit();
}
```

`Application.EnableVisualStyles()` - Pokud nevyužíváme nějaké jiné styly aplikací tak tam tato metoda ani nemusí být, ale je lepší ji pro jistotu psát.

`Application.Run(new MainForm())` - Aplikace otevře nové okno (MainFrom), program pokračuje poté co se ve třídě MainFrom zavolá metoda na ukončení tohoto formu(třeba `this.Close()`).
`Setting.SaveAll()` - je metoda která nám po zavření hlavního okna uloží všechna změněná data nastavení do uživatelské složky (AppData).
`Application.Exit()` - Ukončí aplikaci, aplikace by se měla ukončit sama ale nastává zde pár problémů s ukládáním nastavení tak je to pojištěné tímto příkazem.

Složka _Data obsahuje jen třídu Database, která je tu jen pro demonstraci že se v další verzi mohou brát data z nějaké databáze a další rozšíření a vylepšení aplikace budou napsány v závěru této práce.

Složka _Forms obsahuje všechny formuláře které tato aplikace může využívat.

Složka _Languages obsahuje pouze třídu Texts která se stará o načítání frází z textového souboru.

Složka _Parser obsahuje jednu hlavní třídu Parser z které dědí třídy v podsložkách dalších možných parserů. Další parsery mohou být přidávány s dalšími verzemi. A znamená to že by bylo možné prezentace generovat i z jiných zdrojů než je wikipedia.

Složka _Settings obsahuje jeden soubor s více třídami, kde všechny dědí z jedné hlavní. Defaultní nastavení se takto generuje když si vytvoříte nový projekt a přidáte soubor settings.setting, akorát toto je trochu upravená verze aby se to dalo lépe využívat v aplikaci.

Poslední tu je třída PowerPoint, která je klíčová k otevření microsoft office a přidávání různých komponent.

Každá takováto složka má nový namespace (skupina objektů), je to nadále přehledné a nemusíme deklarovat všechno když to vůbec nepotřebujeme.

Příklad vám to nejspíše upřesní.

```
using System.Threading.Tasks;
using System.Windows.Forms;
using Crock._Forms;
using Crock._Settings;
using Crock._Languages;
```

Každý Form v aplikaci má 3 základní části. Náhled Formu jak asi každý zná. Poté zdrojový kód s částečnou třídou formu a s konstruktorem této třídy. Už tu máme částečnou třídu, takže někde musí být i jiné části té třídy. Druhá půlka se nachází v samostatném souboru form.designer kam se automaticky generuje kód pro komponenty a form který upravujeme v náhledu formu.

V příkladu vytvoříme v náhledu jednoduché tlačítko. A pak hned můžeme v kódu používat jen `button1.příkaz` a přidávat mu vlastnosti nebo nějaké akce. A abychom to mohli tak používat tak nám visual studio vygeneruje automaticky kód pro tu komponentu.

```
this.button1 = new System.Windows.Forms.Button();
```

vytváří novou instanci třídy Button

```
this.button1.Location = new System.Drawing.Point(118, 98);
```

určuje pozici na formu v pixelech

```
this.button1.Name = "Presentation";
```

určuje jméno tlačítka

```
this.button1.Size = new System.Drawing.Size(75, 23);
```

určuje velikost tlačítka v pixelech

```
this.button1.TabIndex = 2;
```

určuje index kdy bude aktivní po stisknuti tlačitka tabulátor

```
this.button1.Text = "button1";
```

text tlačítkam zobrazuje se uprostřed

```
this.button1.UseVisualStyleBackColor = true;
```

umožnuje změnu barev a grafických efektů

a mimo metodu musi být privátní proměnná, kterou potom využíváme v programu a která se inicializuje kódem který je nahoře:

