

# Srovnání návrhových vzorů uživatelského rozhraní pro potřeby výuky

Antonín Spáčil

---

Bakalářská práce  
2016



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2015/2016

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Antonín Spáčil**  
Osobní číslo: **A12059**  
Studijní program: **B3902 Inženýrská informatika**  
Studijní obor: **Informační a řídicí technologie**  
Forma studia: **prezenční**

Téma práce: **Srovnání návrhových vzorů uživatelského rozhraní pro potřeby výuky**

Téma anglicky: **Graphical User Interface Design Patterns Comparison for Education Purposes**

Zásady pro vypracování:

1. Vypracujte literární rešerši na téma návrhových vzorů uživatelského rozhraní.
2. Analyzujte možnosti tvorby aplikací s využitím vzorů MVVM, MVP a MVC především s využitím aplikační platformy .NET.
3. Vytvořte ukázkové implementace aplikace ve vybraných návrhových vzorech.
4. Na základě ukázkových implementací navrhnete zadání úkolů pro potřeby výuky.
5. Demonstrujte výsledky.

Rozsah bakalářské práce: -  
Rozsah příloh: -  
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, 2003, xxiv, 533 s. The Addison-Wesley Signature Series. ISBN 978-0-321-12742-6.**
2. **NAGEL, Christian, Jay GLYNN a Morgan SKINNER. Professional C# 5.0 and .NET 4.5.1. Indianapolis, IN: John Wiley and Sons, 2014, 1 online zdroj (1563 pages). ISBN 978-1-118-83294-3.**
3. **GAROFALO, Raffaele. Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern. Sebastopol, Calif: O'Reilly Media, 2011. ISBN 978-073-5650-923**
4. **GALLOWAY, Jon. Professional ASP. NET MVC 5. Indianapolis, Indiana: John Wiley & Sons, Inc., 2014, 1 online zdroj (622 pages). ISBN 978-1-118-79476-0.**
5. **Knockoutjs [online]. knockoutjs.com, 2016 [cit. 2016-01-25]. Dostupné z: <http://knockoutjs.com/>**

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**  
Ústav počítačových a komunikačních systémů  
Datum zadání bakalářské práce: **19. února 2016**  
Termín odevzdání bakalářské práce: **27. května 2016**

Ve Zlíně dne 19. února 2016



doc. Mgr. Milan Adámek, Ph.D.  
*děkan*



prof. Ing. Vladimír Vašek, CSc.  
*ředitel ústavu*


### **Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

### **Prohlašuji,**

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

  
.....  
půdpis diplomanta

## **ABSTRAKT**

Hlavním cílem této práce je srovnání návrhových vzorů uživatelského rozhraní pro využití ve výuce souvisejících předmětů. V teoretické části je popsána platforma .NET a různé návrhové vzory uživatelského rozhraní. V praktické části jsem vytvořil aplikace s použitím návrhových vzorů popsaných v teoretické části. K aplikacím jsem navrhl několik úkolů pro potřeby výuky.

Klíčová slova: MVC, MVVM, MVP, Passive View, návrhové vzory, Windows Forms, .NET, C#

## **ABSTRACT**

Main goal of this bachelor thesis is comparison of design patterns of user interface for use in teaching of related subjects. In the theoretical section is described platform .NET and different design patterns of user interface. In practical part i have created applications with use of design patterns described in theoretical section. With applications i suggested few assignments for educational purposes.

Keywords: MVC, MVVM, MVP, Passive View, design patterns, Windows Forms, .NET, C#

Chtěl bych poděkovat svému vedoucímu Ing. et Ing. Eriku Královi, Ph.D za poskytnutí materiálu, věnování času a vedení při tvorbě této bakalářské práce.

# OBSAH

<b>ÚVOD</b> .....	<b>9</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>10</b>
<b>1 .NET FRAMEWORK</b> .....	<b>11</b>
1.1 PROGRAMOVACÍ JAZYKY.....	11
1.2 VISUAL C#.....	12
1.3 LINQ A LAMBDA.....	12
1.3.1 Lambda.....	13
1.3.2 LINQ.....	13
1.4 KOMPATIBILITA APLIKACÍ.....	14
1.5 DOSTUPNOST.....	14
1.6 VERZE .NET FRAMEWORK.....	15
1.7 COMMON LANGUAGE RUNTIME.....	16
1.7.1 Garbage collector.....	17
1.8 PROJEKT MONO.....	17
1.8.1 Podporované operační systémy.....	17
<b>2 NÁVRHOVÉ VZORY</b> .....	<b>17</b>
2.1 WINDOWS FORMS.....	18
2.2 WPF – WINDOWS PRESENTATION FOUNDATION.....	19
2.3 MVC – MODEL VIEW CONTROLLER.....	19
2.3.1 Model.....	20
2.3.2 View.....	20
2.3.3 Controller.....	20
2.3.4 Počátky MVC.....	21
2.4 MVVM – MODEL VIEW VIEWMODEL.....	21
2.4.1 Model.....	21
2.4.2 View.....	21
2.4.3 ViewModel.....	21
2.5 MVP – MODEL VIEW PRESENTER.....	22
2.5.1 Passive View.....	22
2.5.2 Supervising Controller.....	22
<b>II PRAKTICKÁ ČÁST</b> .....	<b>23</b>
<b>3 UKÁZKOVÉ APLIKACE IMPLEMENTUJÍCÍ RŮZNÉ NÁVRHOVÉ VZORY</b> .....	<b>24</b>
<b>4 STANDARDNÍ WINDOWS FORMS APLIKACE</b> .....	<b>25</b>
4.1 POPIS APLIKACE.....	25
4.2 NÁVRH A APLIKAČNÍ LOGIKA.....	25
4.2.1 Pomocné třídy v aplikaci.....	25
4.2.2 Formuláře, ze kterých se skládá aplikace.....	26
4.2.3 Diagram tříd.....	27
<b>5 WINDOWS FORMS APLIKACE S VYUŽITÍM NÁVRHOVÉHO VZORU PASSIVE VIEW</b> .....	<b>28</b>

5.1	POPIS APLIKACE .....	28
5.2	NÁVRH A APLIKAČNÍ LOGIKA .....	28
5.2.1	Model .....	28
5.2.2	View .....	28
5.2.3	Controller .....	29
5.2.4	Úprava záznamu .....	30
5.2.5	Diagram tříd .....	32
<b>6</b>	<b>ASP .NET MVC APLIKACE.....</b>	<b>33</b>
6.1	POPIS APLIKACE .....	33
6.2	NÁVRH A APLIKAČNÍ LOGIKA .....	33
6.2.1	Modely .....	33
6.2.2	View .....	33
6.2.3	Controller .....	35
6.2.4	Diagram tříd .....	37
<b>7</b>	<b>MVVM APLIKACE.....</b>	<b>38</b>
7.1	POPIS APLIKACE.....	38
7.1.1	Model .....	38
7.1.2	View .....	38
7.1.3	ViewModel.....	39
7.1.4	Diagram tříd .....	40
<b>8</b>	<b>ÚKOLY PRO POTŘEBY VÝUKY.....</b>	<b>41</b>
	<b>ZÁVĚR .....</b>	<b>42</b>
	<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>43</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>	<b>46</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>47</b>
	<b>SEZNAM TABULEK.....</b>	<b>48</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>49</b>

## ÚVOD

V dnešní době je na kvalitní vývoj aplikací kladen vysoký důraz. Aplikace musí být nejen bezchybné, ale také snadno udržovatelné a u rozsáhlých systémů vytvořené tak, aby na nich mohlo pracovat více vývojářů. Z těchto důvodů je při vývoji aplikací důležitou částí také výběr vzoru, podle kterého se bude aplikace vytvářet.

V této práci jsou popsány základní návrhové vzory, které se dnes používají při návrhu aplikací a které usnadňují práci softwarových vývojářů. Práce obsahuje demonstrační aplikace vytvořené v několika návrhových vzorech.

V teoretické části této práce je popsána platforma .NET, její verze, LINQ a lambda výrazy, programovací jazyk C# a Projekt MONO. Jsou zde popsány návrhové vzory MVC, MVP a MVVM. Také je zde popsán framework Windows Forms, který je součástí .NET a slouží pro vývoj desktopových aplikací.

V praktické části této práce jsem vytvořil několik ukázkových aplikací s použitím návrhových vzorů z teoretické části této práce. Je zde desktopová aplikace demonstrující vývoj ve Windows Forms bez využití specifického návrhového vzoru, dále je zde desktopová aplikace vytvořená ve Windows Forms demonstrující využití návrhového vzoru Passive View. Další z aplikací je webová aplikace demonstrující návrhový vzor MVC. Poslední aplikací vytvořenou v rámci této práce je desktopová aplikace vytvořená ve frameworku Windows Presentation Foundation s využitím návrhového vzoru MVVM.

Práce obsahuje několik zadání úkolů pro potřeby výuky týkajících se demonstračních aplikací.

## **I. TEORETICKÁ ČÁST**

## 1 .NET FRAMEWORK

Rozhraní .NET framework je softwarová platforma, vyvinutá společností Microsoft, určená pro podporu vývoje a spuštění aplikací na jejich produktech.

.NET framework se skládá z rozsáhlé knihovny tříd, která umožňuje programátorům snadnější vývoj aplikací díky přístupu ke kódu pro všechny hlavní oblasti vývoje a z common language runtime (CLR), což je run-time prostředí, které se stará o spuštěné aplikace a poskytuje jim systémové služby. Pro vývoj aplikací na .NET frameworku poskytuje Microsoft vývojové prostředí Visual studio v několika variantách, mezi které patří také verze express, která je zdarma i pro komerční použití. [1]

.NET framework stále prochází vývojem a od jeho první verze do něj bylo již přidáno několik důležitých technologií, jako je například LINQ, lambda, nebo formulářová technologie WPF.

