FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO, BRATISLAVA

INFORMAČNÝ SYSTÉM PRE RIADENIE A VYHODNOCOVANIE SÚŤAŽE

2010 Jozef Čechovský

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO, BRATISLAVA

INFORMAČNÝ SYSTÉM PRE RIADENIE A VYHODNOCOVANIE SÚŤAŽE

(Bakalárska práca)

Študijný program : aplikovaná informatika

Študijný odbor: 9.2.9. aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2010

Autor: Jozef Čechovský

Zadanie

Prehľad vhodných technológií platformy .NET, špecifikácia, návrh a implementácia systému na registráciu tímov, plánovanie turnajov, vyhodnocovanie a prezentovanie výsledkov vrátane internetového portálu súťaže Robocup Junior Slovensko.

Čestne prehlasujem, že túto bakalársku prácu som vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Ďakujem všetkým, ktorí ma podporovali a pomáhali mi vytvoriť túto prácu. Konkrétne Mgr. Pavlovi Petrovičovi PhD. za jeho vedenie, pripomienky a čas, ktorý sa mi pri vypracovávaní práce venoval.

Jozef Čechovský

Abstrakt

Čechovský Jozef: Informačný systém pre riadenie a vyhodnocovanie súťaže. Bakalárska práca, Univerzita Komenského v Bratislave. Fakulta Matematiky, Fyziky a Informatiky. Vedúci záverečnej práce: Mgr. Pavel Petrovič PhD., Bratislava 2010.

Bakalárska práca sa venuje popisu technológií platformy .NET a špecifikácii, návrhu a implementácii systému pre riadenie a vyhodnocovanie súťaže Robocup Junior Slovensko. Teoretická časť práce obsahuje popis základných vlastností, výhod a nevýhod technológií Windows Presentation Foundation, ADO.NET Entity Framework a architektúry aplikácií Model – View – ViewModel. Ďalšia časť práce opisuje špecifikáciu, návrh a implementáciu systému, ktorý je určený na registráciu tímov, plánovanie turnajov, vyhodnocovanie a prezentovanie výsledkov súťaže Robocup Junior.

Kľúčové slová:
Windows Presentation Foundation, ADO.NET Entity Framework, Model–View–ViewModel, Robocup Junior, informačný systém pre riadenie a vyhodnocovanie súťaže

Abstract

Čechovský Jozef: Information system for competition regulating and result evaluating. Bachelor work, Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics. Supervisor: Mgr. Pavel Petrovič PhD., Bratislava 2010.

Bachelor work deals with description of .NET technologies and with specification, design and implementation of system for regulating and result evaluating of competition Robocup Junior Slovakia. Theoretical part describes the base features, advantages and disadvantages of technologies Windows Presentation Foundation, ADO.NET Entity Framework and software architecture Model – View – ViewModel. The practical part of work comprises specification, design and implementation of system, which is indented for team's registration, planning tournament and evaluating and presentation of results of competition Robocup Junior.

Key words:
Windows Presentation Foundation, ADO.NET Entity Framework, Model–View–ViewModel, Robocup Junior, system for competition regulating and result evaluating

Obsah

1 Úvod

1.1 Motivácia a ciele práce

Každoročne sa koná súťaž stredných a základných škôl v programovaní robotov RoboCup Jurnior Slovensko. Súťaž prebieha tri dni, zúčastňuje sa jej približne štyridsať tímov, súťaží sa vo viacerých súťažných a vekových kategóriách. Je nutnosť evidovať tieto tímy, zostaviť časové plány súťaže a taktiež vyhodnotiť poradie tímov v jednotlivých kategóriách. Organizácia je náročná najmä z dôvodu, že všetka administratívna práca sa robí ručne. Pri zväčšovaní počtu tímov na súťaži koordinácia súťaže zaberá organizátorom stále viac času a miestami sa stáva nezvládnuteľná. Z tohto dôvodu by im veľmi pomohol systém, ktorý by im s organizovaním súťaží pomohol a odbremenil ich od množstva práce. V súčasnosti poznáme množstvo technológií, ktoré by bolo možné použiť pri implementácií tohto systému. Použitie vhodných technológií a postupov môže zjednodušiť prácu programátorovi ako aj spríjemniť používanie systému koncovým užívateľom.

Hlavným cieľom práce je špecifikácia, návrh a implementácia kompletného systému na registráciu tímov, evidenciu tímov, plánovanie turnajov a vyhodnocovanie ich výsledkov. Ďalším cieľom je preskúmať vhodné .NET technológie, architektúru využívajúcu tieto technológie a použiť ich pri implementácii systému.

2 Východiská

Vývoj aplikácií v súčasnosti sa stále mení. Prichádzajú nové technológie, postupy a návrhové vzory. Vo svete platformy .NET sa objavili novinky, ktoré sa postupne udomácňujú medzi programátormi a vývojármi. .NET je framework, ktorý je súčasťou operačného systému Windows. Prostredníctvom tohto frameworku vytvárame aplikácie a webové servisy [11].

2.1 Windows Presentation Foundation

Windows Presentation Foundation (WPF) je ďalšia generácia prezentačného systému na vytváranie klient aplikácií s bohatými vizuálnymi možnosťami [14]. Pomocou WPF sa dá vytvoriť samostatná alebo webová aplikácia s použitím množstva grafických prvkov. WPF je najväčšia zmena vo Windows užívateľských rozhraniach od roku 1995.

Jadro WPF je engine, ktorý je nezávislý na rozlíšení a je založený na vektorovej grafike. Výhoda je v tom, že tento systém naplno využíva možnosti grafických hardvérových prvkov. Základné WPF je rozšírené veľkou sadou doplnkov, ktoré nám uľahčujú vývoj aplikácií. Tieto doplnky obsahujú Extensible Application Markup Language (XAML), vizuálne komponenty, dátové väzby (data binding), grafické layouty, 2-D, 3-D grafiku, animácie, štýly, šablóny, dokumenty, médiá a text.

2.1.1 Extensible Application Markup Language

Extensible Application Markup Language (XAML) je značkovací jazyk určený na popísanie WPF užívateľského rozhrania. Ako používame HTML na písanie webových stránok, tak sa používa XAML na popísanie Windows komponentov a vzhľadu celých aplikácií. Pri písaní užívateľského rozhrania nie je nutné ovládať XAML. Elementy v XAML dokumente sú všetky namapované na inštancie .NET tried, čiže komponenty sa dajú napísať aj priamo v kóde. Avšak sila tohto jazyka je práve v tom, že pri vývoji aplikácií je vizuálna časť oddelená od logickej časti, čo ma veľké uplatnenie pri tímovom vývoji. Pomocou tohto jazyka sa dajú definovať užívateľské rozhrania s mnohými grafickými prvkami a doplnkami

s menším úsilím ako to bolo možné doteraz. Zložitejšie efekty sú skôr určené pre grafikov a dizajnérov, ale aj bežný programátor alebo vývojár si môže ľahko nadefinovať užívateľské rozhranie klasického vzhľadu s malými doplnkami. Na nasledujúcom príklade je ukážka použitia XAML jazyka na popísanie vzhľadu okna vo Windows. Na ukážke je konkrétne popis okna s danou výškou a šírkou a s komponentmi TextBox a Button, ktoré sú umiestnené v paneli StackPanel. Jednotlivými atribútmi ako Title, Name, Width, Height, Margin sú nastavené vlastnosti okna alebo komponentov. V popise tlačidla je definovaná udalosť Click, ktorá po kliknutí tlačidla spustí metódu Confirm.

```
<Window x:Class="WpfApplication.Example"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Example" Height="300" Width="300">
  <StackPanel Margin="5">
    <TextBox Name="textBox_Name" Width="100" Height="30" />
    <Button Name="button_Confirm" Width="50" Height="30" Click="Confirm"/>
  </StackPanel>
</Window>
```

2.1.2 Väzby vo WPF

Väzby poskytujú pre aplikácie jednoduchý spôsob zobraziť dáta a pracovať s nimi. Elementy užívateľského rozhrania môžu byť naviazané na zdroje dát vo forme objektov a ich verejných premenných, alebo je možné pomocou väzieb naviazať na udalosti elementov akcie, ktoré sa majú vykonať. Väzby môžu byť:

- Dátové väzby (Data Binding)
- Príkazové väzby (Command Binding)

2.1.2.1 Data Binding

Funkcionalita dátových väzieb vo WPF má veľa výhod oproti tradičným modelom, zahŕňajúcich široké spektrum vlastností, ktoré podporujú flexibilné prezentovanie dát a oddelenie logiky aplikácie od vizuálnej časti. Ak má Data Binding správne nastavenia a dátový zdroj má definované nejaké oznámenia (notifications) o zmene dát, tak pri zmene dát v dátovom zdroji sa automaticky zmenia aj dáta zobrazené prostredníctvom komponentu

v užívateľskom rozhraní. Tak isto platí aj opačný smer dátovej väzby, čiže po zmene vlastností komponentu napríklad užívateľom sa zmenia dáta aj v dátovom zdroji. Záleží však na nastavení módu väzby, ktoré rozlišujeme štyri.

- OneWay - pri zmene zdroja robia automatické zmeny iba vlastností cieľového prvku. Tento mód je vhodný, ak naviazaný element je implicitne iba na čítanie (read-only).

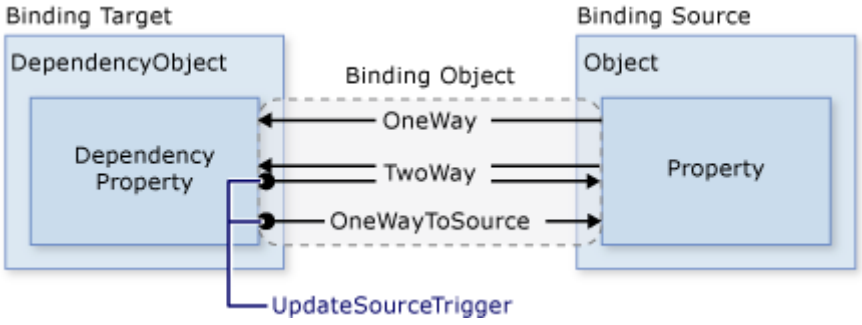
- TwoWay - spôsobuje zmeny dvoma smermi. Aj z naviazaného elementu k zdroju aj opačne. Tento typ je určený najmä pre plne editovateľné formuláre.

- OneWayToSource - je opačný k módu OneWay. Mení iba vlastnosti zdroja ale pri zmene zdroja sa zmena neodráža na naviazanom elemente.

- OneTime - tento mód sa používa na zobrazenie aktuálneho stavu dát v zdroji alebo ak sme si istí že dáta sú statické a stačí ich načítať len raz.

Každá dátová väzba má štyri komponenty. Cieľový objekt (Binding Target Object), cieľovú vlastnosť (Target Property), zdroj dát (Binding Source) a cestu k premennej v zdroji dát, ktorú chceme použiť (Binding Path).

Ďalší základný princíp je, že cieľová vlastnosť musí byť závislá premenná (Dependency Property). Väčšina vlastností elementov užívateľského rozhrania sú tohto typu a väčšina týchto vlastností, okrem vlastností, ktoré sú iba na čítanie (read-only) podporujú Data Binding. Obrázok 2.1. demonštruje WPF Data Binding koncepty.



Obrázok 1.1. Data Binding koncepty.