```
private System.Windows.Forms.Button button1;
```

3.3 Řešení jazyků v aplikaci

Hlavním cílem bylo nějak elegantně vyřešit problém s použitím tohoto v kódu. Tedy v uživatelské složce AppData/Roaming/Crock máme složku Languages do které si můžeme přidat kolik souborů chceme a sami si je i přeložit. Máme tu češtinu a angličtinu, ikdyž je angličtina prázdná tak je možné ji používat protože je v aplikaci nastavená jako defaultní a z příkladu uvidíte důvod proč tam nemusí být nic napsané.

Třída Texts která se stará o překlady v celé aplikaci má zjednodušeně tyto části:

Vlastnost Language, která při startu aplikace veme z nastavení hodnotu posledního používaného jazyku.

Dictionary<string, string> texts, což si můžeme přirovnat k poli, protože pole v c# nemůže mít string jako index tak tu máme tuto třídu. Do té se nám ukládá index => hodnota, index je anglicke slovo a hodnota je nějaký překlad který je načítán ze souboru.

Metodu Load která načte do pole hodnoty ze souboru.

A nakonec nejlepší část této třídy, což je indexer který se požívá k přístupu k zapouzdřené kolekci v našem případě dictionary. Na příkladu si to vysvětlíme jak by to vypadalo bez a s indexerem.

V nějakém formu vytvoříme instanci třídy texts

`Texts text = new Texts();` - to nám vytvoří novou instanci třídy texts a načte nám to hodnoty ze souboru.

Normálním přístupem k poli v naší třídě by bylo přes metodu.

`Komponenta.Text = text.getValue("login");` - a v této metodě by musel být kód na vyhledávání v poli a vrácení nějaké hodnoty.

S indexerem to v kódu využíváme takto:

```
Komponenta.Text = text["Create"];
```

A v indexeru máme pouze podmínku, že pokud nenajde shodu vrátí index který mu byl předán, tím máme vyřešen problém že pokud se načte jen půlka souboru a nastane nějaká chyba zbytek aplikace bude v angličtině protože v poli takový záznam nebude existovat.

```
public string this[string text]
{
	get
	{
		if (this.texts.ContainsKey(text))
			return this.texts[text];
		else
			return text;
	}
}
```

3.4 Nastavení aplikace

K vytvoření nastavení se vrátíme zase ke Xml a budeme ho využívat v plné síle. Nebude využívané žádné šifrování a nastavení se ukládá do uživatelské složky AppData/Roaming/Crock.

Třída Setting využívá už předpřipravené generování xml kódu, avšak třídy jsou znatelně upravené, aby jsme nastavení mohli v aplikaci využívat co nejlépe. V aplikaci proto máme 2 druhy nastavení. Nastavení uživatele (User setting) a nastavení aplikace (App setting).

Příklad záznamu v Xml uloženého v souboru:

```
<setting name="Language" serializeAs="String">

                <value>cz</value>

 </setting>
```

A k tomu vytvořená vlastnost ve zdrojovém kódu:

```
public string Language
{
       get { return (string)this["Language"]; }
       set { this["Language"] = value; }
}
```

V kódu se k hodnotě "cz" dostaneme velice snadno a to tak že napíšeme Setting.App.Language. Zároveň nám tento xml kód sděluje mnoho věcí aby bylo jasné co za data vlastně dostáváme.

Vlastnost "name" nám tedy určuje jméno hodnoty, ta se pak používá i v kódu.

Vlastnost "serializeAs" nám určuje datový typ, kterých muže být hrozná spousta. Například můžeme do hodnoty uložit barvu v rgb složení (c# podporuje i jiný druhy zápisu):