### 1.1 Programovací jazyky

Platforma .NET framework podporuje velké množství programovacích jazyků. Aplikace ve všech podporovaných jazycích jsou po spuštění přeloženy do společného jazyka Common Intermediate Language (CIL), což umožňuje vzájemnou kompatibilitu mezi aplikacemi psanými v různých jazycích podporovaných .NET frameworkem. I přes velké množství podporovaných jazyků není žádný považován za primární. [2]

Mezi podporované jazyky patří tyto:

- Visual Basic .NET
- Visual C# .NET
- Visual C++ .NET
- Transact-SQL
- Windows Script Host

- VBScript
- JScript
- JScript .NET
- J++
- Extensible Markup Language (XML)

Všechny tyto jazyky jsou vlastně implementací originálního jazyka obohacenou o funkcionality .NET frameworku, tudíž se nejedná o originální jazyk, ale o jeho obdobu.

Například Visual C++ je C++ obohacené o funkcionality .NET frameworku a s přístupem ke knihovně v něm obsaženým. Kód napsaný v jazyce C++ tedy bude fungovat i ve Visual C++, protože jsou zde obsaženy všechny funkcionality C++, ale kód napsaný ve Visual C++ již nemusí v „klasickém“ C++ fungovat, protože v tomto kódu mohou být obsaženy třídy z .NET frameworku a také se dá předpokládat že takový kód nebude mít programátorem nedefinovanou správu paměti o kterou se ve Visual C++ stará CLR.

## 1.2 Visual C#

Visual C# je vysokoúrovňový, objektově orientovaný programovací jazyk vytvořený společností Microsoft určený pro tvorbu aplikací běžících na platformě .NET. C# vychází z rodiny programovacích jazyků C. [3]

Jazyk C# je značně podobný jazykům Java, C a C++, obsahuje ale zjednodušení některých jejich vlastností, jako je například přímý přístup do paměti, který není k dispozici v jazyce Java nebo delegáty, lambda výrazy či null datové typy, které nejsou k dispozici v C++.

## 1.3 LINQ a lambda

V novějších verzích .NET frameworku jsou obsaženy technologie LINQ a lambda, které zjednodušují psaní dotazů a funkcí v C# nebo Visual Basic. Tyto technologie jsou na sobě nezávislé, ale při vývoji se často kombinují při práci s datovými kolekcemi.

### 1.3.1 Lambda

Lambda výrazy jsou anonymní funkce, používané převážně k vytváření delegátů a k psaní LINQ dotazů. Lambda výraz vypadá například takto:  $x \Rightarrow x == I$ , kde  $x$  je vstupní parametr a  $x == I$  je vyhodnocovaný výraz. [4]

Pro ilustraci je přiložena ukázka kódu vytvářející anonymní funkci obsahující lambda výraz a její volání.

```
Func<int, int> func1 = x => x + 1;  
int result = func1.Invoke(2);
```

*Kód 1. Anonymní funkce s lambda výrazem*

Datové typy v deklaraci anonymní funkce určují vstupní a výstupní datový typ. Funkce v ukázce výše navrácí vstupní hodnotu navýšenou o 1.

### 1.3.2 LINQ

LINQ (Language Integrated Query) je technologie přidaná do .NET od verze 3.5, která umožňuje vytvářet dotazy nad datovými kolekcemi. LINQ dotazy nad datovými kolekcemi lze rozdělit do několika skupin, mezi které patří například tyto: [5]

- Omezovací (Where)
- Selektovací (Select, SelectMany)
- Rozdělovací (Take, Skip, TakeWhile, SkipWhile)

Pro ilustraci jsou přiloženy ukázky kódu v LINQ, které implementují nalezení sudých čísel v datové kolekci list obsahující čísla od 1 do 9 s použitím standardní linq syntaxe a s použitím lambda výrazu.

```
List<int> numbers= new List<int>() { 1,2,3,4,5,6,7,8,9 };

var evenNumbers =
    from num in numbers
    where (num % 2) == 0
    select num;

List<int> evenNumbersList = evenNumbers.ToList();
```

*Kód 2. Vybrání sudých čísel z listu pomocí standardní LINQ syntaxe*

```
List<int> numbers= new List<int>() { 1,2,3,4,5,6,7,8,9 };

List<int> evenNumbersLambda = numbers.Where(p => p % 2 == 0).ToList();
```

*Kód 3 – vybrání sudých čísel z listu pomocí LINQ s využitím lambda výrazu*

Výstupní list z obou ukázek výše obsazuje stejné hodnoty, tedy pouze sudá čísla ze vstupního listu.

## 1.4 Kompatibilita aplikací

Jedním z cílů .NET frameworku je kompatibilita verzí. Aplikace vytvořené na starších verzích .NET frameworku by až na vzácné výjimky měly fungovat i na nových verzích.

## 1.5 Dostupnost

.NET framework byl donedávna dostupný pouze pro produkty společnosti Microsoft, ke konci roku 2014 bylo však oznámeno, že framework bude se všemi svými součástmi open source, což je stav, ve kterém se nachází nyní. Tato dřívější nedostupnost .NET frameworku na cizí platformy vedla ke vzniku multiplatformních open source projektů implementujících .NET framework. Mezi tyto projekty patří například projekt Mono nebo projekt DotGNU Portable.NET. [6]

Různé verze .NET frameworku se nachází v různých verzích operačních systémů Windows nebo ve vývojovém prostředí Visual Studio. .NET framework je možné do operačního systému doinstalovat, pokud je daná verze frameworku podporována danou verzí operačního systému.

## 1.6 Verze .NET framework

Tabulka 1 - Verze .NET frameworku [7]

<i>Verze .NET framework</i>	<i>Obsaženo ve Windows</i>	<i>Možné doinstalovat do Windows</i>	<i>Přidané funkce</i>	<i>Obsaženo ve verzi Visual Studio</i>	<i>Verze CLR</i>
1.0	-	-	První verze .NET framework	Visual Studio .NET	1.0
1.1	-	-	Side by side provádění kódu	2003	1.1
2.0	-	-	Generické datové typy	2005	2.0
3.0	Vista	-	WPF, WCF, WF	-	2.0
3.5	7, 8, 8.1	Vista	LINQ, AJAX podporované webstránky	2008	2.0
4	-	7, Vista	Rozšíření knihoven, přenosné knihovny, vývoj na dalších Microsoft platformách	2010	4
4.5	8	7, Vista	Podpora pro Windows Store aplikace, aktualizace některých součástí (WPF, ASP.NET)	2012	4
4.5.1	8.1	8, 7, Vista	.NET Native, Podpora pro telefoní Windows Store aplikace	2013	4
4.5.2	-	8.1, 8, 7,	Vylepšení debugingu, nová API pro ASP.NET	-	4

<i>Verze .NET framework</i>	<i>Obsaženo ve Windows</i>	<i>Možné doinstalovat do Windows</i>	<i>Přidané funkce</i>	<i>Obsaženo ve verzi Visual Studio</i>	<i>Verze CLR</i>
		Vista	aplikace		
4.6	10	8.1, 8, 7, Vista	ASP .NET Core 5, vylepšení trasování událostí	-	4
4.6.1	10 - listopadový update	10, 8.1, 8, 7, Vista	Podpora pro certifikáty X509	-	4
4.6.2	10 - Preview Build 14295	10, 8.1, 8, 7, Vista	Vylepšení pro WPF	-	4

## 1.7 Common language runtime

Common language runtime je run-time prostředí, které poskytuje vývojáři systémové služby .NET frameworku. CLR pracuje s metadaty, což jsou informace o struktuře programu, které generuje kompilátor. Metadata obsahují například informace o zdrojích, na které komponenty odkazují, což vede k tomu, že se runtime stará o to, aby aplikace měla veškeré zdroje v potřebných verzích. CLR také umožňuje snadný vývoj mezi různými programovacími jazyky podporovanými platformou .NET, kdy objekty napsané v jednom programovacím jazyce mohou komunikovat s objekty napsanými v jiném programovacím jazyce. Například lze napsat třídu v jednom programovacím jazyce a použít ji v aplikaci psané v jiném programovacím jazyce. [8]

CLR automaticky spravuje odkazy mezi objekty a uvolňuje je, když nejsou používány a také se stará o správu paměti. Při vytváření nových objektů se CLR postará o alokaci paměti a následně se stará o její uvolnění pomocí garbage collectoru, který také zajišťuje ochranu před memory leaky. Vývojáři tedy odpadá práce se správou paměti, kterou musí

řešit například v jazyce C.

### 1.7.1 Garbage collector

Garbage collector je nástroj obsažený v CLR, který se stará o správu paměti programu na spravované haldě, což je část paměti, kterou si alokuje garbage collector pro správu objektů. Garbage collector zajišťuje alokaci paměti pro nové objekty a její uvolnění u objektů, které již nejsou používány, tato alokace probíhá na spravované haldě. Uvolňování paměti probíhá automaticky na pozadí běhu aplikace, vývojář ho ale může také vyvolat manuálně. [9]

## 1.8 Projekt Mono

Mono je open source vývojářská platforma založená na .NET frameworku, umožňující multiplatformní programování v jazyce C#. Hlavní předností Mono je jeho multiplatformnost a to jak pro 32 tak pro 64 bitové operační systémy. [10]

### 1.8.1 Podporované operační systémy

- Linux
- Mac OS X, iPhone OS
- Sun Solaris
- BSD – OpenBSD, FreeBSD, NetBSD
- Microsoft Windows
- Nintendo Wii
- Sony Playstation 3

## 2 NÁVRHOVÉ VZORY

Návrhový vzor je předpis postupu, který je použit při návrhu a tvorbě aplikace. Softwareo-

vé návrhové vzory nejsou vázány na konkrétní programovací jazyky, jedná se o obecně předpisy tvorby aplikací.