2.1.2.2 Command Binding

Tak isto ako môžeme naviazať na vizuálne komponenty nejaké dáta, môžeme naviazať aj príkazy, ktoré sa majú po nejakej akcii vykonať. Command Binding je ďalšia významná vlastnosť WPF, ktorá nám umožňuje oddeliť logickú časť od prezentačnej. Jednotlivé príkazy (Commands) implementujú interface ICommand. Trieda príkazu obsahuje akciu, ktorá sa má vykonať a môže obsahovať aj predikát, ktorý hovorí, či môže byť príkaz vykonaný. Na nasledujúcom príklade je ukážka časti definície triedy, ktorá implementuje interface ICommand.

```
public class RelayCommand : ICommand
{
  readonly Action<object> _execute;
  readonly Predicate<object> _canExecute;
  public RelayCommand(Action<object> execute, Predicate<object> canExecute)
  {
     _execute = execute;
     _canExecute = canExecute;
  }

  public void Execute(object parameter)
  {
     _execute(parameter);
  }
  public bool CanExecute(object parameter)
  {
     return _canExecute == null ? true : _canExecute(parameter);
  }
}
```

2.1.3 Podpora validácií

Ďalšia kľúčová vlastnosť, ktorá dopĺňa dátové väzby je validácia. Inými slovami odchytávanie nesprávnych zadaných hodnôt, reagovanie na ne a zobrazenie chybových správ. WPF poskytuje validácie, ktoré úzko spolupracujú s dátovými väzbami. Pre vrátenie chyby nejakou vlastnosťou stačí pri zmene vlastnosti objektu skontrolovať jej hodnotu a vyhodiť výnimku, ktorú naviazaný komponent zaregistruje a môže zobraziť chybovú správu výnimky. Ďalšia možnosť je implementovať interface IDataErrorInfo. Pri tejto možnosti netreba vyhadzovať výnimky, ale sa nadefinujú podmienky a chybové hlásenia, ktoré má objekt vracať pri zmene vlastností komponentov. Pri splnení požiadaviek objekt nevráti nič (null). Na

nasledujúcej ukážke XAML je znázornená definícia dátových a príkazových väzieb. Väzby sú definované v elemente Binding. Vlastnosti väzby sú nastavené atribútmi Path, ValidatesOnDataErrors a UpdateSourceTrigger. Na príklade je vidno aj možnosť rôzneho zápisu väzieb. Pri príkazovej väzbe, ktorá je nastavená v atribúte tlačidla Command je použitý druhý typ zápisu.

```
<TextBox Name="textBox_Mail">
  <TextBox.Text>
    <Binding Path="Leader_Mail"
             ValidatesOnDataErrors="True"
             UpdateSourceTrigger="PropertyChanged">
    </Binding>
  </TextBox.Text>
</TextBox>

<Button Content="Pridať" Command="{Binding Path=CommandAdd}" />
```

2.1.4 Dátové šablóny

Vo WPF rozlišujeme dva typy šablón. Šablóna ovládacieho prvku (Control Template) nám umožňuje detailne určiť správanie nejakého typu ovládacieho prvku a nadefinovať štýly.
Druhý typ sú dátové šablóny (Data Templates), pomocou ktorých môžeme podľa svojich predstáv popísať zobrazenie dát pomocou rôznych vizuálnych komponentov. Data Templates sa často používajú pri potrebe zobraziť nejakú skupinu objektov reprezentovanú poľom alebo zoznamom. Pre zoznam objektov si nadefinujeme Data Template, ktorý zobrazuje jednotlivé objekty a ich vlastnosti použitím viacerých komponentov. Takto môžeme jednoducho zobrazovať zoznam s obrázkami, tabuľkami, textom alebo tlačidlami, ktoré môžeme naviazať na nejaký príkaz.

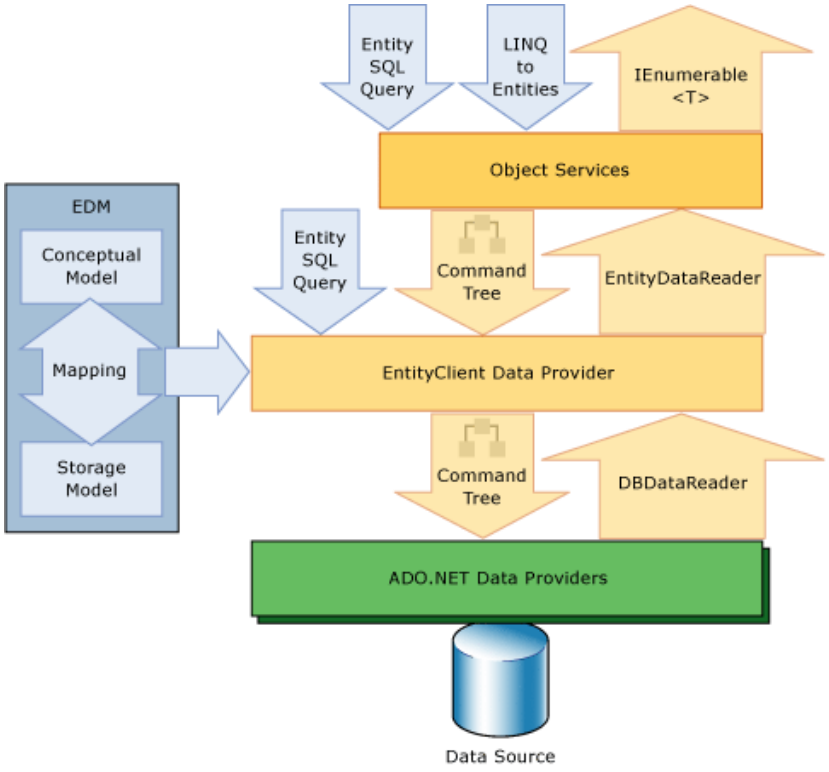
2.1.5 Výhody a nevýhody

Výhody WPF sú hlavne použitie grafického hardvéru pri vykresľovaní, podpora animácií, väzby, dedičnosť vlastností, štýly, šablóny, validácie, oddelené užívateľské rozhranie od logiky aplikácie a deklaratívne programovanie pomocou XAML.
Pri mnohých výhodách, ktoré boli niektoré aj popísane v predchádzajúcich častiach, natrafíme samozrejme vo WPF aj na nevýhody. Pre vývojárov, ktorí doteraz používali WinForms to

bude úplne nový prístup písania užívateľského rozhrania, ktorému je potrebné obetovať nejaký čas na štúdium. Programátor, ktorý nemá skúsenosti s prácou s grafikou naplno, nevyužije grafické možnosti WPF. Každá nová technológia nie je úplná a vo WPF ešte chýbajú niektoré komponenty, na ktoré boli vývojári zvyknutí a je WPF je menej stabilné ako dlho zaužívané WinForms. Niekedy býva problém s použitím vizuálnych WPF dizajnérov a malá podpora intelisense pri písaní kódu. Poslednou väčšou výraznejšou nevýhodou je, že WPF nespustíme na starších operačných systémoch Windows.

2.2 ADO.NET Entity Framework

Prevažná väčšina dnešných aplikácií pracuje s dátami uloženými v nejakej databáze. Prístup k dátam a práca s nimi je niekedy zdĺhavá. Uľahčiť a zrýchliť túto prácu nám môžu objektovo – relačné mapovacie frameworky. Jedným z nich je ADO.NET Entity Framework [6]. ADO.NET Entity Framework podporuje Entity Data Model (EMD) na definovanie dát na konceptuálnej úrovni. Popis databázy, vzťahov a mapovanie tabuliek na jednotlivé entity sú popísané v troch vrstvách. Jednu entity môžeme chápať ako objekt ktorý je reprezentáciou nejakých dát z databázy. Keďže pri WPF používame dátové väzby a naviazanie objektov na komponenty užívateľského rozhrania tak nás objektovo – relačný model databázy ušetrí od zdĺhavého písania kódu a vytvárania objektov. Objekty, s ktorými chceme pracovať, sú ako modely popísané v jednej z vrstiev Entity Frameworku. Pri Entity Frameworku nezáleží na poskytovateľovi dát, môžeme ho teda použiť s databázami od viacerých výrobcov.



Obrázok 2.2. Schéma architektúry Entity Frameworku.

2.2.1 Vrstvy Entity Frameworku

Entity framework poskytuje viac vrstiev pre popísanie zobrazenia databázových tabuliek na objekty. Model databázy si môžeme vygenerovať pomocou dizajnéra, ktorý nám vytvorí súbor s príponou .edmx, ktorý obsahuje XML popis všetkých vrstiev. Tieto vrstvy si môžeme napísať aj sami, ale strávili by sme pri tom neporovnateľne veľa času. Model databázy môže byť popísaný tak, že sa odlišuje od skutočnej štruktúry databázy, ale môže byť popísané aj mapovanie jedna ku jednej. Entity Framework Model má tieto 3 vrstvy:

- Logická – táto vrstva definuje relačné dáta
- Konceptuálna – definuje objekty .NET
- Mapovacia (zobrazovacia) – definuje vzájomné vzťahy medzi triedami .NET a relačnými tabuľkami a ich vzťahy

Každá z týchto vrstiev je popísaná špecifickým XML jazykom, pre každú vrstvu iným. V logickej vrstve sú popísané jednotlivé tabuľky, ich riadky, vzťahy medzi tabuľkami ako aj

uložené procedúry. V konceptuálnej vrstve máme pre nás definované entity, ktoré popisujú vytvorené objekty už vo zvolenom programovacom jazyku. Tieto entity nemusia byť totožné ako v logickej vrstve. To aby boli naviazané správne prvky z logickej vrstvy na prvky s konceptuálnej vrstvy zabezpečuje mapovacia vrstva. Je už na programátorovi ako si vrstvy navrhne. S popisom týchto troch vrstiev dostaneme Entity Framework Model. V konceptuálnej vrstve sú však popísané len entity, objekty musia byť definované v bežnom programovacom jazyku pre .NET C# alebo Visual Basic. Tieto objekty sa vytvárajú pomocou dizajnéra a majú definované verejné vlastnosti, metódy, funkcie a sú implementáciou rozhrania EntityObject.

2.2.2 Práca s dátami pomocou Entity Frameworku

Výber dát z databázy pomocou Entity Frameworku je vo veľa prípadoch pohodlná záležitosť. K dátam môžeme pristupovať pomocou Language Integrated Query (LINQ) [7], čo je ďalšie rozšírenie frameworku. LINQ je jazyk, ktorý umožňuje písať dotazy na dáta natívne v nejakom .NET jazyku. V tomto jazyku môžeme dotazovať na viac typov úložísk, preto existuje LINQ to XML, LINQ to SQL, LINQ to Objects, ale pre Entity Framework je určený LINQ to Entities.

Výhoda LINQ to Entities je hlavne to, že je veľmi jednoduchý, intuitívny a nepotrebuje zdĺhavé štúdium. Môžeme pomocou neho dotazovať na konceptuálnu vrstvu Entity Framework modelu. Jednotlivé dotazy sa potom preložia do jazyka, ktorým sa dotazuje na príslušný dátový zdroj a model databázy vráti objekty, s ktorými môžeme následne pracovať pri programovaní. LINQ to Entities podporujú všetky potrebné funkcie ako v SQL group by, sort, join, agregačné funkcie a iné a dajú sa písať viacerými spôsobmi. Dáta vyberané z databázy nemusia byť striktne objekty, ktoré máme nadefinované a zodpovedajú entitám z konceptuálnej vrstvy. Pri výbere nemusíme definovať o aký objekt ide a vytvoríme anonymné objekty. Môžeme však aj definovať, aký výsledok nám dotaz vráti. Výsledok býva väčšinou kolekcia objektov a tú môžeme prekonvertovať na zoznam, pole alebo môžeme vybrať iba samostatný objekt. Nasledujúce príklady sú ukážky LINQ to Entities a vygenerovaného SQL.

```
//Vytvorenie inštancie dátového dátového kontextu
DataEntities data = new DataEntities();

//LINQ to Entities dotaz, ktorý vráti kolekciu mien tímov
var teams = from t in data.Teams
            where t.Name.Length > 10
            orderby t.Name
            select t.Name;

//Vygenerovaný SQL dotaz
SELECT [Extent1].[Name] AS [Name]
FROM [dbo].[Teams] AS [Extent1]
WHERE (LEN([Extent1].[Name])) > 10
ORDER BY [Extent1].[Name] ASC
```

Tak ako výbery dát je jednoduchá práca aj so zmenou a mazaním dát. Pri zmene objektu, ktorý reprezentuje entitu zavolaním príslušnej metódy, uložíme dáta do úložiska. Tak isto mazanie vykonáme mazaním objektu príslušnou metódou. Keď máme v databáze definované uložené procedúry, tak pri generovaní tried entít sa vygenerujú aj metódy s parametrami, pomocou ktorých môžeme volať tieto uložené procedúry.

Pri práci s objektami nepracujeme s dátami uloženými v databáze, ale s tými, čo sú uložené v pamäti. Výhoda je, že pri každom výbere dát si môžeme overiť, či nie sú načítané v pamäti. Ak sú, nemusíme ich zbytočne vyberať z databázy nanovo. Treba si však dávať pozor či sú dáta aktuálne a ak nie, znova ich načítať z databázy. V prípade ak pracuje s databázou viac užívateľov naraz, treba dbať aby sa pri každej zmene dát znova načítali všetky dáta z databázy nanovo, aby sa tak predišlo nekonzistencii dát.