```
<setting name="BgColor" serializeAs="Color">

                <value>100, 150, 255</value>

 </setting>
```

V kódu k ní potom můžeme přistupovat jako k barvě ikdyž je zde uložená ve formě textu, protože se automaticky přetypuje na takový typ, který je zde určen.

V elementu <value></value> je uložená hodnota ve formátu string. Všechny datové typy mají určeno jak mají vypadat ve formátu string. V příkladu barva byla 100, 150, 255 (v pořadí červená, zelená, modrá neboli RGB).

3.5 Formuláře

3.5.1 Message formulář

Tento formulář byl vytvořen jako náhrada klasických MessageBoxů, ale není v této aplikaci doposud využíván.

3.5.2 Settings formulář

Formulář sloužící ke změně většiny nastavení aplikace, například ke změně jména uživatele, konce prezentace, nebo výběru defaultní šablony prezentace. Tento formulář se dá otevřít přes profil uživatele nebo přes menu na hlavním formuláři.

3.5.3 Update formulář

Pokud by byla funkční třída na připojení k vnější databázi a zjistilo by se, že je k dospozici novější verze, otevřel by se tento formulář a upozoornil na nové možnosti a odkázal by na určenou webovou stránku.

3.5.4 User formulář

Z názvu to není patrné, ale je to formulář obsahující profil uživatele. K našim potřebám postačilo jen jméno tak se na tomto formuláři zobrazuje pouze jméno uživatele, které se bere z nastavení.

3.5.5 Main formulář

Nebo-li hlavní formulář, otevře se ihned po startu aplikace a je zde nejvíce možnosti pro uživatele. Stará se o získání dat (url adresy, nebo cesty souboru v pc) a předání je do parseru, získaná data předává nadále do třídy PowerPoint, aby mohla být z těchto dat vytvořená prezentace.

Na tomto formuláři je možne jen zadat téma prezentace kterou hledáme a aplikace si podle nastavení jazyka složí vlastní url adresu, kterou následně otevře v defautním browseru a pokud uživatel toto téma hledal tak mu už stačí jen kliknout na tlačítko a prezentaci má vytvořenou.

Nakonec na tomto formu máme menu ve kterém se dá změnit jazyk aplikace, otevřít webouwou stránku wikipedia zobrazit profil uživatele a mnoho dalšího.

Jazyky přidávané do složky AppData/Roaming/Crock/Languages se po restartu formuláře v menu obnovují takže je možné si jazyků nastavit kolik chceme.

3.6 Parser

Než začneme popisovat třídu parser musíme si popsat strukturu textu kterou dostaneme s html dokumentu. Jedna hodnota z tagu <p> a <li> bude vždy jedna struktura uložená v listu itemů který se poté bude předávat jiné třídě pro vygenerování prezentace.

Jak vlastně bude fungovat třída Parser. Metody v této třídě jsou určeny a mohou je používat pouze třídy které z této třídy dědí. Jediná vlastnost ke které bude přístup z jiných tříd je vlastnots `ArticleName` což je jméno článku, ze kterého se vytváří prezentace. Dále tu máme 2 metody na načítání html kódu jedna stahuje kód z určené url adresy a druhá ho načíta ze souboru v počítači. Nakonec je tu pár metod na editaci, odstranění vybírání dat z html kódu a to proto aby byl kód jednoduší v třídach které z této třídy dědí.

Teď se konečně dostáváme k „opravdovému“ parseru který se nacházii ve složce Wikipedia. Třída Wikipedia která dědí z třídy Parser vykonává většinu práce. Takže nevoláme třídu Parser ale třídu Wikipedia která z Parseru dědí.

V tříde wikipedia máme enum všech tagů které budeme vybírat(ty které obsahují nějáky text).

```
public enum Tags
{
	h2,
	p,
	li
}
```

Popíšeme si tedy metodu `Parse` která obsahuje pár příkazů na odstranění přebytečných věcí z html kódu a pak cyklus který nám naplní pole hodnot.