## 2.1 Windows Forms

Windows forms je frameworkem .NET, který umožňuje vytváření formulářových aplikací. GUI těchto aplikací je zpravidla navrhováno v designeru, což je grafické prostředí, které umožňuje vývojáři navrhnout aplikaci pomocí rozsáhlého množství nabízených komponent přímým přesouváním komponent na daná místa ve výsledném formuláři. Vývojář zde ale není omezen pouze nabídkou komponent z vývojového prostředí, může si vytvořit své vlastní komponenty nebo použít externí knihovny s těmito komponenty, jako je například C1, tedy ComponentOne. [11]

Samotný formulář i komponenty mají vlastnosti, pomocí kterých je možné nastavit jejich vzhled a chování a také události, které umožňují vývojáři reagovat na interakci uživatele aplikace s jednotlivými komponentami. Mezi události patří například kliknutí myši, změna textu, změna viditelnosti komponenty, zmáčknutí tlačítka myši, puštění tlačítka myši a podobné. Mezi vlastnosti pak patří například text, barva, název, viditelnost a podobně. Designer nám umožňuje události a vlastnosti pro konkrétní komponenty nastavit.

Designer je ale pouze nástroj, který převede vývojářův grafický návrh na kód. Proto je možné pracovat úplně bez něj a vytvořit GUI pouze kódově, což je ale ve většině případů zbytečné, pokud máme k dispozici nástroj jako je designer, který nám tuto část vývoje windows form aplikací značně usnadňuje.

Samotná windows form aplikace se skládá z formulářů, které obsahují základní třídu, ve které vývojář přistupuje ke komponentám daného formuláře, pracuje s nimi a implementuje práci s událostmi, které jsou ve formuláři vyvolané. Na základě návrhového vzoru může být v této třídě implementovaná logika celého formuláře, tato třída může ale také sloužit pouze pro předávání parametrů formuláře do dalších tříd, které obstarávají logiku formuláře, případně celé aplikace. Následně formulář obsahuje třídu pro designer, která obsahuje kódovou reprezentaci grafického návrhu v designeru. Poslední částí, kterou standardní formulář obsahuje jsou zdrojové resx soubory, do kterých je možné ukládat zdroje pro daný formulář jako jsou obrázky nebo texty. Zdrojové soubory umožňují jednoduchou lokalizaci aplikace do více jazyků, kde formulář může obsahovat větší množství zdrojových

souborů které obsahují stejné záznamy pro volání z aplikace, ale pro každou jazykovou lokalizaci navrácí jiné hodnoty.

## 2.2 WPF – Windows Presentation Foundation

WPF je dalším z frameworků .NET pro tvorbu formulářových aplikací. Do .NET frameworku je přidán od verze 3.0 a jedná se o novější verzi formulářového frameworku než je windows form. [12]

Stejně jako u windows forms lze i u WPF vytvářet formuláře pomocí designeru. Narozdíl od windows forms ale nejsou pozice komponent v těchto formulářích ukládány absolutně, což přináší výhody hlavně u aplikací pro mobilní zařízení. Jednotka vzdálenosti v návrhu WPF aplikací není pixel, ale DIP (device independent unit), což je jednotka, která pracuje s DPI zobrazovacího zařízení, takže je možné snadno vyvíjet aplikace pro různá zobrazovací zařízení.

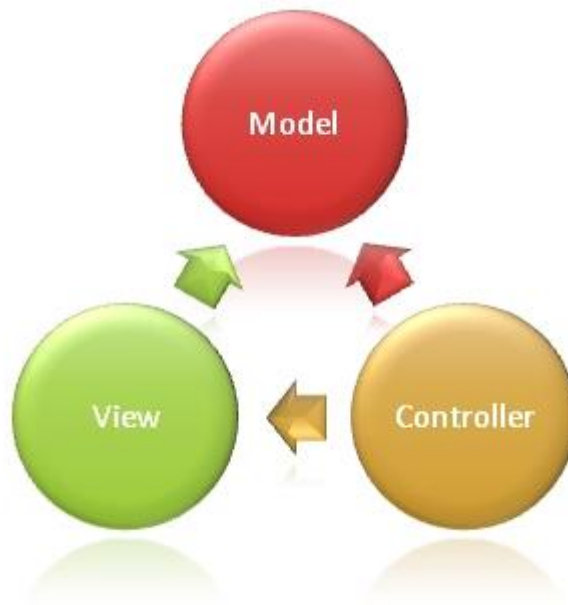
GUI WPF je tvořeno pomocí jazyka XAML, což je značkovací jazyk založený na XML.

Další výhodou WPF aplikací je jejich menší zatížení pro procesor, což je způsobeno tím, že pro vykreslování formulářů je použita technologie Direct3D, která pracuje s grafickou kartou.

## 2.3 MVC – Model View Controller

MVC je návrhový vzor, jehož myšlenkou je rozdělení aplikační logiky od uživatelského rozhraní. Tento návrhový vzor původně vznikl pro desktopové aplikace, používá se ale také u webových frameworků jako jsou např. Ruby on Rails nebo MVC pro ASP .NET. [13] [14]

MVC se dělí na 3 části: model, view a controller. Hlavními výhodami MVC je zpřehlednění kódu, který je rozdělen do několika částí, což vede k další výhodě, kterou je znovupoužitelnost kódu. To je způsobeno tím, že části kódu na sebe vzájemně pouze odkazují a nejsou tedy všechny obsaženy v jednom souboru. MVC také umožňuje snadné testování každé z hlavních částí aplikace zvlášť a tedy vede k snadnějšímu hledání a řešení problémů a celkově snadnější udržovatelnosti aplikace z vývojářského hlediska. [145]



Obrázek 1 – ASP .NET MVC diagram [16]

### 2.3.1 Model

Model reprezentuje data v aplikaci a její logiku. Model obsahuje definice datových objektů, práci s nimi, výpočty, validaci a podobně. Model nemá odkaz na view ani na controller, neví tedy odkud data která mu byla předána pochází. Není na view ani controlleru závislý a může tedy být použit pro různé view a různé controllery.

### 2.3.2 View

View reprezentuje uživatelské rozhraní aplikace, uživatelé jsou zde zobrazeni data z modelu a prvky pro interakci s aplikací jako jsou tlačítka, textová pole a podobně. View obsahuje minimální množství logiky. View má odkaz na model, aby mohlo zobrazovat data, která jsou v něm obsažena.

### 2.3.3 Controller

Controller reprezentuje tu část aplikace, která reaguje na vstup uživatele, tedy například na zmáčknutí tlačítka, nebo na parametry v URL. Controller tyto vstupy uživatele vyhodnocuje a následně předává do modelu, aby se v něm povedly požadované změny a ty mohly být ve view zobrazeny uživateli. Controller také rozhoduje, které view se uživateli zobrazí. Controller může pracovat s několika různými modely a několika různými view.

V MVC mezi sebou často také má vzájemnou vazbu controller a view kde controller rea-

guje na interakci uživatele s view.

#### 2.3.4 Počátky MVC

Návrhový vzor MVC formuloval v letech 1978 – 1979 Trygve Reenskaug a jeho hlavním účelem bylo poskytnout uživatelům rozhraní pro práci s více view s různými daty. Upravenou verzi Reenskaugova návrhu následně použil Xerox-PARC jako součást knihoven Smalltalk-80. [17]

### 2.4 MVVM – Model View ViewModel

MVVM je návrhový vzor vytvořený architekty Microsoftu Kenem Cooperem a Tedem Petersem pro framework WPF. Cílem MVVM je stejně jako u MVC oddělení prezentační a aplikační vrstvy. MVVM se skládá ze tří částí a to z view, modelu a viewmodelu. [18] [19]

Návrhový vzor MVVM není používán pouze pro WPF aplikace, ale je využit například v javascriptovém frameworku KnockoutJS. [20]

#### 2.4.1 Model

Model představuje stejně jako u MVC strukturu pro data v aplikaci a aplikační logiku.

#### 2.4.2 View

View představuje prezentační vrstvu aplikace, obsazuje prvky, se kterými v aplikaci uživatel interaguje, jako jsou tlačítka nebo textová pole. View neobsahuje žádný kód, pouze je mu přiřazen viewmodel, který se stará o zpracovávání dat z view a jejich zobrazování. Komunikace mezi view a viewmodelem je řešena pomocí bindingů, kde se ke konkrétním elementům ve view přiřadí konkrétní proměnná z viewmodelu. Bindingy se starají o aktualizaci view v případě, že jsou proměnná ve viewmodelu změněny. Vývojář tedy nemusí nastavovat žádné události reagující na změnu view, aby ji mohl předat do viewmodelu.

#### 2.4.3 ViewModel

ViewModel je vrstvou aplikace, která reaguje na uživatelskou interakci s view. Pomocí bindingu jsou do viewmodelu předány informace o uživatelské interakci s view. ViewModel se stará o zpracování těchto informací a reakci na ně. Pro každý viewmodel existuje pouze jedno view a naopak.

## 2.5 MVP – Model View Presenter

MVP je návrhový vzor podobný MVC. Model a view jsou v MVP stejné jako v MVC, rozdíl je v controlleru, v MVP nazývaný presenter. Pro každý presenter zde na rozdíl od MVC existuje pouze jedno view.

Podle Martina Fowlera je ale návrhový vzor MVP zastaralý a je třeba ho rozdělit na dva návrhové vzory a to Passive View a Supervising Controller. [21]

### 2.5.1 Passive View

Passive view je návrhový vzor velmi podobný MVC, skládá se stejně jako MVC z modelu, view a controlleru. Na rozdíl od standardního MVC zde ale není žádná vazba mezi view a modelem a samotné view je pasivní a není zodpovědné za jakékoliv vlastní změny. O ty se v tomto návrhovém vzoru kompletně stará controller, který předává do view data, která se mají zobrazit a od view přijímá informace o událostech vyvolaných uživatelem. [22]

### 2.5.2 Supervising Controller

Supervising controller rozděluje prezenční funkcionalitu na dvě části a to na view a na controller, často nazývaný presenter. Návrhový vzor také obsahuje model, který je stejný jako v případě MVC. [23]

Supervising controller má dvě primární odpovědnosti a to reagování na vstupy a synchronizaci modelu a view. Reagování na vstupy je řešeno pomocí komponent ve view, které předávají události vyvolané uživatelem do controlleru. Synchronizace view a modelu zpravidla používá data binding, který ale v případě komplexnějších interakcí uživatele nemusí stačit. V tomto případě se o synchronizaci view a modelu stará controller.