2.2.3 Nevýhody Entity Frameworku

Entity Framework je nová technológia takže má okrem výhod samozrejme aj nedostatky. Pri práci prináša veľmi jednoduchý prístup k dátam a málo písania zložitých SQL dotazov, ale pri niektorých prípadoch nie je tento framework taký flexibilný a pri zložitejšie navrhnutých databázach sa stávajú dotazy neprehľadné a nie je dobré, keď programátor nemá pod kontrolou SQL dotazy, ktoré sa vykonávajú voči databáze. To môže mať za následok napríklad zníženie výkonu. Ďalej pri entitách sa nedá namapovať priamo foreign key a pri dotazoch sa musí k nemu pristupovať cez druhú entitu, pretože pri reláciách sa mapuje len relačná entita. Ďalšia nevýhoda je, že sa musí dávať pozor, či sú dáta v entitách aktuálne alebo

boli medzi časom zmenené. Tento problém môže ľahko zapríčiniť nekonzistenciu dát. Entity Framework síce podporuje uložené procedúry, ale ani ich použitie nie je veľmi flexibilné, pretože sa dajú definovať len na operácie insert, update a select a nemožno si napríklad definovať uložené procedúry na mazanie celých tabuliek.

2.3 Model – View – ViewModel

Vývoj užívateľského rozhrania (User Interface, UI) aplikácií býva často zložitý. Užívateľské rozhranie musí obsahovať rôzne dáta, interaktívny dizajn, validácie a byť sa s ním pohodlne pracovať. Vzhľadom na to, že užívateľské rozhranie aplikácie odhaľuje celý systém, musí uspokojiť koncových užívateľov a môže to byť veľmi menená časť aplikácie. Existujú populárne návrhové vzory (Design Patterns), ktoré nám veľmi pomáhajú tieto problémy riešiť. Poznáme jednoduchšie aj zložitejšie návrhové vzory. Pri zložitých je pravdepodobné, že neskôr programátor nedodržuje presne dané postupy a kazí tak snahu vyvíjať aplikácie správnym spôsobom. Niekedy to však nie je zložitosťou návrhového vzoru, ale používaním nejakého návrhového vzoru pri technológiách, na ktorý nie je určený. To má za následok okrem iného aj písanie veľa riadkov kódu.

Ako sa postupne presadzuje v softvérovom svete WPF, komunita vývojárov , ktorí používajú tento prezentačný framework vyvinula svoj návrhový vzor Model-View-ViewModel (MVVM) [12] . Tento návrhový vzor je určený špeciálne pre WPF.

2.3.1 História

Po celú dobu ako ľudia začali vyvíjať softvér a ich užívateľské rozhranie, existovali návrhové vzory, ktoré im uľahčovali prácu. Model – View – Presenter (MVP) si získal veľkú popularitu vo veľa platformách. Je to vlastne variácia Model – View – Controller (MVC) návrhového vzoru.

V roku 2004 predstavil Martin Fowler Presentation Model, ktorý je podobný ako MVP v tom, že je oddelený View od logiky aplikácie [12]. Zaujímavá črta Presentation Modelu je, že mení svoj View pomocou dvoch stavov v synchronizácii medzi sebou. Logika je definovaná v triedach Presentation Modelu.

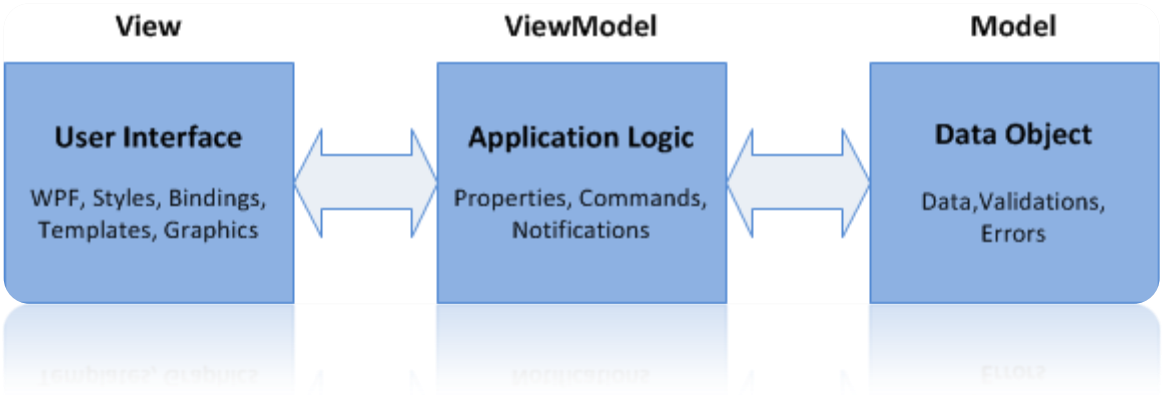
Jeden zo známych architektov WPF a Silverlight John Grossman v roku 2005 na svojom blogu uverejnil MVVM návrhový vzor, ktorý je identický Presentation Modelu. Odlišný je v tom, že

je určený striktne pre WPF. Presentation Model bol určený pre vyvíjanie aplikácii nezávisle na platforme.

2.3.2 Štruktúra MVVM

Ako vyplýva z názvu tak MVVM návrhový je rozdelený na tri časti.

- Model – objekt s ktorým pracujeme
- ViewModel – logická časť
- View – vizuálna časť



Obrázok 2.3. Štruktúra MVVM návrhového vzoru.

2.3.2.1 Model

Model reprezentuje pomocou objektov dáta, s ktorými pracujeme. Trieda Modelu obsahuje jednotlivé premenné, vlastnosti, funkcie a validácie. Pokiaľ majú tieto objekty vracať chybové hlásenia o nevalidných dátach musia implementovať IDataErrorInfo interface a nadefinovať pre jednotlivé vlastnosti validácie. Toto je veľmi dôležitá črta každého modelu, u ktorého validácie vyžadujeme, pretože v spojení s WPF sa dátovými väzbami chyby jednoducho zobrazujú.

2.3.2.2 ViewModel

Táto časť obsahuje logickú časť medzi časťami Model a View. Je to vlastne objekt, ktorý má nadefinované vlastnosti určené pre zobrazenie. Pre spoluprácu s WPF musí implementovať interface INotifyPropertyChanged. Potom bude zaručené, že objekt bude obsahovať udalosť,

ktorá oznámi vizuálnym prvkom, keď sa zmení hodnota nejakej vlastnosti. ViewModel môže obsahovať vlastnosti aj viacerých modelov. V tejto časti sú definované aj akcie, ktoré sú naviazané na udalosti komponentov užívateľského rozhrania. V skutočnosti chybové hlásenia nevracia Model, ale ViewModel, ktorý musí byť tiež implementácia interfacu IDataErrorInfo. ViewModel si však chybové hlásenia berie z Modelu.

2.3.2.3 View

View je prezentačná alebo vizuálna časť návrhového vzoru. Keďže MVVM vzor je určený pre WPF, View je napísaný v XAML jazyku a má priradený ako zdroj dát inštanciu ViewModelu. Potom jednotlivým komponentom užívateľského rozhrania nadefinujeme dátové a príkazové väzby, prípadne zobrazenie chybových hlásení.

2.3.3 Príkazy

Každý View v aplikácií používajúcej MVVM model má prázdny codebehind súbor, v ktorom pri bežnom postupe býva napísaná aplikačná logika. Jediná vec, ktorú v ňom nájdeme je inicializácia komponentov v konštruktore. V tomto súbore býva väčšinou definované, čo sa má vykonať po zavolaní udalosti komponentov. V MVVM tento súbor nie je vôbec potrebný z dôvodu, že v MVVM modeli sa využívajú príkazové väzby a komponenty sú naviazané na príkazy definované vo ViewModeli. Objekty týchto príkazov sú implementácie interfacu ICommand. Každý príkaz má definovanú akciu, čo má vykonať a môže mať definovaný aj predikát, či sa môže vykonať. Vizuálne komponenty ako button, hyperlink, menuitem môžu byť jednoducho naviazané na vlastnosti ViewModelu typu ICommand. Na nasledujúcom príklade je ukázané definovanie príkazu vo ViewModeli, ktorý sa môže naviazať na komponent z užívateľského rozhrania. Trieda RelayCommand je implementácia rozhrania ICommand. Verejná vlastnosť SaveCommand vráti inštanciu objektu príkazu. Pri vytváraní inštancie mu predáme ako parametre metódu Save() a funkciu CanSave().

```
RelayCommand _saveCommand;
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
        {
            _saveCommand = new RelayCommand(param => this.Save(),
                                            param => this.CanSave);
        }
        return _saveCommand;
    }
}
```

2.3.4 Testovateľnosť

Návrhový vzor MVVM je veľmi populárny aj z toho dôvodu, že triedy ViewModel sú ľahko testovateľné. Ak je aplikačná logika zapísaná vo ViewModel triedach, je jednoduché napísať testy týchto tried. Takto budú testy ďalšími konzumentmi ViewModel tried. S testami napísanými pre jednotlivé ViewModel triedy môžeme rýchlo otestovať správnosť a týmto môžeme ušetriť čas, pretože na chyby by sme prichádzali až pri zobrazovaní dát vo Viewoch.

2.3.5 Výhody a nevýhody

MVVM návrhový vzor je vhodné pre WPF, pretože je určené pre túto platformu a WPF bolo navrhnuté tak, aby bolo jednoduché vytvárať aplikácie pomocou tohto vzoru. Vyvíjať aplikácie týmto spôsobom má veľa výhod.

Keďže ViewModel je nezávislý na vizuálnych komponentoch, tak sa môžu písať najskôr triedy ViewModelov a ich testy a až potom vizuálne časti. Táto možnosť je veľmi dobre využiteľná pri tímovom vývoji. Kým dizajnéri pracujú na pokročilom grafickom užívateľskom rozhraní, programátori môžu písať ViewModel a Model triedy a ich testy. Spojenie práce dvoch tímov spočíva už len v nadefinovaní dátových a príkazových väzieb popísaných v jazyku XAML.

Pri vyvíjaní veľkých aplikácií sa stáva štruktúra neprehľadná a chaotická. Tento vzor nás núti štruktúrovať aplikáciu do logických celkov a následne pomáha orientovať sa v množstve riadkov kódu.

Nevýhodou MVVM môže byť jeho použitie v drobných aplikáciách. Pri menej rozsiahlych aplikáciách a jednoduchých užívateľských rozhraniach je použitie vzoru zbytočná komplikácia. Ďalej pri zlej správe väzieb a ich nastavení môže byť viac zaťažovaná pamäť počítača. Pri odlaďovaní aplikácie a hľadaní chýb v nastaveniach väzieb sa často stretneme s problémami pretože nemáme väzby nastavené v kóde tried, ale vo vizuálnej časti. Kód užívateľského rozhrania je validný, ale väzby môžu byť nadefinované na neexistujúce vlastnosti ViewModelu.

3 Informačný systém pre riadenie a vyhodnocovanie súťaže

Informačné technológie veľmi uľahčujú a zefektívňujú prácu v mnohých odvetviach. Stretneme sa s nimi na každom kroku a postupne nahrádzajú ľudí pri práci, ktorú môžu rýchlejšie a efektívnejšie vykonať za nich. Dobre špecifikovaný, navrhnutý a implementovaný softvér alebo systém môže v určitých prípadoch výrazne pomôcť a robiť prácu, ktorá bola doteraz vykonávaná ručne, bola zdĺhavá a potrebovala viac ľudí, aby bola zvládnuteľná.

Každoročne sa koná súťaž RoboCup Junior Slovensko, pri ktorej sa stretneme s množstvom práce pri pripravovaní, organizovaní, evidovaní tímov, zapisovaní informácií a vyhodnocovaní. Veľká časť tejto práce by mohla byť vykonávaná pomocou systému navrhnutého pre túto súťaž a tak ušetriť organizátorov od práce, ktorú nemusia vykonávať. Našou úlohou bolo špecifikovať, navrhnúť a implementovať tento systém s použitím technológií popísaných v predchádzajúcej kapitole.

3.1 Požiadavky

Pre správny návrh a implementáciu je potrebné definovať úlohy a očakávania od systému. Definovať požiadavky nie je triviálna záležitosť. Ak dobre definujeme požiadavky, môžeme tak predísť budúcim nezrovnalostiam s koncovým užívateľom a zmenám systému, ktoré môžu byť veľmi náročné na čas. Ako prvá vec k definovaniu požiadaviek bolo detailné štúdium informácií okolo súťaže o súťaži Robocup Junior.

