Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Tvorba aplikace pomocí C#, WPF - Databáze parkourových spotů v Praze

Lukáš Kolář

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Lukáš Kolář

Informatika

Název práce

Tvorba aplikace pomocí C#, WPF – Databáze parkourových spotů v Praze

Název anglicky

Application creation in C#, WPF – Database of parkour spots in Prague

---

Cíle práce

Práce je zaměřena na problematiku vývoje aplikací v jazyce C# s využitím technologie WPF. Cílem práce je návrh a implementace databázové aplikace sloužící k evidenci míst pro trénink parkouru v Praze.

Metodika

Metodika řešení bakalářské práce je založena na analyticko-syntetickém přístupu. Bude provedena analýza odborných informačních zdrojů, jejíž souhrn bude uveden v teoretické části práce. Na základě syntézy těchto poznatků a uživatelských požadavků bude proveden návrh a posléze implementace aplikace sloužící k evidenci míst pro trénink parkouru v Praze. Aplikace bude využívat lokální a externí databázi.

Při tvorbě aplikace bude využit programovací jazyk C# na platformě .NET a technologie WPF pro tvorbu uživatelského rozhraní.

Doporučený rozsah práce

35-40 stran

Klíčová slova

C#, WPF, XAML, databáze, programování, tvorba aplikace

Doporučené zdroje informací

AGARWAL, V.V – HUDDLESTON, J – RAGHURAM, R. Beginning C# 2008 Databases From Novice to Professional. New York, 2008. ISBN13: 978-1-59059-900-6

HANÁK, Ján. Praktické objektové programování v jazyce C# 4.0. Brno: Artax, 2009. Microsoft (Artax). ISBN 978-80-87017-07-4.

MACDONALD, M. Pro WPF 4.5 in C# Windows Presentation Foundation in .NET 4.5. New York, 2012. ISBN: 978-1-4302-4365-6

MAMTA DALAL, Ashish Ghoda. XAML developer reference. Sebastopol, Calif: O'reilly Media, 2011. ISBN 9780735658967.

NAKOV, S – KOLEV, V. Fundamentals of Computer Programming with C#. Sofia, 2013. ISBN: 978-954-400-773-7

PIRKL, J. Řešené příklady v C sharp – aneb C# skutečně prakticky. 2005. ISBN: 80-7232-265-6

SEMPF, B. – DAVIS, S.R. – SPHAR, C. C# 2010 All-in-One For Dummies. Hoboken, 2010. ISBN: 978-0-470-56348-9

STELLMAN, Andrew. a Jennifer GREENE. Head first C#. Sebastopol, CA: O'Reilly, c2007. ISBN 0596514824.

YOSIFOVICH, P. Windows Presentation Foundation 4.5 Cookbook 2012. Birmingham, UK, 2012. ISBN: 978-1849686228

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 19. 11. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Tvorba aplikace pomocí C#, WPF - Databáze parkourových spotů v Praze" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 19.11.2016 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení, za možnost konzultací a poskytování rad během tvorby této práce.

Tvorba aplikace pomocí C#, WPF - Databáze parkourových spotů v Praze

Souhrn

Tato bakalářská práce se zabývá návrhem a implementací databázové aplikace parkourových tréninkových míst v Praze ve vývojovém prostředí Microsoft Visual Studio 2015. Cílem práce je vytvoření funkční aplikace, která poskytuje přehled a informace o tréninkových místech v Praze dostupné pro všechny zájemce, zejména samotné parkouristy. Aplikace by měla fungovat jak v režimu s funkčním internetovým připojením, tak i v režimu offline, tedy bez připojení k internetu.

První část práce se zabývá principy objektově orientovaného programování v jazyce C#, technologií WPF s využitím značkovacího jazyka XAML pro tvorbu grafického uživatelského rozhraní, vývojovým prostředím Visual Studio a databázovými systémy MySQL a SQL Server Compact Edition pro možnost ukládání dat. Druhá část práce je zaměřena na samotný vývoj databázové aplikace. Vývoj se skládá z návrhu programové a databázové části aplikace, implementace těchto jednotlivých částí, které jsou navzájem propojené a publikace aplikace.

Závěr práce poskytuje shrnutí výsledků návrhu a implementace společně se zhodnocením přínosnosti a využitelnosti vytvořené databázové aplikace.