## **II. PRAKTICKÁ ČÁST**

### **3 UKÁZKOVÉ APLIKACE IMPLEMENTUJÍCÍ RŮZNÉ NÁVRHOVÉ VZORY**

V rámci praktické části této bakalářské práce bylo vytvořeno několik ukázkových aplikací implementujících různé návrhové vzory. Všechny tyto aplikace byly vytvořeny na platformě .NET ve vývojovém prostředí Microsoft Visual Studio Professional 2013.

## 4 STANDARDNÍ WINDOWS FORMS APLIKACE

První z aplikací vytvořených v rámci této bakalářské práce je windows forms aplikace bez využití návrhových vzorů. Aplikace demonstruje vývoj formulářových aplikací pomocí technologie windows forms bez využití návrhových vzorů popsaných v teoretické části této bakalářské práce.

### 4.1 Popis aplikace

Aplikace je jednoduchou evidencí příjmů a útrat. Uživatel v ní eviduje částky, poznámky k nim a data vytvoření. Samotná aplikace k těmto informacím dále přidává unikátní identifikátor guid, což je řetězec, skládající se z 32 hexadecimálních znaků.

Aplikace se skládá ze dvou formulářů. Všechny existující záznamy jsou uživateli zobrazeny v prvním formuláři aplikace, který také slouží k vkládání nových záznamů. Záznamy je možné z aplikace mazat nebo je upravovat. Při upravení záznamu je vyvolán druhý formulář pro úpravu vybraného záznamu, do kterého jsou přednačtena data vybraného záznamu.

### 4.2 Návrh a aplikační logika

Aplikační logika je implementovaná přímo ve třídách samotných formulářů, přistupuje tedy přímo k vlastnostem komponent zobrazených v prezenční vrstvě. Události, jako je například zmáčknutí tlačítka mají přímo implementováno obstarání konkrétní úlohy, která je k dané události přiřazena. Aplikace kromě samotných formulářů a jejich tříd také obsahuje několik dalších pomocných tříd.

#### 4.2.1 Pomocné třídy v aplikaci

*Constants.cs* – Tato třída obsahuje statické konstanty v aplikaci. V tomto případě je zde uložen název XML souboru, do kterého se ukládají data z aplikace. Důvodem vytvoření této třídy je to, aby se stejné záznamy používané ve více místech v aplikaci daly měnit na jednom místě.

*Entry* – Tato třída reprezentuje datovou strukturu jednotlivých záznamů pro vnitřní práci aplikace s nimi, v samotné aplikaci jsou záznamy udržovány v listu těchto tříd.

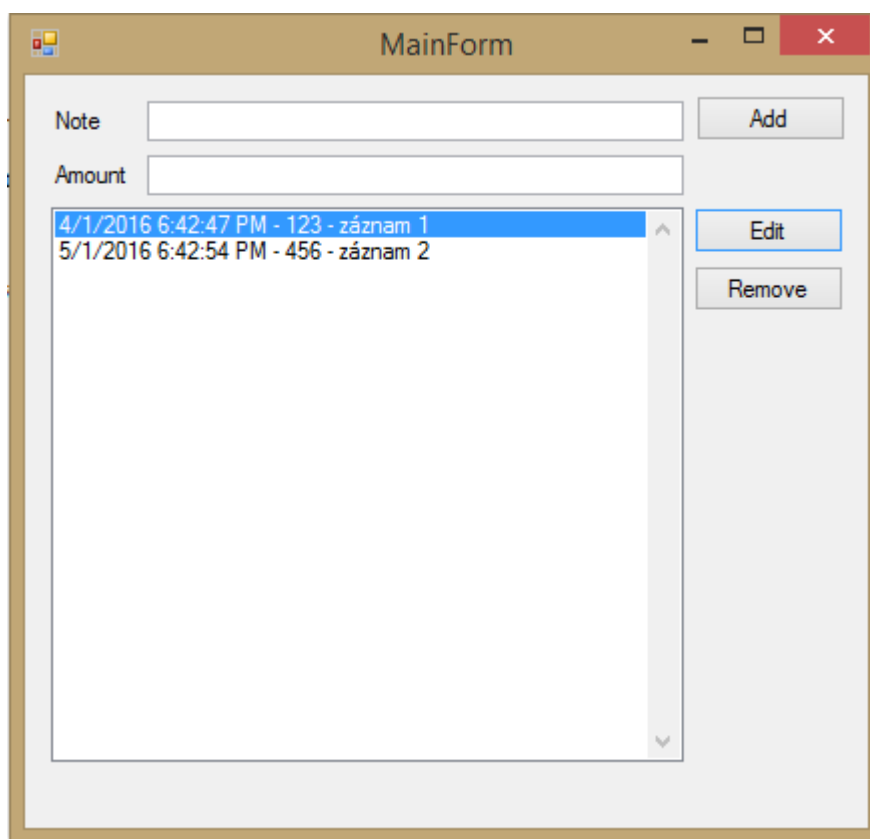
*ListEntry* – Tato třída reprezentuje datovou strukturu záznamů používanou v prezenční vrstvě aplikace, stejně jako u předchozí třídy jsou v rámci aplikace záznamy, které tvoří

instance této třídy udržovány v listu..

*XMLDataWorker* – Tato třída obsahuje funkce pro ukládání a načítání dat z XML souboru. O práci s XML daty se zde stará třída *XmlSerializer*, která je součástí .NET frameworku.

#### 4.2.2 Formuláře, ze kterých se skládá aplikace

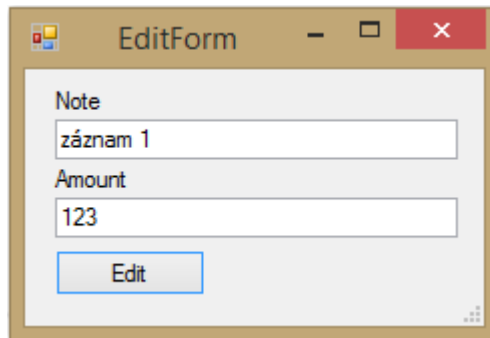
*MainForm* – Jedná se o hlavní formulář aplikace, který se otevře při jejím spuštění. Prezenční vrstva tohoto formuláře slouží k zobrazení vložených záznamů v aplikaci a k vkládání nových záznamů. V aplikační vrstvě tohoto formuláře je obsažena logika pro práci s prezenční vrstvou, jsou zde udržována data, se kterými aplikace pracuje, probíhá zde také práce se třídou *XMLDataWorker*. Při zmáčknutí tlačítka pro editaci záznamu je zde vyvoláno otevření nového formuláře *EditForm.cs*.



Obrázek 2 - MainForm

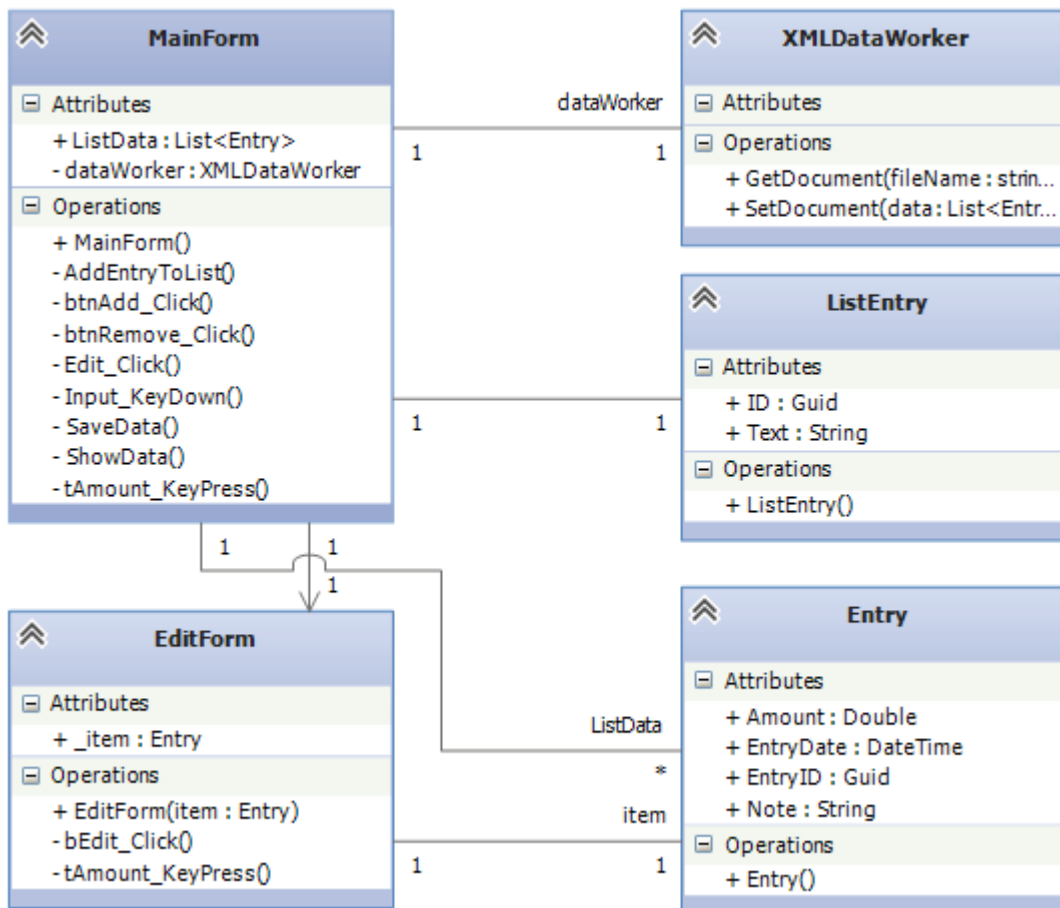
*EditForm* – Jedná se o formulář, který slouží pro úpravu jednotlivých záznamů uložených v aplikaci. Formulář obsahuje pole pro částku a popis. Při inicializaci formuláře jsou do těchto polí vepsána data pro upravovaný záznam. Datová pole tohoto formuláře jsou veřejná a při potvrzení úprav vybraného záznamu proběhne načtení hodnot v polích tohoto for-

muláře do MainFormu a následné uzavření tohoto formuláře.