3.1.1 Definícia požiadaviek

Požiadavky na systém pre vyhodnocovanie výsledkov súťaží sú obsiahle. Pri definícii nám pomohla účasť na súťaži Robocup Junior a zistenie, ako súťaž prebieha, pričom by mohol pomôcť nejaký systém a pričom by bolo zbytočné použitie iných riešení, ako sa využívali doteraz. Počas účasti sme už využívali malú časť naimplementovanej aplikácie, ktorá pri

niektorých procesoch pomohla organizátorom. Bolo to však len malá časť oproti tomu, čo by organizátori potrebovali a aj prostredníctvom jej použitia sme zistili, aké zmeny je nutné vykonať.

Súťaže sa zúčastnilo viac než, približne štyridsať tímov, spolu okolo dvesto jednotlivcov. Evidencia týchto súťažiacich bola v minulosti vykonávaná ručne. Čiže ešte pred štartom nebolo niekedy jasné, ktoré tímy štartujú, v akých kategóriách a s akými členmi, pretože ručná evidencia bola ťažko zvládnuteľná. Pri evidencii tímov sme už pracovali s prvou verziou softvéru, kde sme mali uložené dáta o všetkých tímoch v databáze. Pri používaní sme prišli na mnoho nedostatkov aplikácie a funkcionalít, ktoré by sa dali zmeniť, vylepšiť alebo úplne odstrániť, pretože boli zbytočné. Po začiatku súťaže sa konali súťažné kolá jednotlivých kategórií. Naša prvá verzia softvéru ešte neobsahovala funkcionality na organizovanie súťaže. Preto organizátori museli všetky štartovné listiny vytvárať ručne na papieri alebo v počítači. Ďalšou kritickou časťou bolo vyhodnotenie výsledkov z bodov zaznačených rozhodcami do hárkov a následné zostavenie výsledkových listín, ktoré by bolo potrebné zverejniť na internete.

Tímy sa môžu prihlásiť do viacerých kategórií. Do vekovej kategórie sú zaradené podľa veku členov tímu. Vekové kategórie sú dve:

- Základná škola
- Stredná škola

Okrem vekových kategórií sú súťažné kategórie. Tieto kategórie predstavujú jednotlivé disciplíny, v ktorých tímy môžu súťažiť. Súťažné kategórie sú nasledovné:

- Konštrukcia
- Záchranár
- Tanec
- Futbal (futbal sa delí na viac podkategórií)

Po naštudovaní pravidiel, priebehu súťaže a po praktických skúsenostiach sme rozdelili požiadavky na štyri hlavné časti. Podrobná definícia požiadaviek je obsiahla, preto ich popíšeme len stručne. Hlavné časti požiadaviek sú:

- Evidencia tímov
- Generovanie štartovných listín
- Zápis výsledkov
- Vyhodnotenie

3.1.1.1 Evidencia tímov

Na začiatku súťaže je nevyhnutné pre všetky tímy registrovať sa p nejakej webovej stránke. Podmienkou pre všetky tímy je, ešte pred začiatkom súťaže, registrovať sa na prostredníctvom nejakej webovej stránky. Aby bolo možné meniť údaje už registrovaných tímov, je dôležitá možnosť prihlásiť sa pod menom a heslom a po prihlásení editovať svoje údaje. Po prihlásení ako administrátor musí byť možnosť meniť každý tím, alebo možnosť vymazať ho. Pre organizovanie súťaží je potrebné mať k dispozícii údaje o tímoch, ich vedúcich, členoch a to, v akých kategóriách súťažia. Ďalej evidovať počty členov tímu, ktorí potrebujú ubytovanie a stravu. Pre túto potrebu musí systém obsahovať exporty, ktoré organizátorom dávajú prehľad o týchto informáciách. Ďalšia dôležitá časť je registrácia na mieste, kde tímy potvrdia svoju účasť a ešte môžu zmeniť svoje údaje aj tesne pred súťažou. Ak sa nejaké tímy nestihli zaregistrovať prostredníctvom webovej stránky, musí byť možnosť ich pridať aj na mieste súťaže.

3.1.1.2 Generovanie štartovných listín

Na začiatku súťaže je potrebné, aby systém vygeneroval po zadaní kategórie štartovné listiny. Pri generovaní je potrebné zadať kategóriu, vekovú kategóriu, dátum začiatku, čas začiatku, počet kôl, prestávku medzi kolami a počet ihrísk, na ktorých môže súťaž prebiehať paralelne. Po zadaní týchto údajov systém vygeneruje časový harmonogram, podľa ktorého bude prebiehať súťaž. Vygenerované štartovné listiny musia byť vo forme, ktorú je možné vytlačiť.

Pre kategórie konštrukcia, záchranár a tanec sú štartovné listiny podobné. Rozdiel však nastáva pri kategórií futbal, kde sa rozpis zápasov generuje podľa viacerých pravidiel.

3.1.1.3 Zápis výsledkov

Počas každej súťaže zapisujú výsledky jednotlivých tímov rozhodcovia do hárkov rozhodcov. Každá kategória má iné hodnotiace kritériá, ktoré sa každý rok menia. Niektoré z kritérií sa môžu ešte násobiť koeficientom. Z tohto dôvodu systém musí obsahovať možnosť definovať formuláre na zapisovanie výsledkov a možnosť uložiť body tímov do databázy. Pri chybe treba umožniť vymazať alebo upraviť hodnotenie.
Futbal je iný typ kategórie a pre futbal je potrebná možnosť zapisovať len výsledky jednotlivých zápasov.

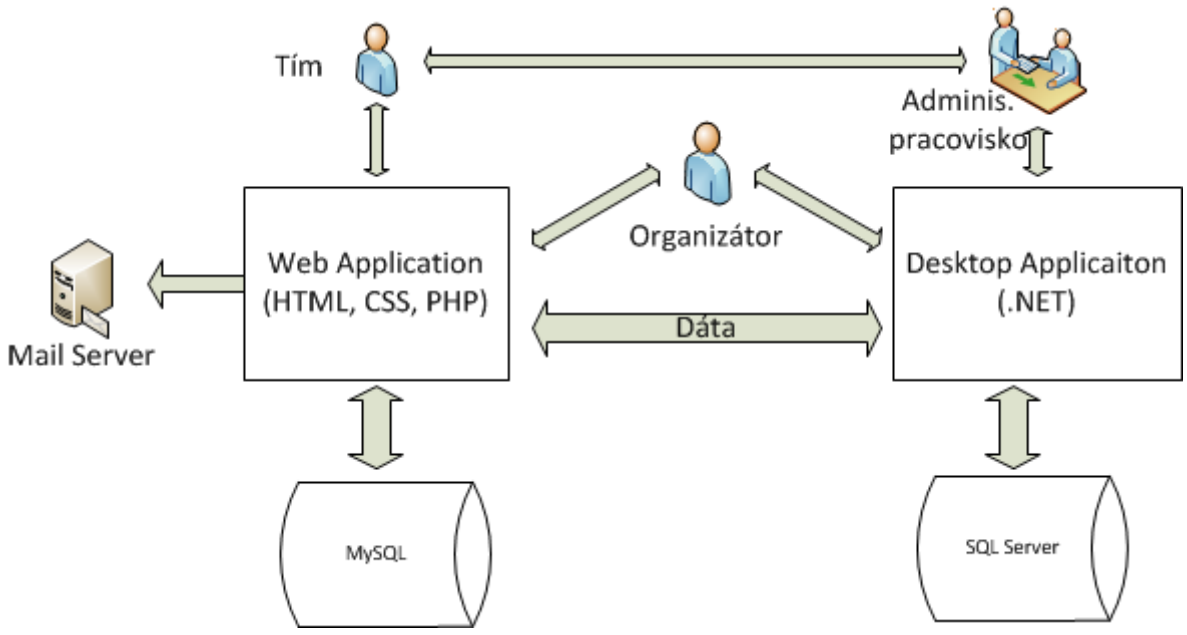
3.1.1.4 Vyhodnotenie

Po zadaní výsledkov rozhodcami je potrebné vyhodnotiť jednotlivé kategórie. Vyhodnocuje sa podľa daných pravidiel pre každú kategóriu iných. Po vyhodnotení je potreba vyexportovať výsledky s následnou možnosťou tlače.

Na adrese http://www.robotika.sk/rcj/index.php?page=rules sú umiestnené podrobné pravidlá súťaže.

3.2 Návrh systému

Z požiadaviek vyplýva, že systém bude rozdelený na dve hlavné časti. Na webovú aplikáciu, ktorá bude prístupná verejnosti a desktopovú aplikáciu, ktorá bude určená pre organizátorov súťaže. Webová aplikácia bude poskytovať dáta desktopovej v definovanej forme, aby bolo možné dáta jednoducho importovať a uložiť do inej databázy. Keďže systém mal byť navrhnutý, aby pracoval aj bez pripojenia nebolo možné použiť jednu databázu, ale zabezpečiť výmenu dát medzi oboma aplikáciami. Výmena dát bude prevádzkovaná pomocou XML súborov, ktoré môžu aplikácie exportovať a importovať. Exporty a importy bude vykonávať admin, pretože dáta budú prístupne len po jeho prihlásení. Na Obrázku 3.1. je znázornený pohľad na systém a vzťahy s užívateľmi.



Obrázok 3.1. Schéma systému a interakcia s užívateľmi.

3.2.1 Návrh webovej aplikácie

Hlavný účel webovej aplikácie je registrácia tímov. Táto aplikácia však bude slúžiť aj ako prezentačná a informačná stránka pre záujemcov o zúčastnenie sa súťaže alebo aj širokej verejnosti. Dáta sa budú ukladať do databázy MySQL a aplikácia bude napísaná v PHP. Logické celky rozdelím na tieto časti:

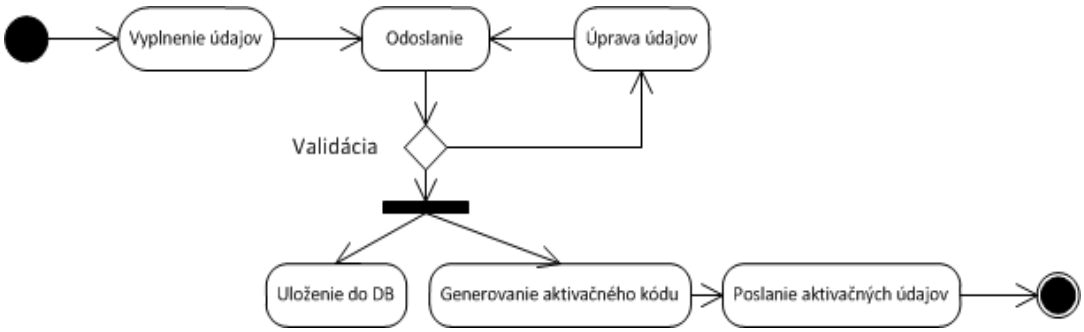
- Registrácia
- Prihlásenie
- Editácia
- Export dát

3.2.1.1 Registrácia

Registrácia bude prebiehať nasledovne.

1. Vyplnenie a odoslanie formulára – v tejto časti užívateľ vyplní formulár webovej aplikácie a odošle ho. Ak nie sú vyplnené údaje validné, systém upozorní na chyby. Po úspešnej kontrole korektnosti údajov údaje sa uložia do databázy, ale ešte nie sú aktivované. Po úspešnom uložení dát do databázy systém vygeneruje URL link

s kódom, pomocou ktorého sa dajú aktivovať dáta a tak potvrdiť registráciu a pošle ho na zadaný mail.



Obrázok 3.2. Stavový diagram prvej časti registrácie.

2. Potvrdenie – po potvrdení registrácie pomocou linku poslaného mailom sa stanú údaje v databáze aktívne. Systém vygeneruje meno a heslo pre prihlásenie a pošle ho na uložený mail registrovaného tímu. Po potvrdení registrácie sa pošle mail aj definovaným organizátorom, ako informácia, že pribudol nový tím.

Potvrdenie registrácie je dôležité z dôvodu poistenia, či môže byť registrácia považovaná za vážny záujem a či je platný aspoň kontaktný mail.