```
document.LoadHtml(this.htmlCode);
this.ArticleName = this.GetOneTag("h1").InnerText;
this.RemoveTags("table");
this.LoadHtmlIn("div[@id='mw-content-text']");
string[] tagsArray = Enum.GetNames(typeof(Tags));
```

`document` - je třída v naší knihovně HtmlAgilityPack do které nahrajeme html kód.

Hned potom si do vlastnosti `ArticleName` uložíme nadpis článku který vybereme přes metodu `GetOneTag` která je ve třídě Parser a která nám vrací třídu `HtmlNode` a z ní si vybereme pouze vlastnost `InnerText` což je text mezi tagy <h1></h1>. na stránce se tag <h1> využívá pouze na nadpisy takže nadpis sme vyparsovali.

Metodu `RemoveTags` která je ve třídě parser použijeme na smazání všecho co je mezi tagy <table></table>

Metoda `LoadHtmlIn` je zase v třídě Parser a z jedné části dělá to co první krok `document.LoadHtml` a z druhé části vybere všechno co je mezi tagy a to nahraje do proměnné. Jak už bylo řečeno mezi tímto tagem je célá html stránka co obsahuje nějaké užitečné data, které budeme vybírat.

Nakonec si do `tagsArray` uložíme tagy z enumu ve formátu string aby sme je mohli projet cyklem a vybrat všechny data.

```
foreach (string tag in tagsArray)
{
	foreach (HtmlNode node in this.GetItems(tag))
	{
		this.Items.Add(new ppItem()
		{
			Index = node.Line,
			Text = tag == "h2" ? EditH2Tag(node.InnerText)
					: node.InnerText,
			Type = tag == "h2" ? ppItemType.Title
					: ppItemType.Text
		});
	}
}
```

Máme tu vnořený cyklus, takže si popíšeme co dělá první. Z příkladu nahoře bylo jasně zřejmé že tento cyklus tedy bude projíždět jen 3x protože máme jen 3 prvky v enumu tedy 3 prvky v poli.

Druhý cyklus projíždí třídu `HtmlNode` v `HtmlNodeCollection` co nám vrací metoda `this.GetItems(tag))`, která je v tříde Parser. Ta očekává jako parametr tag ve stringu například <p>(bez závorek) a vrací kolekci všech tagů co najde v dokumentu. Rychlost tohoto cyklu je tedy závislá na velikosti článku, čím více tagů tím více opakování.

Pak si v tomto cyklu přidáváme struktury `ppItem` (power point item) do listu, který je zas ve třídě Parser.

```
public struct ppItem
{
	public string Text { get; set; }
	public int Index { get; set; }
	public ppItemType Type { get; set; }
}
```

Struktura `ppItem` obsahuje 3 vlastnosti:

`Text` – je text mezi tagy, tedy ten co vybíráme

`Index` – je pozice toho textu v dokumentu. Pozici potřebujeme protože do listu přidáváme tagy podle jména. Takže se nám zpřeházejí a nejdřív budeme mít všechny <a>, <li> a nakonec <p> tagy. Nakonec budeme muset ještě toto pole podle indexu totiž srovnat.

`Type` - je enum typů jaký bude moct mít tag v prezentaci.

```
public enum ppItemType
{
	Title,
	Text,
	Image,
	Table,
	Chart
}
```

V cyklu přídáváme tyto struktury zjednodušeně aby jsme nemuseli psát celý zapis.

```
new ppItem()
{
	inicializace vlastností
}
```

Nastává nám zde jeden menší problém a to že v podnadpisech kapitol máme za textem nějaké věci které nechceme. Jako třeba „Internetový vyhledávač [editovat | editovat zdroj]“ ale na každé stránce ať už ruské nebo české wikipedii je tento přebytečný obsah v hranatých závorkách tak si jednoduše napíšeme regulerní výraz.