Obrázek 3 - EditForm

### 4.2.3 Diagram tříd



Obrázek 4 - Diagram tříd

## 5 WINDOWS FORMS APLIKACE S VYUŽITÍM NÁVRHOVÉHO VZORU PASSIVE VIEW

Tato aplikace demonstruje využití návrhového vzoru Passive View při vývoji formulářových aplikací pomocí technologie Windows Forms. Návrh aplikace vychází z MVC návrhu pro Windows Forms od Alexe Volynského zveřejněného na webu Codeproject. [24]

Aplikace je vytvořena podle vzoru, který jeho tvůrce Alex Volynsky označuje jako MVC, avšak struktura aplikace lépe vystihuje návrhový vzor Passive View od Martina Fowlera.

### 5.1 Popis aplikace

Z uživatelského hlediska je tato aplikace stejná jako aplikace, které se týká předchozí bod této bakalářské práce, tedy standardní windows forms aplikace. Rozdíl mezi těmito aplikacemi je ve způsobu, kterým jsou vytvořeny z vývojářského hlediska.

### 5.2 Návrh a aplikační logika

Aplikace je navržena podle návrhového vzoru Passive View a skládá se celkem ze 4 projektů. Těmito projekty jsou Model, View, Controller a User.

#### 5.2.1 Model

Model je projekt obsahující třídy, které v rámci návrhového vzoru Passive View slouží jako modely a také třídy, ze kterých se tyto modely skládají. Projekt obsahuje 2 modely.

*MainViewModel* – Jedná se o třídu, která obsahuje datovou vrstvu pro tu část aplikace, která se stará o zobrazování záznamů. Třída se skládá z listů instancí tříd Entry a ListEntry, jejichž struktura je stejná jako v předchozí aplikaci.

*EditViewModel* – Jedná se o datovou vrstvu pro tu část aplikace, která se stará o úpravu jednotlivých záznamů. Tento model obsahuje pouze instanci třídy Entry.

#### 5.2.2 View

View je projekt, který obsahuje prezenční vrstvu aplikace. V této aplikaci obsahuje projekt View 2 view, což jsou standardní windows forms formuláře, pouze se s nimi v rámci celé aplikace pracuje jinak než v předchozí aplikaci. Tyto view v sobe udržují instanci příslušného controlleru a modelu.

*MainView* – Jedná se o hlavní formulář aplikace, ve které se zobrazují všechny záznamy a která slouží k vkládání nových záznamů. Prezenční vrstva tohoto formuláře je stejná jako prezenční vrstva u formuláře MainForm z předchozí aplikace. Aplikační vrstva formuláře ale v tomto případě neobsahuje žádnou aplikační logiku, slouží pouze k předávání hodnot mezi prezenční vrstvou a controllerem a k volání příslušných metod controlleru v případě vyvolání události v některé z komponent formuláře. Jedinou výjimkou je volání a otevření formuláře EditView, kde je tato logika implementována v MainView, její inicializace ale stále probíhá přes controller. MainView implementuje rozhraní IMainView a udržuje si aktivní instanci MainViewControlleru.

*EditView* – Tento formulář slouží v rámci aplikace jako uživatelské rozhraní vrstva pro úpravu záznamů. Prezenční vrstva tohoto formuláře je stejná jako prezenční vrstva formuláře EditForm z předchozí aplikace. Aplikační vrstva zde slouží stejně jako u předchozího view pouze k předávání hodnot do controlleru a o jeho volání v případě vyvolání události v rámci view. EditView implementuje rozhraní IEditView a udržuje si instanci EditViewControlleru.

### 5.2.3 Controller

Controller je projekt, který obsahuje řídicí logiku aplikace. Jsou zde obsaženy třídy pro práci s XML souborem, controllery MainViewController a EditViewController a také rozhraní IMainView a IEditView. Controllery zde implementují logiku, která byla v předchozím projektu obsažena v aplikační logice samotných formulářů.

*MainViewController* – Jedná se o třídu, která slouží k vyhodnocování uživatelské interakce s MainView. MainViewController v sobě udržuje obecnou instanci view, předepsanou rozhraním IMainView, controller tedy neudržuje odkaz na konkrétní view. MainViewController také implementuje logiku práce s XML souborem, ve kterém jsou uloženy uživatelem vytvořené záznamy.

*EditViewController* – Jedná se o třídu, která slouží k vyhodnocování uživatelské interakce s EditView. EditViewController v sobě udržuje obecnou instanci view, předepsanou rozhraním IEditView.

*IMainView, IEditView* – IMainView a IEditView jsou rozhraní, se kterým pracují konkrétní controllery a které musí implementovat view, se kterým daný controller pracuje. Tyto

rozhraní předepisují konkrétní metody a vlastnosti, které musí view mít v případě, že s ním má daný controller pracovat. V případě potřeby výměny view tedy stačí pouze vytvořit nové view, které bude implementovat tyto rozhraní a controller s ním bude moci pracovat.

#### 5.2.4 Úprava záznamu

Úprava záznamu, tedy vyvolání dalšího formulářového okna je v této aplikaci inicializováno pomocí události v MainView, která je vyvolána při zmáčknutí tlačítka pro „Edit“, které slouží k vyvolání formuláře úpravu záznamu. Tato událost zavolá metodu controlleru, který následně předá do view konkrétní záznam, který má být upraven. MainView následně vytvoří instanci EditView a EditViewControlleru, kterému předá záznam pro úpravu a instanci EditView. Po potvrzení úpravy záznamu je do controlleru předán upravený záznam, který následně controller vymění s původním záznamem v modelu. Toto je jeden z mála případů, kdy je aplikační logika implementována přímo ve view namísto v controlleru a je to z toho důvodu, že controller nemá přístup do projektu View, takže nemůže vytvořit instance view.

```
private void Edit_Click(object sender, EventArgs e)
{
    Guid ID = (List.SelectedItem as ListEntry).ID;
    _controller.EditEntry(ID);
}
```

*Kód 3. Událost v MainView pro zmáčknutí tlačítka úpravy záznamu*

```
public void EditEntry(Guid entryID)
{
    Entry toEdit;
    toEdit = _model.Entries.Where(p => p.EntryID == entryID).FirstOrDefault();

    if (toEdit != null)
        _view.EditEntry(toEdit);
}
```

*Kód 4. Metoda controlleru zpracovávající žádost o úpravu záznamu*

```
public void EditEntry(Entry entry)
{
    EditView editView = new EditView();

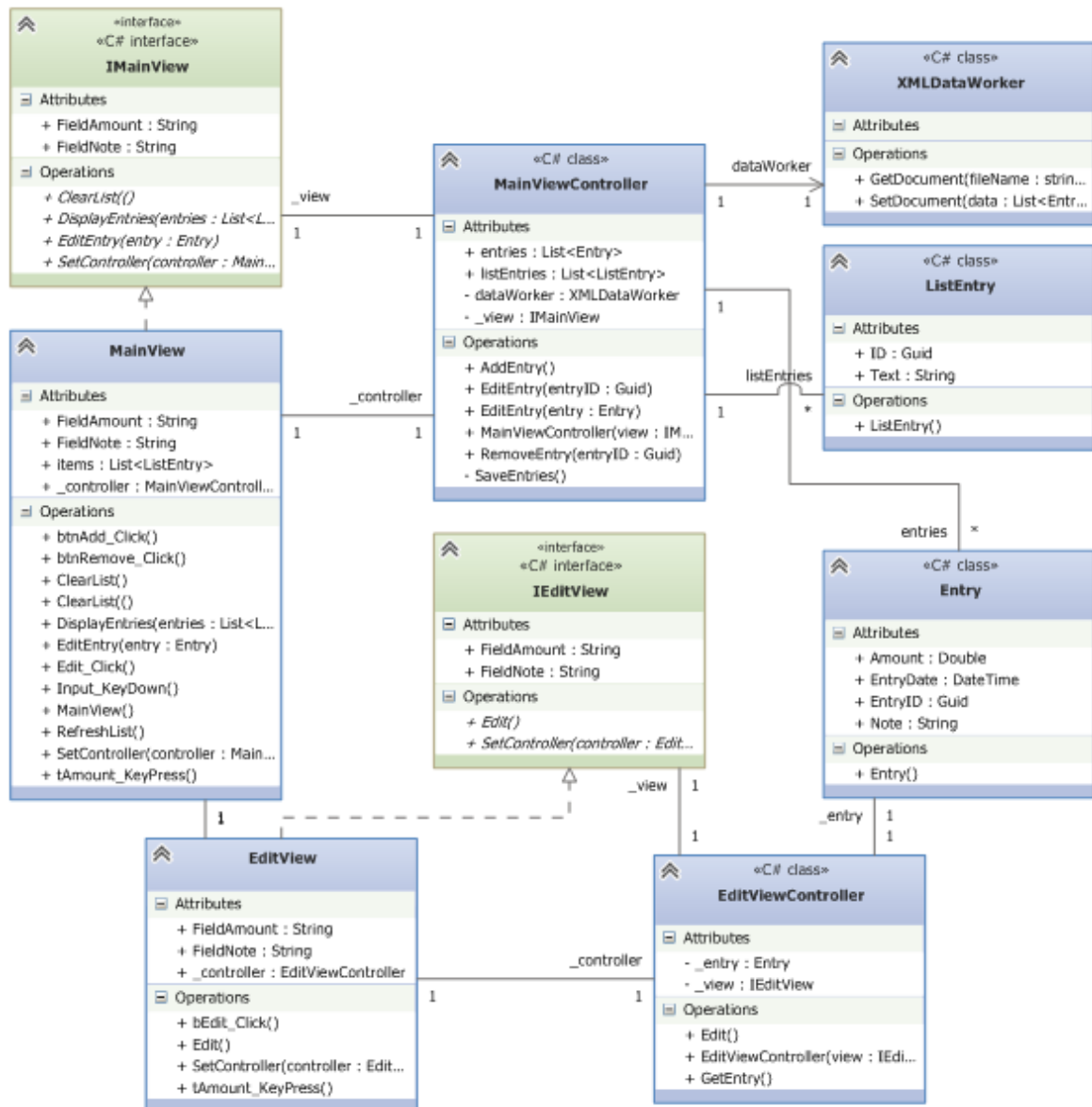
    Controller.EditViewController controller;
    controller = new Controller.EditViewController(editView, entry);
    editView.SetController(controller);

    using (editView)
    {
        var result = editView.ShowDialog();
    }

    _controller.EditEntry(controller.GetEntry());
}
```

*Kód 5. Metoda v MainView implementující otevření formuláře pro úpravu*

### 5.2.5 Diagram tříd



Obrázek 5 - Diagram tříd

## 6 ASP .NET MVC APLIKACE

Tato aplikace demonstruje využití návrhového vzoru MVC při použití ve webových aplikacích, konkrétně použitím technologie ASP .NET obsažené v .NET frameworku.