3.2.1.2 Prihlásenie

Po registrácií a potvrdení sa odošlú údaje, pomocou ktorých sa dá prihlásiť do webovej aplikácie. Po vyplnení prihlasovacích údajov systém rozpozná typ užívateľa a povolí prístup k ďalším častiam webovej aplikácie. Prihlásený užívateľ môže byť dvoch typov:

- Tím (zástupca tímu) – povolená editácia svojich údajov
- Admin – povolená editácia všetkých údajov, zobrazenie detailných informácií, export dát

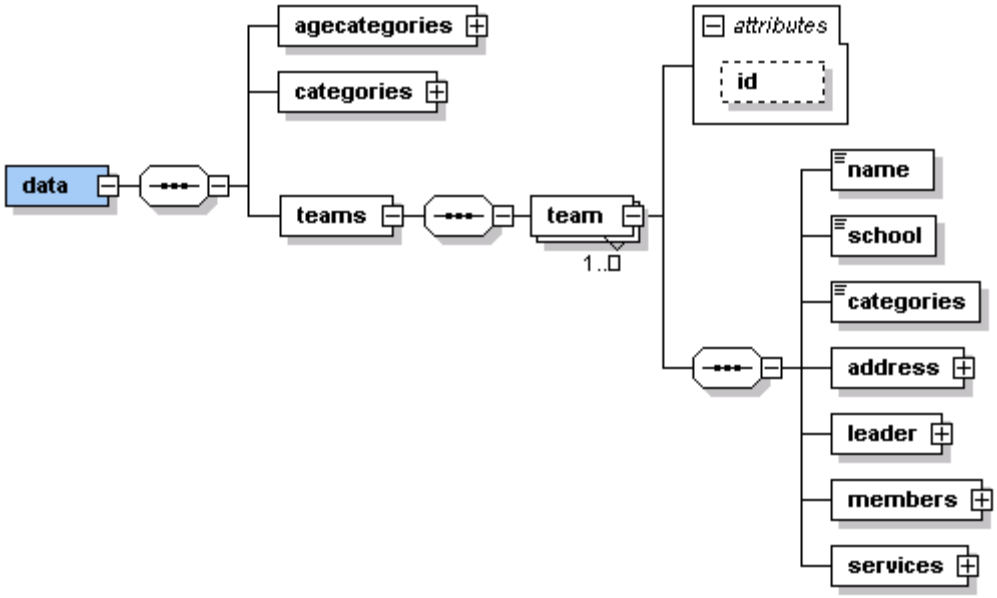
3.2.1.3 Editácia

Editácia tímov bude prístupná iba pre prihlásených užívateľov. Keď bude prihlásený tím, môže editovať iba svoje údaje. Adminovi bude zverejnený zoznam celých tímov, z ktorých každý bude môcť editovať. Po odoslaní údajov sa zmenia údaje príslušného tímu v databáze.

3.2.1.4 Export dát

Táto funkcionalita bude prístupná len prihlasným užívateľom typu admin. Exporty dát budú dvoch typov. Prvý typ budú len informácie zobrazené na stránke, ktoré budú slúžiť ako detailné údaje o tímoch. Druhý typ bude export dát určený pre ďalšiu časť systému, desktopovú aplikáciu.

Export dát pre desktopovú aplikáciu bude obsahovať všetky údaje o tímoch, ďalej o všetkých vekových a súťažných kategóriách. Formát dát bude XML, ktorého štruktúra bude pevne definovaná, aby bolo zaručené, že aplikácia dokáže tieto dáta korektne čítať. XML súbor bude mať štruktúru definovanú v XSD (XML Schema Definition).



Obrázok 3.3. Stromová štruktúra XML dát z exportu.

3.3 Návrh desktopovej aplikácie

Návrh tejto časti systému je dôležitá časť projektu. V návrhu budeme vychádzať z požiadaviek a z vlastností a možností technológií, ktoré budú použité pri implementácii. Z dôvodu, že systém má byť používaný počas nasledujúcich rokov, na ktorých môžu byť zmeny v pravidlách alebo v priebehu súťaže, musí byť systém navrhnutý flexibilne, aby sa zmenám ľahko prispôsobil, prípadne aby bol ľahko rozšíriteľný. Pre potrebu príjemného user-friendly používania je tiež potrebné navrhnúť intuitívne užívateľské rozhranie.

3.3.1 Architektúra aplikácie

Dobrým návrhom architektúry aplikácie je možné ušetriť veľa času pri implementácii. Vzhľadom na to, že výsledný systém bude prvá verzia, je pravdepodobné, že v budúcnosti bude treba vykonávať zmeny. Pri zlej architektúre by bolo vykonávanie zmien komplikované, pri niektorých prípadoch s potrebou preprogramovať veľkú časť aplikácie. Z požiadaviek vyplýva, že aplikácia bude rozsiahla a preto bolo ďalšou našou prioritou pri návrhu dobre rozdeliť aplikáciu na logické celky. Rozdelenie aplikácie prinesie veľa výhod: ľahko pridávať ďalšie časti aplikácie, jednoduchšia testovateľnosť, odstraňovanie chýb a dobrá orientácia v kóde.

Pri návrhu sme brali ohľad na technológie, ktoré budú použité pri implementácii. V predchádzajúcich kapitolách boli vysvetlené princípy vývoja aplikácií s použitím WPF a MVVM návrhového vzoru a vlastnosti objektovo – relačného mapovania na databázu pomocou ADO.NET Entity Frameworku. Vlastnosti WPF a Entity Frameworku sa dajú veľmi dobre použiť pri MVVM architektúre preto sme architektúru navrhli s použitím tohto vzoru.

3.3.1.1 Logické celky aplikácie

Základná filozofia MVVM vzoru je rozdelenie vyvíjaného softvéru do troch hlavných častí.

- Models – v tejto časti budú definované dátové objekty pre použitie v aplikácií. Tieto objekty budú mať definované dátové zložky a v prípade potreby aj validácie s návratovými chybovými hláškami.
- Views – táto časť budú obsahovať jednotlivé časti užívateľského rozhrania napísane ako typ UserControl v XAML jazyku.
- ViewModels – v tejto časti budú triedy s logikou pre jednotlivé okná aplikácie, verejné vlastnosti a príkazy. Jedno okno sa môže skladať z viacerých Views a jedna trieda ViewModel môže pracovať s viacerými modelmi.

3.3.1.2 Objekty entít

Viaceré dátové objekty nebudú umiestnené v časti Models, pretože budeme pracovať s dátovými objektmi entít Entity Frameworku definovaných v časti Data. Tieto triedy budeme však považovať tiež za triedy typu Model. Triedy budú vygenerované z entít Entity

Frameworku a budú vlastne zodpovedať tabuľkám v databáze s referenciami na iné objekty, ktoré reprezentujú asociované tabuľky. Ak bude potrebné, tieto triedy bude možné rozšíriť o validácie a o chybové hlásenia a o ďalšie potrebné vlastnosti, metódy alebo funkcie. Výhodou týchto objektov bude, že ich hocikedy môžeme vložiť načítať alebo vymazať z databázy, samozrejme pokiaľ to nebude v rozpore s ostatnými dátami v databáze.

3.3.1.3 Prepojenie objektov

V triede ViewModel bude vytvorená inštancia alebo viac inštancií triedy nejakého Modelu a ViewModel bude zverejňovať jeho vlastnosti a pýtať sa ho informácie o korektnosti údajov. Dôležité je prepojenie medzi objektami View a ViewModel. Toto prepojenie bude založené na väzbách, ktoré podporuje WPF. Komponenty triedy View budú naviazané na verejné vlastnosti a na definované príkazy ViewModel triedy. Oznámenia, že sa zmenili hodnoty premenných vo ViewModel objekte budú pre View oznamované prostredníctvom udalostí.

3.3.1.4 Validácie

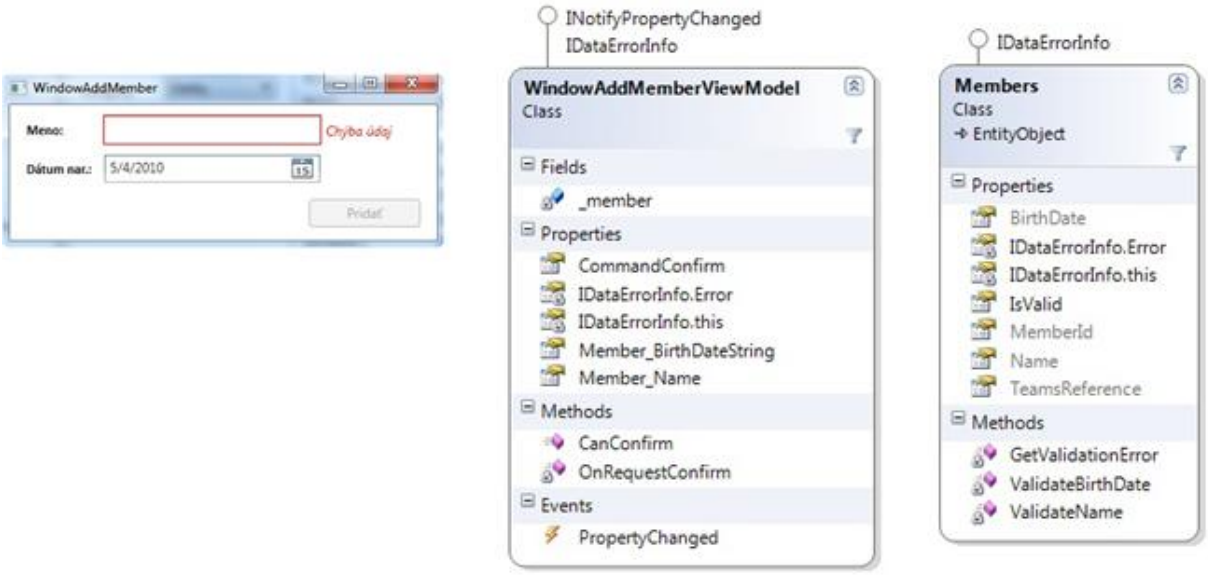
Validácie budú vykonávané pomocou vlastností a väzieb vo WPF. Objekty naviazané na View objekty budú implementáciou rozhrania IDateErrorInfo. Implementáciou tohto rozhrania dostaneme objekt, z ktorého View môže získať informáciu či je zadaný údaj validný alebo nie. Ak je validný, tak sa ako chybové hlásenie vráti null, inak vráti reťazec hovoriaci o chybe. Validácie budú definované v objektoch Model. ModelView je implementáciou rozhrania IDataErrorInfo z dôvodu, že je naviazaný na View. V skutočnosti bude vracať len objekt Modelu pretypovaný na IDataErrorInfo.

3.3.1.5 Príkazy

V triedach ViewModelu budú definované príkazy, ktoré bude možné naviazať na komponenty užívateľského rozhrania. Príkaz bude ako prístupný cez verejnú vlastnosť typu ICommand. Trieda príkazu má definovanú akciu, ktorá sa po zavolaní vykoná a môže mať definovaný aj predikát, ktorý hovorí, či je možne príkaz vykonať.