Klíčová slova: C#, WPF, XAML, .NET Framework, databáze, programování, tvorba aplikace

Application creation in C#, WPF - Database of parkour spots in Prague

Summary

This bachelor thesis focuses on the design and the implementation of Database of parkour spots in Prague application in the Visual Studio 2015 development environment. The aim of this thesis is to create functional application that provides an overview and information on training locations in Prague, available to all interested people, especially the very parkourists. Application should operate in the mode with a working internet connection, as well as offline, without an internet connection.

The first part deals with the principles of object oriented programming in C#, WPF technology using XAML markup language for creating graphical interface, Visual Studio development environment, MySQL and SQL Server Compact Edition database systems for data storage. The second part is focused on the development of database application. The development consists of the design of programming and database part of the application, implementation of these two individual parts, which are interconnected and publication of the application.

The thesis conclusion provides a summary of the results of design and implementation together with evaluation of usefulness and usability of the created database application.

Keywords: C#, WPF, XAML, .NET Framework, database, programming, application creation

Obsah

1 Úvod 12

2 Cíl práce a metodika 13

2.1 Cíl práce 13

2.2 Metodika 13

3 Teoretická východiska 14

3.1 .NET Framework 14

3.1.1 Architektura .NET Frameworku 14

3.1.2 Výhody .NET Frameworku 15

3.1.3 Verze .NET Frameworku 15

3.2 Programovací jazyk C# 16

3.2.1 Základy jazyka C# 16

3.2.2 Objektově orientované programování v C# 17

3.3 Technologie WPF 24

3.3.1 Grafika 24

3.3.2 Porovnání WinForms a WPF 25

3.4 Značkovací jazyk XAML 26

3.4.1 Struktura a syntaxe jazyka XAML 26

3.4.2 Data Binding 28

3.4.3 Události 31

3.4.4 Styly 32

3.4.5 Templates (Šablony) 34

3.4.6 Triggers (Spouště) 38

3.5 Microsoft Visual Studio 40

3.5.1 Editor kódu 40

3.5.2 Designer 41

3.5.3 Debugger 41

3.5.4 Další nástroje 42

3.6 Databáze 43

3.6.1 Lokální databáze – SQL Server Compact Edition (CE) 43

3.6.2 Externí databáze – MySQL 44

4 Vlastní práce 45

4.1 Návrh aplikace 45

4.1.1 Požadavky, funkcionalita aplikace a diagram případů užití 45

4.1.2 Použité třídy 48

4.1.3 Použité ovládací prvky 49

4.1.4 Databázové systémy, tabulky 50
4.2 Implementace databázových systémů 51
4.2.1 Implementace externí databáze MySQL 51
4.2.2 Implementace lokální databáze SQL Server Compact Edition 56
4.3 Implementace aplikace 61
4.3.1 Uživatelské rozhraní 61
4.3.2 Funkcionalita tříd a událostí 67
4.4 Publikace aplikace 73
4.4.1 Zpřístupnění projektu 73
4.4.2 Distribuce aplikace 73

5 Výsledky a diskuse 74
5.1 Výsledky 74
5.1.1 Kompatibilita 74
5.1.2 Funkcionalita 74
5.2 Diskuze 75
5.3 Závěr 77

6 Seznam použitých zdrojů 78

7 Přílohy 80

Seznam obrázků

Obrázek 1: Deklarace proměnné a přiřazení hodnoty 17
Obrázek 2: Řídící struktury 17
Obrázek 3: Příklad třídy v jazyce C# 18
Obrázek 4: Statická třída a přístup k prvkům třídy 19
Obrázek 5: Dědění a polymorfismus 21
Obrázek 6: Rozhraní a jeho implementace 22
Obrázek 7: Abstraktní třáda 22
Obrázek 8: Definování a vytvoření delegáta 23
Obrázek 9: Párové a nepárové elementy v XAML 26
Obrázek 10: Defionování jmenného prostoru v XAML 27
Obrázek 11: Ukázka struktury a syntaxe XAML kódu 28
Obrázek 12: Data Binding mezi dvěma elementy 29
Obrázek 13: Data Binding – přenos dat z kódu C# do XAML elementu 30