### 6.1 Popis aplikace

Aplikace je stejnou evidencí příjmů a útrat, jako předchozí 2 aplikace, pouze je implementovaná ve webovém rozhraní. Z uživatelského hlediska je zde rozdíl pouze v tom, že přidání nového záznamu zde neprobíhá v hlavní části aplikace, ale je odděleno zvlášť, stejně jako editace záznamu.

### 6.2 Návrh a aplikační logika

Aplikace se skládá z jednoho controlleru, dvou modelů a několika view. Dále jsou zde obsaženy 2 pomocné třídy a to třída obsahující konstantní data v aplikaci a třída pro práci s XML souborem.

#### 6.2.1 Modely

Aplikace obsahuje 2 modely, prvním modelem je třída `EntriesModel.cs`, druhým modelem je třída `Entry.cs`. Modely jsou obsaženy v adresáři `Models`.

*Entry* – `Entry` je třída, která slouží jako model pro akce vkládání a upravování záznamů, třída se skládá z proměnné pro datum a čas, sumu, poznámku a unikátního identifikátoru `Guid`. Oproti třídě `Entry` použité v předchozích aplikacích je tu navíc booleovná proměnná „`IsNewEntry`“, která slouží pro identifikaci toho, zda se jedná o úpravu záznamu nebo vložení nového.

*EntriesModel* – `EntriesModel` je třída, která obsahuje list instancí třídy `Entry.cs`. `EntriesModel` slouží jako model pro view, ve kterém je zobrazen seznam uživatelem vložených záznamů.

#### 6.2.2 View

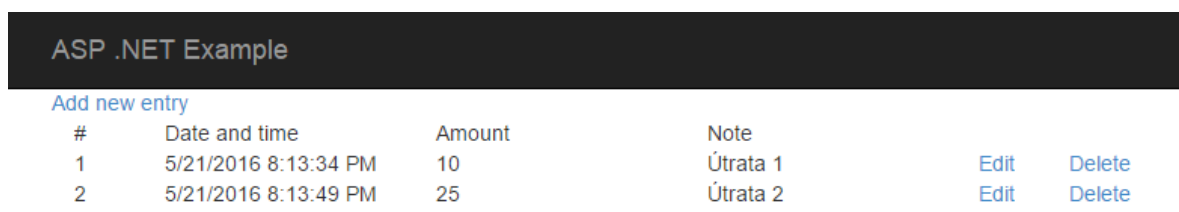
View slouží jako prezenční vrstva pro uživatele. V ASP .NET jsou view HTML stránky, které zobrazují data z modelu. View jsou obsaženy v adresáři `Views` a následně v jeho konkrétních podadresářích, které jsou pojmenovány podle controlleru, který s danou skupinou view pracuje. GUI view je vytvářeno pomocí HTML tagů, kaskádových stylů `css` a případně pomocí `javascriptu`. Všechny view nemusí obsahovat model, je možné je mít

v aplikaci pouze jako statické stránky. View obsahují obecnou definici modelu se kterým pracují, konkrétní instanci modelu s konkrétními daty, které bude view obsahovat jsou předávány z controlleru.

*\_Layout* - *\_Layout* view je view, ve kterém je nadefinovaná hlavička a patička celé aplikace, v tomto view jsou vykreslovány view, které jsou předávány z controlleru pro zobrazení uživateli.

*\_ViewStart* – V tomto view je nadefinováno použití view *\_Layout* jako společného view pro celou aplikaci.

*Index* – Toto view slouží jako hlavní view aplikace, je v něm zobrazen seznam uživatelem vložených záznamů a jsou zde odkazy pro úpravu, smazání nebo přidání záznamu. Modelem pro toto view je *EntriesModel*.



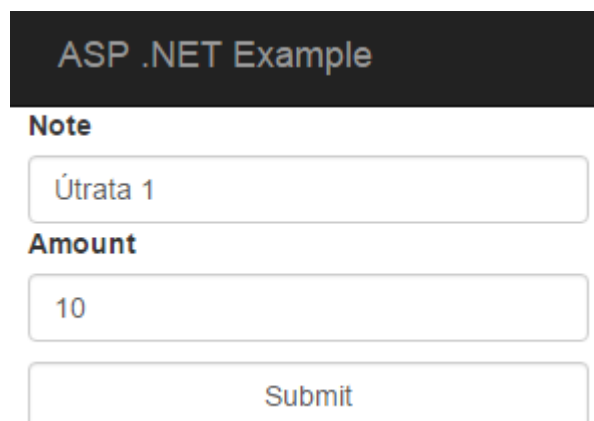
ASP .NET Example

[Add new entry](#)

#	Date and time	Amount	Note		
1	5/21/2016 8:13:34 PM	10	Útrata 1	<a href="#">Edit</a>	<a href="#">Delete</a>
2	5/21/2016 8:13:49 PM	25	Útrata 2	<a href="#">Edit</a>	<a href="#">Delete</a>

Obrázek 6 – Hlavní stránka ASP .NET aplikace

*ProcessEntry* – Toto view slouží pro vkládání nových záznamů nebo pro úpravu existujících záznamů. Modelem pro toto view je *Entry*. V případě potřeby mít rozdílné view pro vkládání a pro úpravu záznamu je možné na základě parametru „*IsNewEntry*“ vytvořit pomocí Razor syntaxe podmínky, které budou rozdíly implementovat.



ASP .NET Example

**Note**

**Amount**

Obrázek 7 - Úprava a vkládání záznamu v ASP .NET aplikaci

### 6.2.3 Controller

Aplikace má pouze jeden controller a to „HomeController“. Controller je umístěn v adresáři Controllers.

V ASP .NET MVC je controller třídou, která se stará o zpracování uživatelských interakcí s view. Tato činnost je implementovaná pomocí ActionResultů, což jsou funkce, které zpracovávají uživatelské akce a na jejich základě navrací určitá view s určitými modely.

*HomeController* – V této aplikaci obsahuje HomeController 6 ActionResultů a 3 pomocné funkce pro vkládání, mazání a načtení dat z XML souboru.

První ActionResult slouží k zobrazení hlavní stránky aplikace, tedy seznamu záznamů, které uživatel do aplikace vložil. Pokud není v návratovém view určen název view, je automaticky navrženo view, jehož název odpovídá názvu ActionResultu.

```
public ActionResult Index()
{
    EntriesModel model = GetEntriesModel();
    return View(model);
}
```

*Kód 6. ActionResult pro zobrazení hlavní stránky aplikace*

Dalšími z ActionResultů jsou AddEntry. První z těchto ActionResultů je volán při žádosti uživatele aplikace o vložení nového záznamu a navrací view „ProcessEntry“. Druhý ActionResult slouží jako reakce aplikace na potvrzení vložení záznamu s daty z uživatelské strany. Parametr „HttpPost“ zde slouží k určení toho, že daný ActionResult je volán pouze jako odpověď na http metodu Post. ActionResult provede zavolání funkce pro uložení nového záznamu se vstupními daty, které jsou v tomto případě ve formě instance třídy „Entry“ a následně navrací ActionResult „Index“, tedy přesměruje uživatele zpět na hlavní stránku aplikace.

```
public ActionResult AddEntry()
{
    Entry model = new Entry();
    model.IsNewEntry = true;
    return View("ProcessEntry", model);
}
```

```
[HttpPost]
public ActionResult AddEntry(Entry model)
{
    model.EntryDate = DateTime.Now;
    model.EntryID = Guid.NewGuid();
    ProcessEntry(model, true);
    return RedirectToAction("Index");
}
```

*Kód 7. ActionResulty pro přidání nového záznamu do aplikace*

Další 2 ActionResulty slouží k úpravě záznamů v aplikaci. Vstupem do prvního ActionResultu je identifikátor záznamu, který se bude upravovat, následně jsou načteny všechny záznamy a je vybrán ten, který odpovídá danému identifikátoru, tento záznam je následně předán do view „ProcessEntry“ jako jeho model. Druhý ActionResult slouží jako reakce na potvrzení úpravy záznamu. Podobně jako v předchozím případě pro vkládání záznamu je zde volána funkce pro zpracování daného záznamu.

```
public ActionResult Edit(Guid ID)
{
    EntriesModel model = GetEntriesModel();
    var entry = model.Entries.Where(p => p.EntryID == ID).FirstOrDefault();
    entry.IsNewEntry = false;

    return View("ProcessEntry", entry);
}

[HttpPost]
public ActionResult Edit(Entry model)
{
    ProcessEntry(model, false);
    return RedirectToAction("Index");
}
```