Na obrázku 3.4 sú znázornené jednotlivé triedy a formulár (View) na operácie pridávanie a editovanie člena tímu. Takéto triedy budú navrhnuté pre každé okno aplikácie. Trieda Members je Model a je implementácia rozhrania IDataErrorInfo. Obsahuje jednotlivé verejné

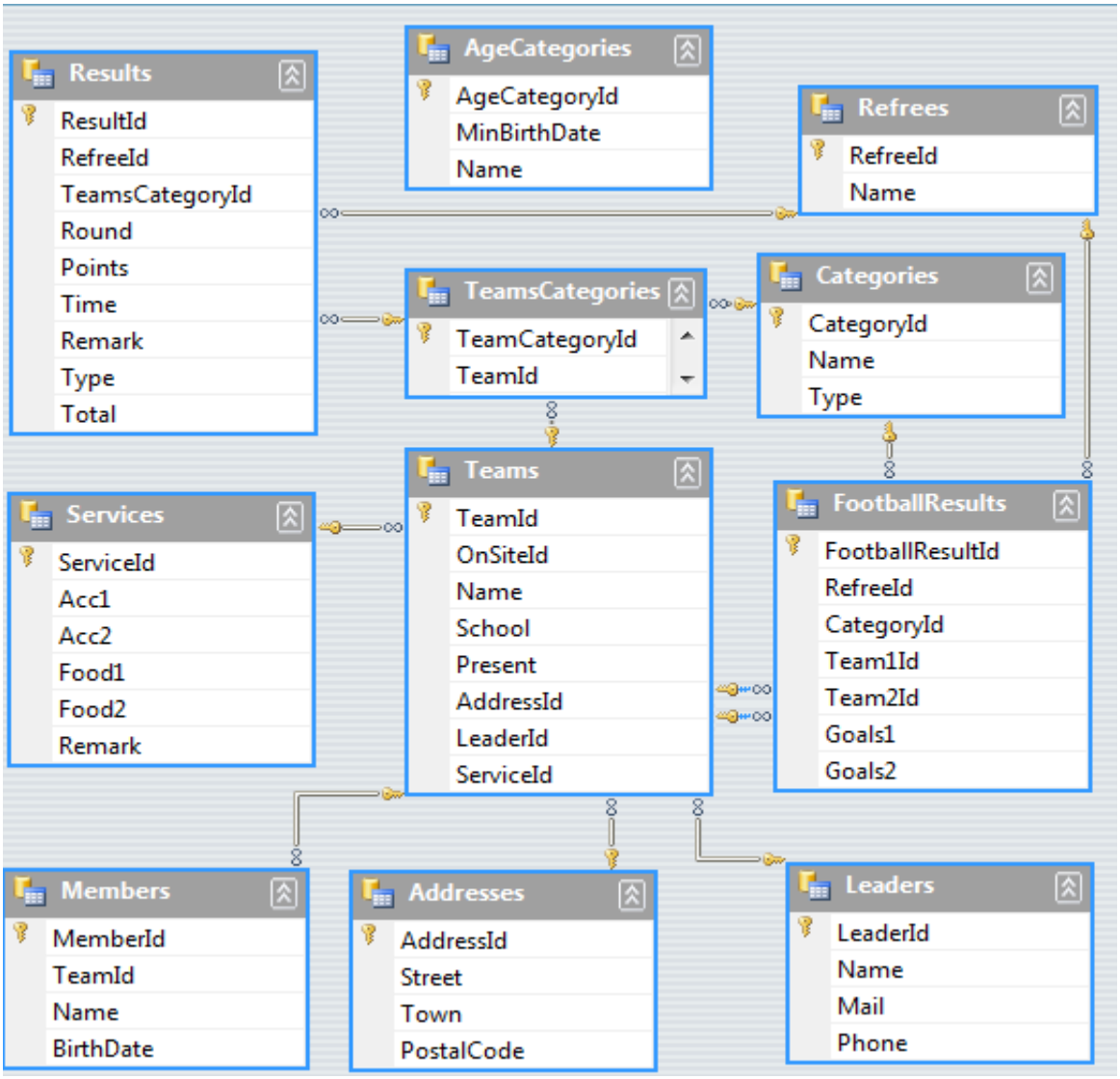
premenné a validácie s chybovými hláseniami. Názov triedy Members je v množnom čísle z dôvodu rovnakého názvu tabuľky v databáze. Trieda WindowAddMemberViewModel je trieda ViewModelu. V tejto triede sa nachádzajú verejné vlastnosti, ktoré sú určené k naviazaniu na View. Tieto vlastnosti vracajú vlastnosti inštancie objektu Members. Ďalej objekt obsahuje CommandConfirm, ktorý je naviazaný na tlačidlo Viewu. Pri vytváraní inštancie triedy príkazu sa posielajú ako parametre metóda OnRequestConfirm() a CanConfirm. Udalosťou PropertyChanged je oznámené Viewu, či sa zmenila nejaká hodnota naviazanej vlastnosti.



Obrázok 3.4: Ukážka častí architektúry (View, ViewModel, Model).

3.3.2 Dátový model

V aplikácii bude prioritná práca s dátami, preto je potrebné navrhnúť dátový model na zobrazenie logickej štruktúry dát. Tabuľky v databáze približne zodpovedajú dátovému modelu webovej aplikácie. Pridané budú tabuľky obsahujúce výsledky a rozhodcov. Aby bolo možné údaje spárovať s údajmi v databáze pre webovú aplikáciu, v tabuľke Teams je pridané OnSiteId. Pomocou tohto stĺpca bude možné neskôr aktualizovať dáta v druhej databáze.



Obrázok 3.5: Dátový model.

S databázou sa bude pracovať prostredníctvom ADO.NET Entity Frameworku. Mapovanie bude vygenerované a entity budú presne zodpovedať tabuľkám.

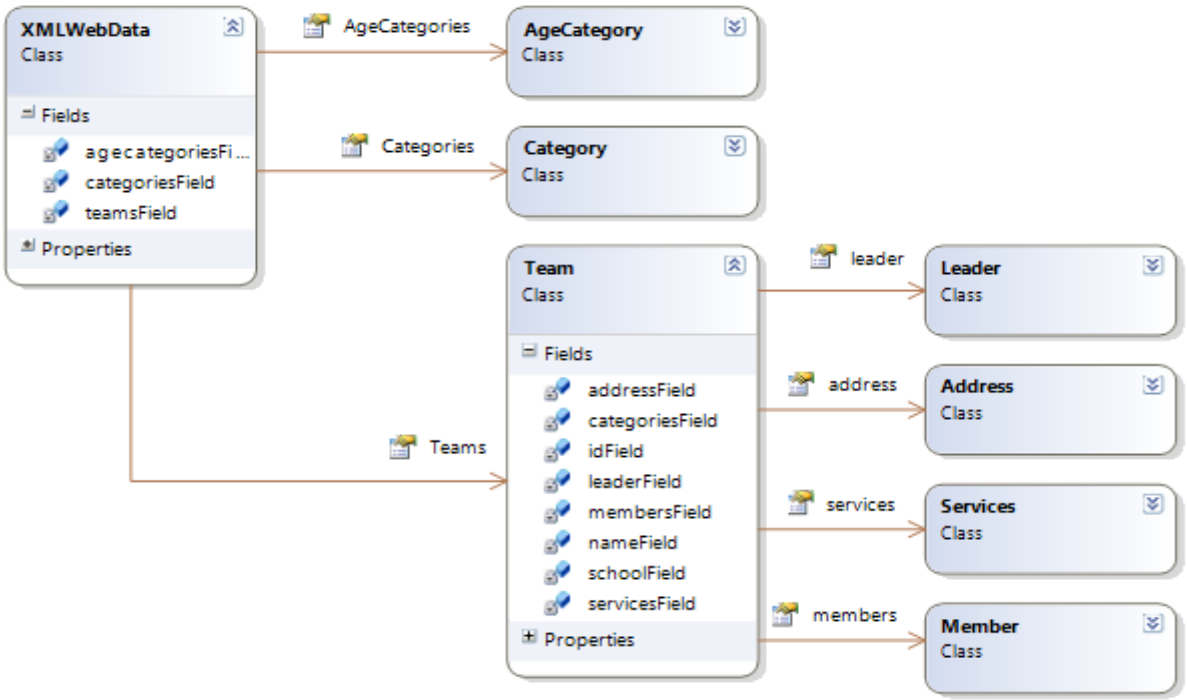
3.3.3 Import dát

Import dát je riešený nasledovne:

1. Otvorenie XML súboru s dátami (dáta budú uložené v XML súbore vygenerovanom web aplikáciou),
2. kontrola správnosti štruktúry,

3. kontrola existencie dát v databáze (ak už boli dáta importované, je potreba na to upozorniť),
4. serializácia XML na objekty.
5. Pridanie objektov do databázy

Pri importe využívame serializáciu XML dát na objekty, aby sme predišli pracnému parsovaniu XML dokumentu. Po vytvorení objektov ich prostredníctvom metód entity frameworku uložíme do databázy.

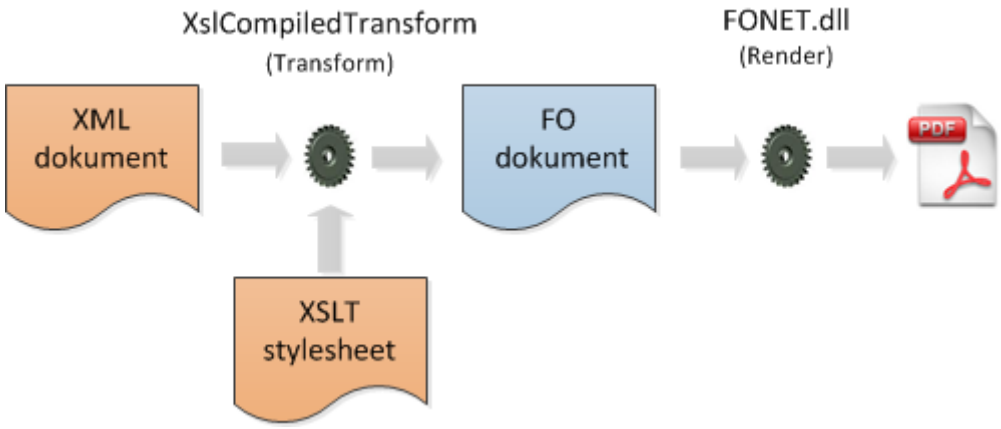


Obrázok 3.6: Objektová reprezentácia dát pre import.

3.3.4 Exporty pre tlač

Systém musí obsahovať exporty dát do formy, ktorá sa dá jednoducho tlačiť. Najpoužiteľnejšia forma pre tlač je v našom prípade PDF (Portable Document Format). PDF exporty je možné generovať pomocou naprogramovaných voľne stiahnuteľných knižníc. Tento spôsob však nie je veľmi flexibilný a ťažko by sa editovali výsledné formy exportov v prípade zmien. PDF exporty sme preto navrhli prostredníctvom XSL transformácií s využitím XML, XSL a XSL:FO. Základom bude definícia objektov, ktoré budú deserializovateľné do XML. Po načítaní objektov s dátami z databázy je možné pomocou

definovanej funkcie v objekte dostať reťazec, ktorý obsahuje XML reprezentáciu dát objektu. Ďalej sú potrebné nadefinovane transformačné súbory XSL s definíciami XSL:FO elementov. Pomocou transformácie vykonanej metódou vhodnej knižnice vytvoríme pomocou XML a XSL XSL-FO dokument. Z XSL-FO dokumentu už je možné priamo pomocou vhodnej knižnice vygenerovať výsledné PDF, ktoré sa užívateľovi zobrazí. Výhodou je ľahké editovanie XSL transformačných súborov v prípade zmien potrieb používateľov. Niektoré XML je možné použiť aj pre viac exportov. Obrázok 3.7 demonštruje postup vytvárania PDF. Na vytvorenie XSL-FO dokumentu z XML a XSLT sa použije trieda XslCompiledTransform a jej metóda Transform(). Z výsledného Fo dokumentu vytvoríme PDF použitím triedy FonetDriver knižnice Fonet.dll a jej metódy Render().



Obrázok 3.7: Schéma vytvárania PDF.

Exportov do PDF je potrebných niekoľko.

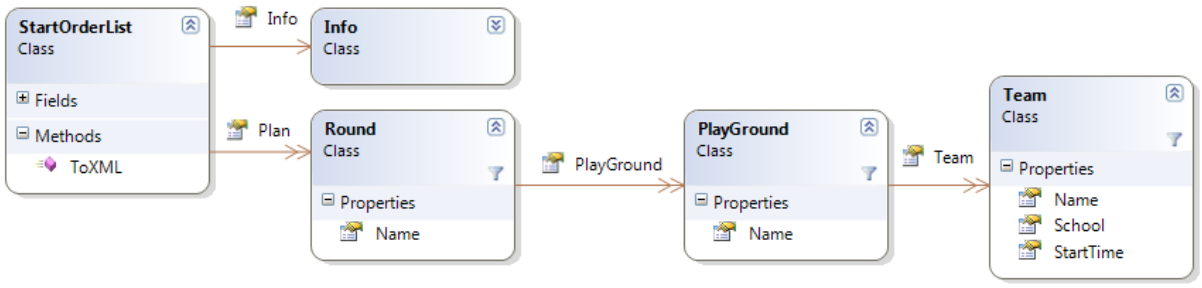
- Export všetkých údajov
- Export údajov jedného tímu
- Export štartovného poradia (rozdielne pre jednotlivé kategórie)
- Export výsledkov (rozdielne pre jednotlivé kategórie)

3.3.5 Generovanie štartovného poradia

Pre každú kategóriu je možné vygenerovať štartovné poradie. Pred generovaním sa zobrazí príslušný formulár, kde sú nastavenia pre generovanie. Jednotlivé kategórie môžu prebiehať aj na viacerých ihriskách zároveň a môžu mať aj viac kôl. Tieto možnosti tam sú zahrnuté. Pre kategórie konštrukcia, tanec a záchranár je generovaním možné vyplniť tieto nastavenia:

- Kategória
- Veková kategória
- Dátum a čas začiatku
- Počet kôl
- Počet ihrísk

Na obrázku 3.8 je zobrazený triedny diagram so znázornenou objektovou štruktúrou, ktorá sa naplní údajmi a funkciou ToXML() získame potrebné XML pre generovanie PDF štartovného poradia.