```
private string EditH2Tag(string text)
{
	Regex regex = new Regex(string.Format("\\[.*?\\]"));
	return regex.Replace(text, string.Empty);
}
```

A nakonec určíme typ pro tag, na wikipedii se spokojíme se 2 druhy typu a to pro tag <h2> Title(Nadpis) a pro <li> a <p> Text.

3.7 Třída na generování prezentace

Jako poslední bod tu máme třídu PowerPoint, která obsahuje jen pár důležitých metod.

První RunPowerPoint, která nám otevře nové okno aplikace Microsoft Power Point pokud je nainstalovaná nějaká verze na počítači.

Další metoda Create, která očekává jako parametr list itemů (list co obsahuje struktury s vyparsovanými texty), jelikož máme 2 druhy otevření nové prezentace tak máme taky 2 druhy zápisu v kódu. Jeden druh nám otevře šablonu a ten druhý pouze vytvoří nový prázdný dokument. Takože je tu podmínka jestli máme zvolenou šablonu v nastavení.

Potom tu máme 3 metody na vytváření 3 druhů slidů. Uvodní stránka která z nastavení očekává jméno a příjmení uživatele, ke kterému si přidá jméno článku a datum.

Obsah prezentace ma jeden druh, takže metoda očekává nejaký nadpis a text který bude na slide přidán.

A pokud jsme si v nastavení napsali nějaký závěr prezentace tak bude zavolána poslední metoda na přidání závěru prezentace.

Nakonec uprostřed toho všeho máme cyklus který tyto metody volá podle toho jestli zrovna v listu narazil na nadpis nebo text. Pokud text je příliš dlouhý vytvoří nový slide se stejným nadpisem a pokračuje dokud není celé tělo prezentace vytvořeno.

Příklad na přidání slidu do prezentace:

```
private void AddSlide(string title, string text)
{
	if (title == "" || text == "")
		return;
	this.pptSlide = this.pptSlides.Add(pptSlides.Count + 1,
	ppt.PpSlideLayout.ppLayoutText);
	this.pptTextRange = this.pptSlide.Shapes[1].TextFrame.TextRange;
	this.pptTextRange.Text = title;
	this.pptTextRange.Font.Name = "Comic Sans MS";
	this.pptTextRange.Font.Size = 48;

	this.pptTextRange = this.pptSlide.Shapes[2].TextFrame.TextRange;
	this.pptTextRange.Text = text;
	this.pptTextRange.Font.Name = "Comic Sans MS";
	this.pptTextRange.Font.Size = 32;
}
```

Jak je z příkladu vidět je zde nastaven jeden druh písma pro celou prezentaci a jelikož se prezentace vytváří generováním Xml kódu tak tu máme spoustu objektu které můžeme využívat a přidávat do prezentace různé komponenty.

4 Závěr

4.1 Spokojenost s prací

Tato práce byla velmi dobře vymyšlená a myslím si že by se I uplatnila a lidé by ji používali.

Vlastně ani nevím, kolik sem do této práce dal času, ale k dokonalému programu co bych programoval sám bych potřeboval nejméně další půlrok. Je velká škoda že ten půlrok nemám, protože bych mohl zkoušet další nové nápady a zdokonalovat se v programování a aplikaci vylepšit až na úroveň kdy by začla být úspěšná.

4.2 Grafika a testování

Jelikož nejsem grafik ani tester tak sem testoval jen základní možnosti, které by mohl normální uživatel udělat. Pro prográmátory to můžu to přirovnat k parsování čísla. Ošetřil sem že text, který má být číslo int, je číslo, ale neošetřil sem jestli je v platném rozsahu. Takže z toho plyne, že je zde určitě mnoho neošetřených chyb a vyjímek a doufám že nebudou vidět.

4.3 Představy do budoucna

Tento program je jen jednoduchou kostrou toho co by v budoucnu mohl být. Mohl by se rozvíjet ať už o podpory nových parserů až po celý nový grafický návrh.

5 Seznam použité literatury

Mareš, Amadeo. 2011. *1001 tipů a triků pro C# 2010.* s.l. : Computer Press, 2011. 9788025132500.

Nagel, Christian a kolektiv. 2009. *C# 2008 Programujeme profesionálně.* Computer Press, 2009. 80-251-2401-7.

Sharp, John. 2010. *Microsoft Visual C# 2010 - Krok za krokem.* s.l. : Computer Press, 2010. 9788025131473.