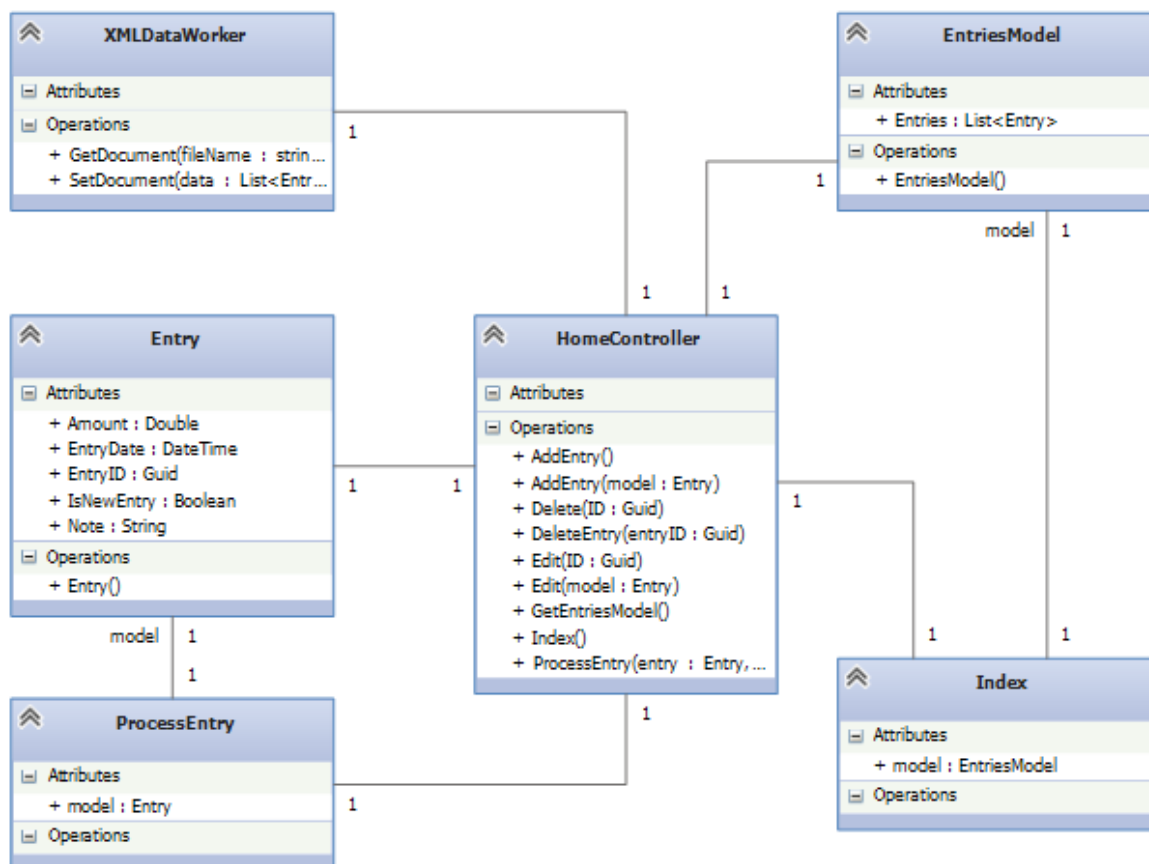
*Kód 8. ActionResulty pro úpravu existujícího záznamu do aplikace*

Posledním ActionResultem controlleru této aplikace je „Delete“, který slouží pro smazání existujícího záznamu. Vstupním parametrem pro smazání záznamu je stejně jako u úpravy záznamu identifikátor pro konkrétní záznam.

```
public ActionResult Delete(Guid ID)
{
    Entry model = new Entry();
    DeleteEntry(ID);
    return RedirectToAction("Index");
}
```

*Kód 9. ActionResult pro smazání existujícího záznamu*

### 6.2.4 Diagram tříd



Obrázek 8 - Diagram tříd

## 7 MVVM APLIKACE

Tato aplikace demonstruje využití návrhového vzoru MVVM. Aplikace je vytvořena pomocí frameworku WPF, pro který byl návrhový vzor MVVM vytvořen.

### 7.1 Popis Aplikace

Aplikace je stejnou evidencí příjmů a útrat, jako předchozí 3 aplikace, na rozdíl od předchozích aplikací je ale vytvořena ve frameworku WPF a je vytvořena jako desktopová aplikace. Aplikace je vzhledově odlišná od předchozích windows forms aplikací, protože Framework WPF má jiné komponenty a jiný způsob tvorby GUI.

Aplikace se skládá ze tří hlavních částí a to z view, modelu a viewmodelů.

#### 7.1.1 Model

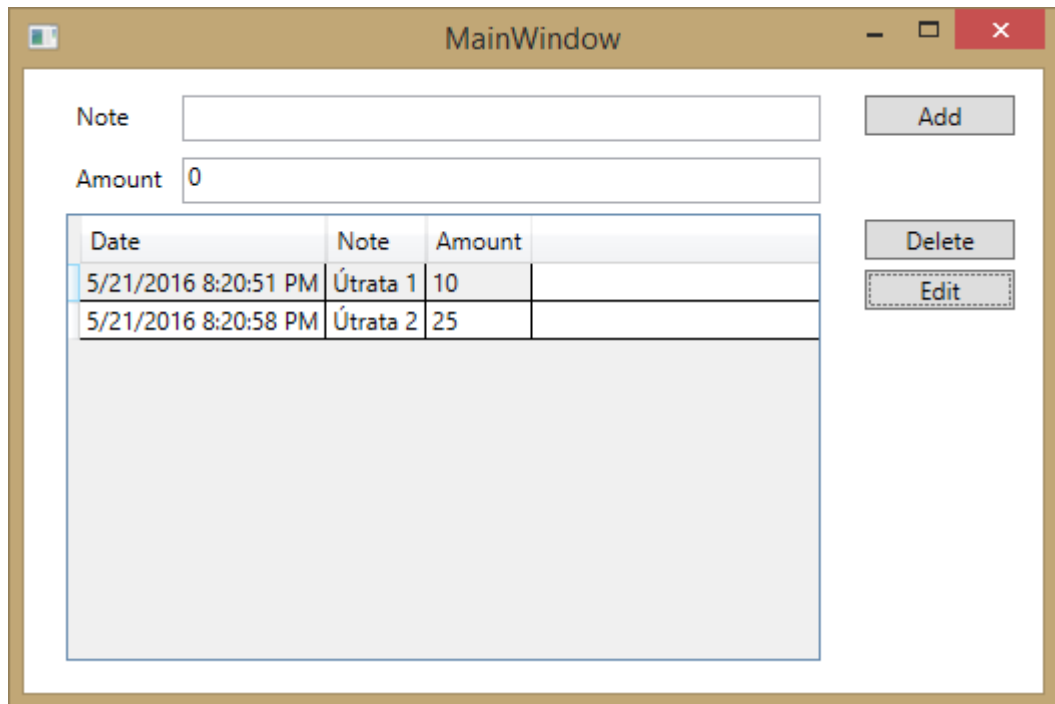
*Entry* – *Entry* je třída, která slouží jako model pro akce vkládání a upravování záznamů, třída se skládá z pole pro datum a čas, sumu, poznámku a unikátního identifikátoru *Guid*.

#### 7.1.2 View

*MainView* – *MainView* je hlavním formulářovým oknem aplikace, obsahuje komponenty pro vkládání záznamů a pro zobrazení již vložených záznamů. *MainView* neobsahuje žádnou aplikační logiku, je mu pouze přiřazen *viewmodel* jako *DataContext* a následně jsou na konkrétní proměnné ve *viewmodelu* nabíndované komponenty z *view*.

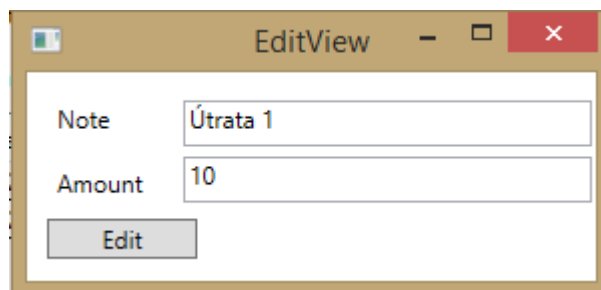
```
<TextBox Text="{Binding NewEntry.Note}" x:Name="tNote" HorizontalAlignment="Left"
    Height="23" Margin="79,13,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
    Width="319"/>
```

*Kód 8. TextBox nabíndovaný na proměnnou NewEntry.Note z viewmodelu*



Obrázek 9 - Hlavní formulář MVVM aplikace

*EditView* – *EditView* slouží podobně jako v předchozích aplikacích pro úpravu existujících záznamů. Obsahuje pole pro poznámku a sumu.



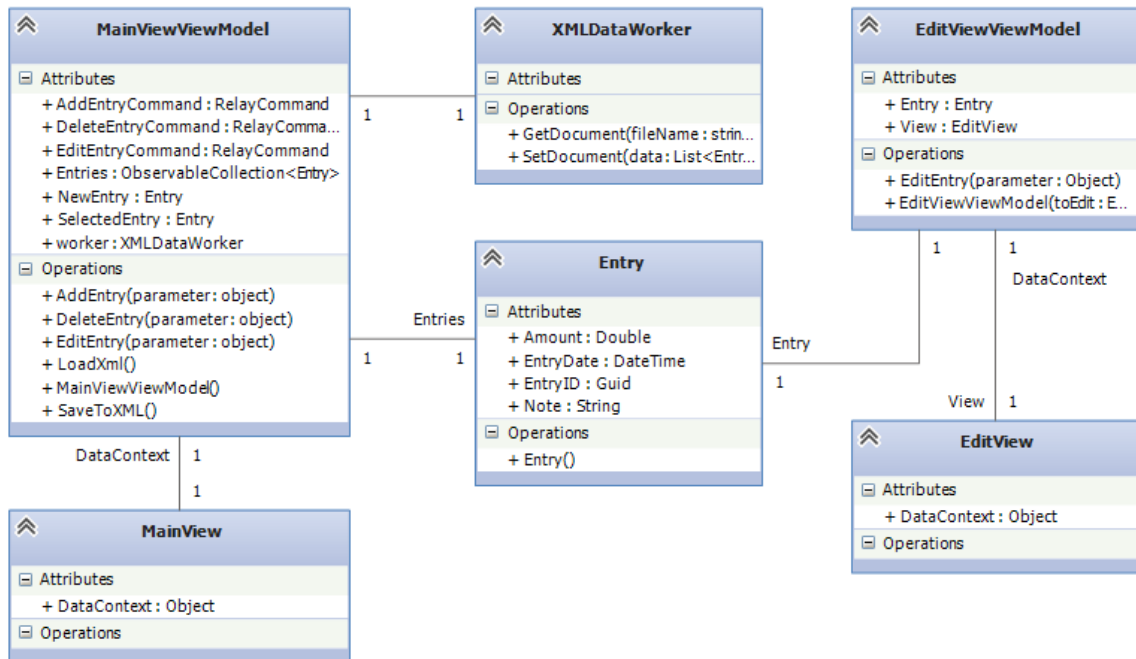
Obrázek 10 - Formulář pro úpravu záznamu v MVVM aplikaci

### 7.1.3 ViewModel

*MainViewViewModel* – *MainViewViewModel* je třída, která je propojená se *MainView* a stará se o aplikační logiku související s interakcí uživatele s tímto view. Jsou zde vytvořeny instance modelu, na které jsou nabitované komponenty v *MainView*.