Obrázok 3.8: Triedny diagram pre PDF export štartovného poradia.

3.3.5.1 Kategória futbal

Pri kategórií futbal je logika generovania štartovného poradia iná. Generuje sa program zápasov tímov. Po zadaní kategórie a vekovej kategórie sa vygeneruje tabuľka pre zobrazenie zápasov typu každý s každým a časový rozpis jednotlivých zápasov.

Tím	Tím 1	Tím 2	Tím 3	Tím 4	Tím 5
Tím 1	X	Tím 1 : Tím 2	Tím 1 : Tím 3	Tím 1 : Tím 4	Tím 1 : Tím 5
Tím 2	Tím 2 : Tím 1	X	Tím 2 : Tím 3	Tím 2 : Tím 4	Tím 2 : Tím 5
Tím 3	Tím 3 : Tím 1	Tím 3 : Tím 2	X	Tím 3 : Tím 4	Tím 3 : Tím 5
Tím 4	Tím 4 : Tím 1	Tím 4 : Tím 2	Tím 4 : Tím 3	X	Tím 4 : Tím 5
Tím 5	Tím 5 : Tím 1	Tím 5 : Tím 2	Tím 5 : Tím 3	Tím 5 : Tím 4	X

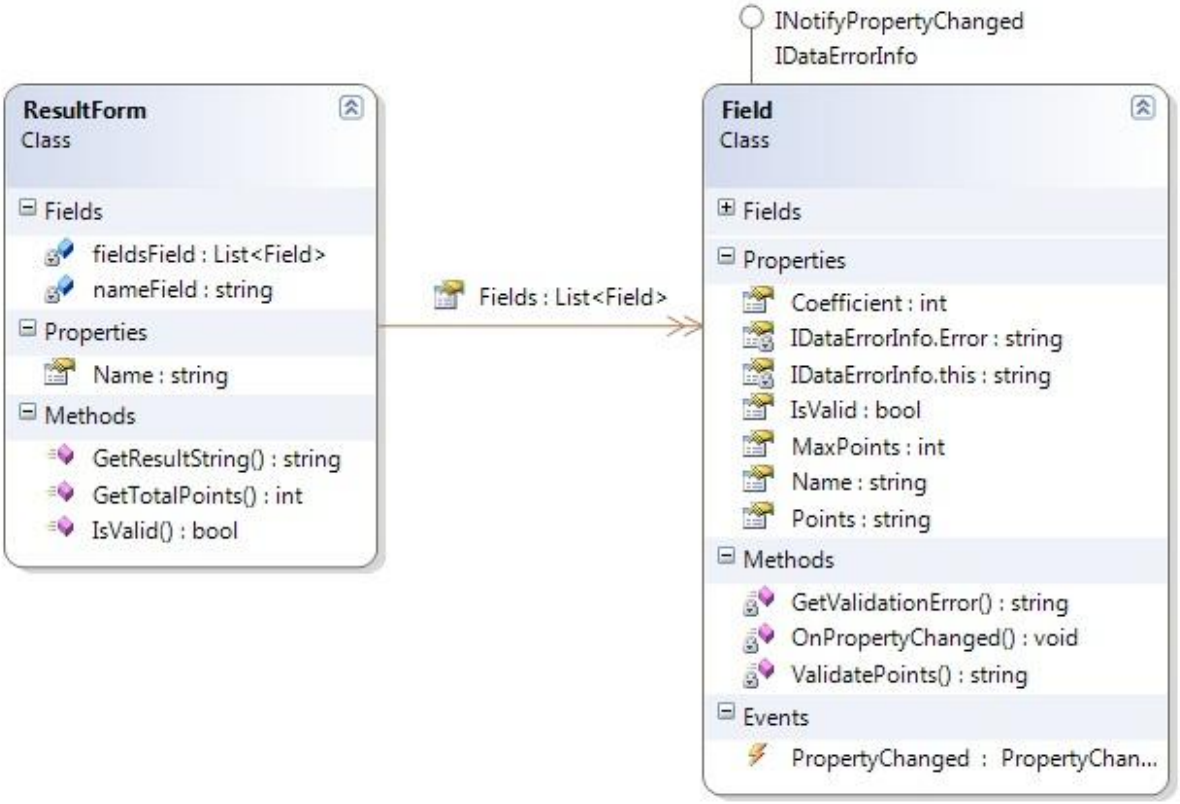
Obrázok 3.9: Tabuľka zápasov kategórie futbal.

Podľa tabuľky na obrázku 3.9 sa vygeneruje poradie zápasov. Zápasy pôjdu v poradí po diagonálach zľava doprava od súradnice [1,2]. V tomto prípade pôjdu zápasy nasledovne [1,2], [2,3], [3,4], [4,5], [1,3], [2,4], [3,5], [1,4], [2,5], [1,5].

Pri veľkom počte tímov by sa muselo pri potrebe hrať zápasy každý s každým odohrať exponenciálny počet zápasov, čo by nebolo možné kvôli množstvu potrebného času na ich odohranie. Preto pri počte tímov viac ako sedem sú tímy rozdelené na dve skupiny a potom z každej postupujú prvé dva tímy a už hrajú vyraďovacím spôsobom semifinále nasledovne víťazi finále a porazení o 3. miesto.

3.3.6 Návrh hodnotiacich formulárov

Každá kategória má iné hodnotiace formuláre, preto je potrebné aby aplikácia obsahovala pre jednotlivé kategórie definovateľné formuláre. Každým rokom sa hodnotiace formuláre menia, preto musia byť tiež ľahko definovateľné. Každé políčko vo formulári môže byť vynásobené definovaným koeficientom. Formuláre budú definované v XML súboroch, pomocou ktorých sa vygeneruje formulár na zapisovanie jednotlivých položiek hodnotenia. Pri hodnotení bude na výber kategória, tím, kolo, rozhodca. Jednou z častí hodnotenia je aj čas, prípadne poznámka rozhodcu. Definované XML bude serializovateľné na objekt, ktorý bude mať nadefinované doplňujúce metódy, funkcie a validácie, pretože bude naviazaný na formulár s možnosťou vyplnenia jednotlivých položiek. Pri zobrazení sa využije definovanie dátovej šablóny vo WPF. Na obrázku 3.10 je znázornená objektová reprezentácia zadaného formulára. Trieda ResultForm obahuje zoznam políčok vo formulári.



Obrázok 3.10: Objektová reprezentácia hodnotiaceho formulára.

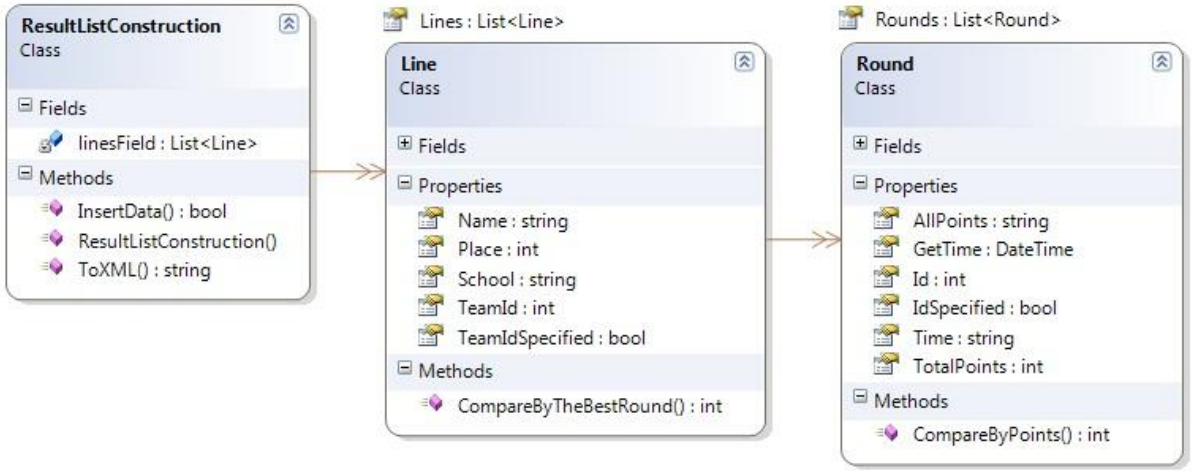
Nadefinovaný formulár sa zobrazí pomocou nadefinovanej dátovej šablóny vo WPF s nastavenými dátovými väzbami. Po zadaní dát užívateľom a potvrdením sa uložia do databázy. Jednotlivé body budú v jednom reťazci oddelené čiarkou, ktoré sa pri spätnom načítaní budú rozdeľovať tiež podľa v XML definovanom formulári.

3.3.7 Export výsledného poradia

Poslednou časťou návrhu je zoradenie tímov do výsledkových listín podľa získaných bodov. Každá kategória má iné kritériá hodnotenia, preto sa nedá navrhnúť jednotné vyhodnotenie všetkých kategórií. Navrhli sme teda spôsob ako vyhodnotíme jednotlivé kategórie s tým, že vyhodnotenie má definované pre každú kategóriu iné objekty. Jeden objekt je reprezentácia tímu vo výslednom poradí s tým, že tento objekt obsahuje všetky údaje potrebné pre triedenie. Hlavná myšlienka bude triedenie objektov podľa definovaných porovnávacích funkcií týchto objektov. Jednotlivé kategórie sa budú zoraďovať podľa nasledujúcich kritérií:

- Konštrukcia – bude sa brať do úvahy prvých N najlepších kôl. Ak sa niektoré tímy zhodujú v počte bodov najlepšieho kola, rozhodne ďalšie kolo v poradí. V prípade rovnosti všetkých kôl rozhoduje čas.
- Záchranár – rozhoduje bodové maximum zo všetkých kôl, v prípade rovnosti rozhoduje čas.
- Tanec – zohľadňuje sa súčet bodov od všetkých rozhodcov.
- Futbal – poradie sa určuje podľa bodovej tabuľky. V prípade rovnosti rozhoduje lepšie skóre.

Na obrázku 3.11 je znázornená objektová reprezentácia výsledného poradia pre kategóriu konštrukcia. Objekt ResultListConstruction obsahuje zoznam riadkov výsledného poradia Lines. Každý riadok v zozname Lines má zoznam kôl Rounds s počtami bodov a s časmi. Objekty Round a Line majú definované porovnávacie funkcie, ktoré umožňujú určiť poradie objektov. Usporiadať jednotlivé zoznamy Rounds a Lines je možné vykonať cez metódu Sort() definovanú v objekte List, kde ako parameter vložíme statickú porovnávajúcu funkciu. V objekte Round je porovnávacia funkcia CompareByPoints a v objekte Line CompareByBestRound. Objekt ResultListConstruction má definovanú funckiu ToXML(), ktorá nám po utriedení vráti XML reprezentáciu výsledného poradia. Cez XSL transformáciu vytvoríme následne zo získaného XML výsledné PDF určené pre tlač.



Obrázok 3.11: Triedny diagram pre výsledné poradie kategórie konštrukcia.

3.4 Realizácia

3.4.1 Web aplikácia

Implementácia web aplikácie nebola náročná úloha. Ako grafický dizajn sme zvolili voľne dostupný webový template, ktorý sme upravili pre naše potreby. Pri implementácii sme použili HTML, CSS a PHP. Na webovej stránke boli umiestnené aj informácie o priebehu súťaže a pravidlá jednotlivých kategórií. Pri programovaní sme sa nestretli s výraznými problémami a web aplikácia bola implementovaná ešte pred začatím súťaže, tak ju bolo možné otestovať aj v praxi. V praxi sa ukázalo, že budú potrebné zmeny pre lepšie použitie.

- Zmeniť zadávanie členov tímu a vyžadovať aj zadanie dátumu narodenia pre automatický výpočet vekovej kategórie. Zadávanie bolo riešené prostredníctvom jedného textového poľa, kde sa zadajú všetci členovia tímu.
- Pre ubytovanie a stravu umožniť zadať počet, pretože niektorí účastníci boli aj vo viacerých tímoch, alebo si mohli stravu priniesť, prípadne chceli byť ubytovaní na inom mieste. Zadanie potreby ubytovania a stravy bolo riešené len zaškrtávacím tlačidlom, preto sa po zaškrtnutí počítali do výsledného počtu všetci členovia tímu.