*EditViewViewModel* – *EditViewViewModel* je třída, která je propojená se *EditView* a stará se o aplikační logiku související s interakcí uživatele s tímto view. Pomocí konstruktoru je do této třídy předána z *MainViewViewModel* instance *EditView*, aby ho bylo možné zavřít a záznam, který má být upraven.

### 7.1.4 Diagram tříd



Obrázek 11 - Diagram tříd

## 8 ÚKOLY PRO POTŘEBY VÝUKY

- 1) Rozšiřte všechny demonstrační aplikace o novou hodnotu vstupu „User“, která se bude vkládat při vytváření záznamu v aplikaci i při jeho úpravě. Tato hodnota bude indikovat, kdo daný záznam vložil. V každé aplikaci proveďte toto rozšíření dle příslušného návrhového vzoru.
- 2) Aplikaci ASP MVC rozšiřte o součet celkové částky ze všech záznamů uložených v aplikaci. Toto rozšíření vytvořte bez úprav controlleru.
- 3) Aplikaci ASP MVC přepracujte tak, že práce s XML souborem (načítání a ukládání dat), tedy třídou XMLDataWorker bude přesunuta z controlleru do modelu. V controlleru budou pouze volání příslušných funkcí modelu.
- 4) Pomocí technologie ASP .NET MVC vytvořte jednoduchou kalkulačku pro sčítání a odčítání dvou hodnot. Veškerá výpočtová logika této kalkulačky bude obsažena v modelu. Uživatelské rozhraní bude obsahovat pole pro vstupní hodnoty, tlačítko pro provedení výpočtu a výběr matematické operace. Prvek pro výběr operace si může student zvolit sám.

## ZÁVĚR

V této práci byl popsán framework .NET, historie jeho verzí a programovací jazyk C#, který je jeho součástí. Detailněji zde byl popsán dotazovací jazyk LINQ a lambda výrazy, common language runtime a jeho výhody a také projekt Mono, který je alternativou k .NET frameworku.

Práce obsahuje základní popisy návrhových vzorů MVC, MVP a MVVM, passive view a supervising controller a také popis frameworků pro vývoj desktopových aplikací Windows Forms a WPF.

V rámci praktické části této práce byla vytvořena windows forms aplikace demonstrující vývoj bez použití návrhových vzorů, windows forms aplikace demonstrující použití návrhového vzoru Passive View, ASP .NET MVC aplikace demonstrující využití návrhového vzoru MVC pro vývoj webových aplikací a WPF aplikace demonstrující vývoj s použitím návrhového vzoru MVVM.

Na závěr práce bylo navrženo několik zadání úkolů pro potřeby výuky týkajících se problematiky popsané v této práci a demonstračních aplikací v ní vytvořených.

**SEZNAM POUŽITÉ LITERATURY**

- [1] Overview of the .NET Framework. In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [https://msdn.microsoft.com/en-us/library/zw4w595w\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx)
- [2] Programming Languages. In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [https://msdn.microsoft.com/en-us/library/aa292164\(v=vs.71\).aspx](https://msdn.microsoft.com/en-us/library/aa292164(v=vs.71).aspx)
- [3] C#. In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: <https://msdn.microsoft.com/en-us/library/kx37x362.aspx>
- [4] Lambda Expressions (C# Programming Guide). In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: <https://msdn.microsoft.com/en-us/library/bb397687.aspx>
- [5] NAGEL, Christian., Jay. GLYNN a Morgan. SKINNER. *Professional C# 5.0 and .NET 4.5.1*. Indianapolis, IN: John Wiley and Sons, 2014. ISBN 9781118832943.
- [6] LANDWERTH, Immo. .NET Core is Open Source. In: *.NET Blog* [online]. 2014 [cit. 2016-05-25]. Dostupné z: <https://blogs.msdn.microsoft.com/dotnet/2014/11/12/net-core-is-open-source/>
- [7] .NET Framework Versions and Dependencies. In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [https://msdn.microsoft.com/en-us/library/bb822049\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/bb822049(v=vs.110).aspx)
- [8] Common Language Runtime (CLR). In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [https://msdn.microsoft.com/cs-cz/library/8bs2ecf4\(v=vs.110\).aspx](https://msdn.microsoft.com/cs-cz/library/8bs2ecf4(v=vs.110).aspx)
- [9] Garbage Collection. In: *Microsoft Developer Network* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [https://msdn.microsoft.com/en-us/library/0xy59wtx\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/0xy59wtx(v=vs.110).aspx)
- [10] *Mono* [online]. 2016 [cit. 2016-05-21]. Dostupné z: <http://www.mono-project.com/>
- [11] ČÁPKA, David. 1. díl - Úvod do Windows Forms aplikací. In: *ITnetwork.cz* [online]. Praha, 2013 [cit. 2016-05-25]. Dostupné z:

<http://www.itnetwork.cz/csharp/windows-forms/c-sharp-tutorial-windows-forms-okenni-aplikace-uvod/>

- [12] PETZOLD, Charles. *Mistrovství ve Windows Presentation Foundation: [aplikace = kód + markup]*. Brno: Computer Press, 2008. Mistrovství. ISBN 978-80-251-2141-2.
- [13] Model-view-controller. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-05-25]. Dostupné z: <https://cs.wikipedia.org/wiki/Model-view-controller>
- [14] GALLOWAY, Jon. *Professional ASP. NET MVC 5*. Indianapolis, Indiana: John Wiley & Sons, Inc., 2014. ISBN 9781118794760.
- [15] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003. ISBN 0321127420.
- [16] ASP.NET MVC Tutorial. In: *W3schools.com* [online]. 2016 [cit. 2016-05-25]. Dostupné z: [http://www.w3schools.com/aspnet/mvc\\_intro.asp](http://www.w3schools.com/aspnet/mvc_intro.asp)
- [17] Interactive Application Architecture Patterns. *Aspiring Craftsman* [online]. Nashville: derekgreer, 2006 [cit. 2016-05-21]. Dostupné z: <http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/>
- [18] GAROFALO, Raffaele. *Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern*. [Online-Ausg.]. Sebastopol, Calif: O'Reilly Media, 2011. ISBN 9780735650923.
- [19] Model–view–viewmodel. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2016-05-20]. Dostupné z: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>
- [20] *KnockoutJS* [online]. [knockoutjs.com](http://knockoutjs.com/), 2016 [cit. 2016-05-25]. Dostupné z: <http://knockoutjs.com/>
- [21] Retirement note for Model View Presenter Pattern. *martinfowler.com* [online]. Chicago: Martin Fowler, 2006 [cit. 2016-05-21]. Dostupné z: <http://martinfowler.com/eaDev/ModelViewPresenter.html>
- [22] Passive View. *martinfowler.com* [online]. Chicago: Martin Fowler, 2006 [cit. 2016-05-21]. Dostupné z: <http://martinfowler.com/eaDev/PassiveScreen.html>

- [23] Supervising Controller. *martinfowler.com* [online]. Chicago: Martin Fowler, 2006 [cit. 2016-05-21]. Dostupné z: <http://martinfowler.com/eaDev/SupervisingPresenter.html>
- [24] The Model-View-Controller(MVC) Pattern with C#/WinForms. In: *CODEPROJECT* [online]. online: Volynsky, 2012 [cit. 2016-05-01]. Dostupné z: <http://www.codeproject.com/Articles/383153/The-Model-View-Controller-MVC-Pattern-with-Csharp>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

MVC	Model View Controller
MVP	Model View Presenter
MVVM	Model View ViewModel
WPF	Windows Presentation Foundation
CLR	Common Language Runtime
LINQ	Language Integrated Query

**SEZNAM OBRÁZKŮ**

<i>Obrázek 1 – ASP .NET MVC diagram [16]</i> .....	20
<i>Obrázek 2 - MainForm</i> .....	26
<i>Obrázek 3 - EditForm</i> .....	27
<i>Obrázek 4 - Diagram tříd</i> .....	27
<i>Obrázek 5 - Diagram tříd</i> .....	32
<i>Obrázek 6 – Hlavní stránka ASP .NET aplikace</i> .....	34
<i>Obrázek 7 - Úprava a vkládání záznamu v ASP .NET aplikaci</i> .....	34
<i>Obrázek 8 - Diagram tříd</i> .....	37
<i>Obrázek 9 - Hlavní formulář MVVM aplikace</i> .....	39
<i>Obrázek 10 - Formulář pro úpravu záznamu v MVVM aplikaci</i> .....	39
<i>Obrázek 11 - Diagram tříd</i> .....	40

## SEZNAM TABULEK

Tabulka 1 - Verze .NET frameworku [7] .....	15
---	----

**SEZNAM PŘÍLOH**

- ASP MVC.zip - Demonstrační aplikace využití návrhového vzoru MVC
- MVC.zip - Demonstrační aplikace využití návrhového vzoru Passive View
- MVVM.zip - Demonstrační aplikace využití návrhového vzoru MVVM
- WinForm.zip - Demonstrační aplikace vývoje Windows Forms aplikací bez použití návrhových vzorů