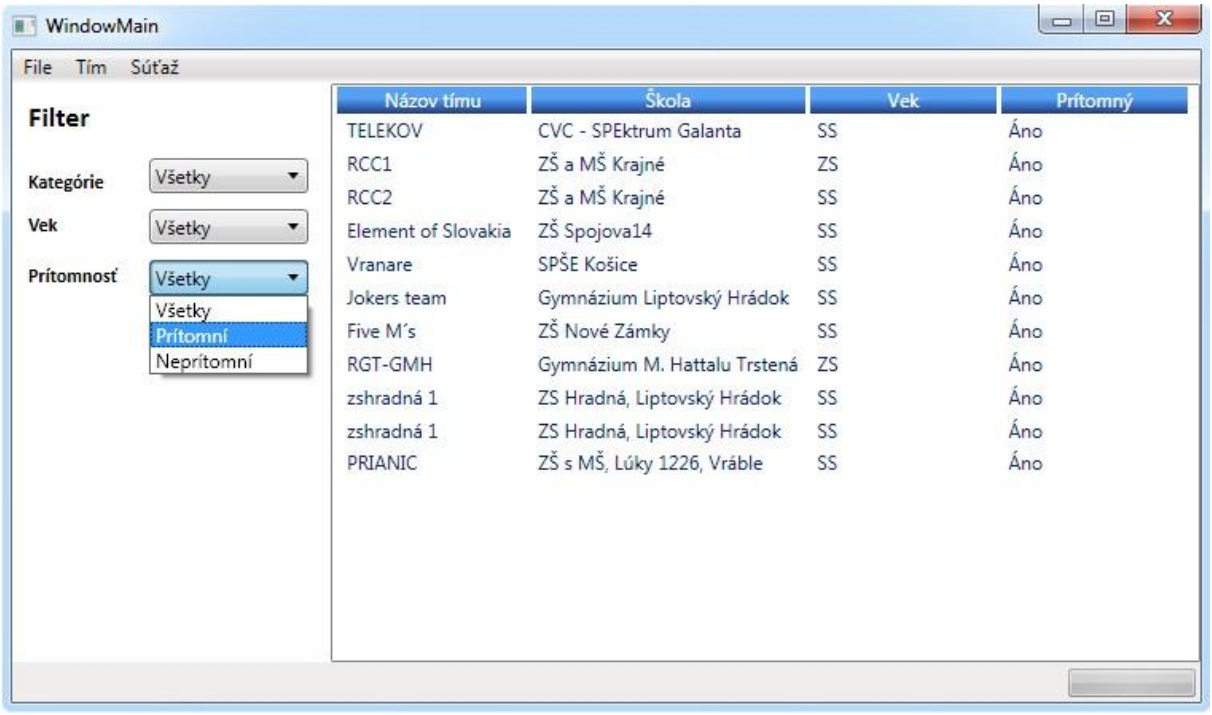
Výsledná web aplikácia je umiestnená na adrese www.robotika.sk/rcj.

3.4.2 Desktopová aplikácia

Počas realizácie sme použili technológie a návrh popísaný v predchádzajúcich kapitolách. Ako vývojové nástroje nám poslúžili Visual Studio a SQL Management Studio. Pri realizácií nám výrazne pomohla MVVM architektúra pre prehľadnú prácu s narastajúcimi časťami aplikácie a riadkami kódu.

3.4.2.1 Užívateľské rozhranie

Pri zostavovaní užívateľského rozhrania sme sa zamerali na jednoduchosť a rýchlosť použitia. Nebol však priestor plne využiť grafické možnosti WPF. Na obrázku 4.1 je znázornená časť užívateľského rozhrania aplikácie, konkrétne hlavného okna.



Obrázok 4.1: Užívateľské rozhranie hlavného okna aplikácie.

3.4.2.2 Filtrovanie údajov a multithreading

V hlavnom okne aplikácie je možné nastavením filtra zobrazovať potrebné údaje. Pri fitrovaní nastal problém, že fitrovanie trvalo nejakú dobu, hlavne pre potrebu výpočtu do akej vekovej kategórie jednotlivé tímy patria. Počas tejto doby sa aplikácia zasekla, preto bolo nutné použiť viac vlákien. Pri potrebe niečo vykonať v inom vlákne sme využili .NET knižnicu BackgroundWorker. Táto trieda má nadefinované jednotlivé EventHandlery. EventHandlerom sme priradili metódy, ktoré sa majú vykonať. Metóda, ktorá obsahuje kód učený pre vykonanie v druhom vlákne, je poskytnutá EventHandleru DoWork. Na zisťovanie stavu druhého vlákna a zobrazovanie tohto stavu poskytneme EventHandleru ProgressChangedEventHandler metódu, ktorá môže pracovať s poslanými údajmi z druhého vlákna a zobrazovať tak stav práce vykonávanej v druhom vlákne. Druhé vlákno sa spustí metódou RunWorkerAsync().

```
BackgroundWorker backgroundWorker = new BackgroundWorker();

backgroundWorker.DoWork +=
   new DoWorkEventHandler(backgroundWorker_DoWork);
backgroundWorker.ProgressChanged +=
   new ProgressChangedEventHandler(backgroundWorker_ProgressChanged);
backgroundWorker.RunWorkerCompleted +=
   new RunWorkerCompletedEventHandler(backgroundWorker_RunWorkerCompleted);

backgroundWorker.RunWorkerAsync();
```

Na ukázanie, že pracuje druhé vlákno sme použili ProgressBar. S využitím druhého vlákna sa pri zdĺhavejších operáciách aplikácia nezasekne a dá sa s ňou popri tom pracovať.

3.4.2.3 Logovanie aplikácie

Pri rôznych operáciách sme sa stretli s neočakávanými chybami. Najväčšia časť chýb sa vyskytla pri práci s databázou, alebo pri parsovaní dát s XML. Pri testovaní aplikácie bolo nutné úspešné a neúspešné operácie zachytiť, aby boli zaznamenané a pri pozeraní záznamov sme mohli ľahšie chyby odhaliť a to, v ktorých častiach aplikácie nastali. Na zápis záznamov sme využili knižnicu Log4Net. Táto knižnica je určená na logovanie operácií a ukladá ich do formátu XML. Kontrolovaním logu aplikácie sme ďalej kontrolovali korektnosť operácií a možné chyby, ktoré mohli v špeciálnych prípadoch nastať.

3.4.2.4 Serializácia a deserializácia

Pod serializáciou rozumieme uloženie stavu objektov do súboru. Ako štruktúra sa využíva XML. Opačný proces je deserializácia, kedy z XML súboru vytvoríme pomocou tried .NET objekty. Serializáciu a deserializáciu sme využili najmä pri čítaní dát z XML súborov a pri generovaní XML pre následné XSL transformácie. Tým sme sa vyhli zdĺhavému parsovaniu XML súborov.
Na nasledujúcom príklade je ukázané logovanie a deserializácia dát.

```
//inicializácia loggera
public static ILogger logger = Logger.GetLogger();
//funkcia na vrátenie objektovej reprezenácie dát pre import
private XmlWebData GetDeserializedData(string path)
{
   XmlWebData result = new XmlWebData();
   try
   {
      XmlSerializer serializer = new XmlSerializer(typeof(XmlWebData));
      using (StreamReader reader = new StreamReader(path))
      {
         //deserializovanie dát
         result = (XmlWebData)serializer.Deserialize(reader);
         //zapísanie správy do logu
         logger.Trace("Data z xml deserializovane.");
      };
   }
   catch (Exception ee)
   {
         //zapísanie správy a výnimky do logu
         logger.Error("Chyba pri deserializovani dat", ee);
   }
   return result;
}
```

3.4.2.5 Generovanie PDF

Pre každé generovanie PDF sme si nadefinovali serializovateľné objekty. Tie sme následne serializovali do XML a s použitím nadefinovaných transformačných súborov XSL a .NET knižníc vygeneroval výsledné PDF. Na nasledujúcom príklade je ukážka implementácie generovania PDF.

```
XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load("StartOrderFo.xsl");
//transformácia do Fo dokumentum ktorý sa uloží na disk
xslt.Transform("StartOrder.xml", StartOrder.fo");
//inicializácia objektu pre vytvorenie PDF z knižnice Fonet.dll
FonetDriver driver = FonetDriver.Make();
//vytvorenie PDF, ktoré sa zapíše na disk
driver.Render("StartOrder.fo", "StartOrder.pdf");
```

4 Záver

V prvej kapitole sme popísali nové technológie a princípy pri vývoji aplikácií. Pri technológiách sme sa zamerali na základný popis Windows Presentation Foundation a ADO.NET Entity Frameworku. Popísali sme ich vlastnosti, využitie, výhody a nevýhody. V ďalšej časti kapitoly sme sa zamerali na architektúru aplikácií používanú pri týchto technológiách Model – View – ViewModel. Táto architektúra prináša veľa výhod pri vyvíjaní softvéru a je navrhnutá a určená špeciálne pre Windows Presentation Foundation.

Teoretické poznatky sme využili pri následnom návrhu a implementácii. Preskúmaním priebehu a pravidiel súťaže Robocup Junior Slovensko sme definovali požiadavky. Pre ich definovanie bola veľmi dôležitá aj osobná účasť na samotnej súťaži, počas ktorej sme zistili potreby pre organizátorov. V návrhu sme kládli dôraz na použiteľnosť, flexibilnosť a možnosť ďalšieho rozširovania. Flexibilnosť sme zaručili možnosťou definovania formulárov pre vyhodnocovanie, čím je umožnené prispôsobiť sa na zmeny v budúcnosti. Rozšíriteľnosť a úpravy aplikácie sú jednoduchšie pre použitú architektúru, ktorá rozdeľuje aplikáciu na jednotlivé logické celky a oddeľuje dizajn od logiky aplikácie. Pri exportoch sú definované transformácie v jednotlivých súboroch a nie sú napísané priamo v kóde aplikácie, čo znamená, že pri zmenách pri zobrazovaní dát nie je potrebné meniť kód aplikácie. Počas implementácie sme si osvojili použitie popísaných technológií a princípov pri vývoji a zistili výhody a nevýhody spojené s ich použitím.

Výsledkom práce bola implementácia web aplikácie, ktorá bola použitá v praxi a väčšia časť desktopovej aplikácie, ktorá môže v budúcnosti veľmi pomôcť organizátorom počas priebehu súťaže.

Literatúra

[1] Pro WPF in C# 2008 Second Edition, Matthew MacDonald, 2008

[2] WPF Programmer's Reference, Rod Stephens, 2010

[3] Mistrovství ve Windows Presentation Foundation, Charles Petzold, 2008

[4] C# 2008 Programujeme profesionálně, Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, 2008

[5] Programming .NET 3.5, Jesse Liberty, Alex Horovitz, 2008

[6] Programming Entity Framework, Julia Lerman, 2009

[7] Pro LINQ Object Relational Mapping in C# 2008, Vijay P. Mehta, 2008

[8] XSLT: Mastering XML Transformations, Doug Tidwell

[9] Inside XSLT, Steven Holzner, 2001

[10] XML Schema, Eric van der Vlist, 2002

[11] .NET Framework Conceptual Overview
http://msdn.microsoft.com/library/zw4w595w.aspx

[12] WPF Apps With The Model-View-ViewModel Design Pattern
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[13] Using LINQ to SQL
http://weblogs.asp.net/scottgu/archive/2007/05/19/using-linq-to-sql-part-1.aspx

[14] Introduction to WPF
http://msdn.microsoft.com/en-us/library/aa970268.aspx

[15] Apache Logging Services
http://logging.apache.org/

[16] Use XSL-FO for page breaks and tables
http://www.ibm.com/developerworks/xml/library/x-tippgbk/index.html

[17] A Guided Tour of WPF
http://joshsmithonwpf.wordpress.com/a-guided-tour-of-wpf/

[18] Model View ViewModel
http://en.wikipedia.org/wiki/Model_View_ViewModel

[19] BackgroundWorker Threads and Supporting Cancel
http://www.codeproject.com/KB/cpp/BackgroundWorker_Threads.aspx

[20] FO.NET
http://fonet.codeplex.com/

[21] LINQ to Entities
http://msdn.microsoft.com/en-us/library/bb386964.aspx