Obrázek 14: Data Binding s využitím vlastnosti RelativeSource 31
Obrázek 15: Využití DataContextu – vypsání stejného zdroje do více elementů 31
Obrázek 16: Defionování stylu u elementu StackPanel pro všechny elementy Button ..32
Obrázek 17: Definování stylu v souboru App.xaml a přiřazení stylu do elementu 33
Obrázek 18: Dědění stylu 33
Obrázek 19: Definováná šablony - Control Template.....35
Obrázek 20: Definování šabony - Data Template 37
Obrázek 21: Vytvoření triggeru v elementu Control.....38
Obrázek 22: Nastavení Property triggeru 39
Obrázek 23: Příklad Data triggeru.....39
Obrázek 24: Uživatelské rozhraní Visual Studia 2015.....40
Obrázek 25:Diagram případů užití - Databáze parkourových spotů v Praze 47
Obrázek 26: Struktura MySQL tabulek.....52
Obrázek 27: Vytvoření uživatele MySQL databáze.....53
Obrázek 28: Přidání MySQL referencí do Visual Studia.....54
Obrázek 29: Databázová třída pro připojení aplikace k MySQL databázi.....55
Obrázek 30: Connection string pro připojení k SQL CE databázi 56
Obrázek 31: Založení nové SQL CE databáze 57
Obrázek 32: Vytvoření SQL CE tabulky pomocí jazyka SQL.....58
Obrázek 33: Struktura SQL CE tabulek 59
Obrázek 34: Databázová třída pro připojení aplikace k SQL CE databázi 60
Obrázek 35: Hlavní okno aplikace 61
Obrázek 36: Okno detailu spotu 62
Obrázek 37: Okno galerie spotu 63
Obrázek 38: Okno pro přihlášení a registraci uživatele 64
Obrázek 39: Okno pro vytvoření nového spotu 65
Obrázek 40: Okno načítání aplikace.....66
Obrázek 41: Okno pro zobrazení zprávy.....66

Seznam tabulek

Tabulka 1: Seznam databázových tabulek 50
Tabulka 2: Nejdůležitější metody a události třídy MainWindow 70
Tabulka 3: Nejdůležitější metody a události třídy PrihlaseniRegistrace.....70
Tabulka 4: Nejdůležitější metody a události třídy NovySpot 71
Tabulka 5: Nejdůležitější metody a události třídy SplashScreen 71
Tabulka 6: Nejdůležitější metody a události třídy CustomMessageBox.....71
Tabulka 7: Nejdůležitější metody a události třídy MySqlDatabaze.....72
Tabulka 8: Nejdůležitější metody a události třídy SqlCeDatabaze.....72

Seznam použitých zkratek a symbolů

CLR - Common Language Runtime

VB.NET - Visual Basic .NET

MSIL - Microsoft Intermediate Language

XML - Extensible Markup Language

CLS – Common Language Specification.

OOP - Objektově orientované programování

WPF - Windows Presentation Foundation

XAML - Extensible Application Markup Language

DIP - Device Independent Pixel

SQL CE - SQL Server Compact Edition

1 Úvod

Využití informačních technologií v dnešní době představuje silný nástroj pro předávání informací. Jedním možným způsobem předání těchto informací jsou vytvořené aplikace, které uchovávají na jednom snadno dostupném místě údaje o zvoleném tématu. Tato bakalářská práce se zabývá vytvořením formulářové aplikace Databáze parkourových spotů v Praze, dostupné pro platformu Windows.

Toto téma jsem si zvolil, neboť sám již několik let provozuji parkour a jedním z hlavních problémů pražské parkourové komunity je nedostatek míst k tréninku v Praze. Každým rokem přibývá více zájemců o tento sport, kteří nevědí, kde začít trénovat. Aplikace jejím uživatelům, zejména samotným parkouristům, poskytuje jednotné místo, kde jsou shromážděny všechny potřebné informace o tréninkových parkourových místech v Praze, tzv. spotech. Aplikace, vytvořená využitím nejnovějších technologií pro tvorbu desktopových aplikací, by měla přinášet značné zlepšení v možnostech tréninků v Praze pro začínající i zkušené parkouristy a podávat jim srozumitelné a potřebné informace o místech vhodných pro trénink.

V teoretické části práce se věnuji technologiemi, které jsou nezbytné pro vývoj aplikace. Naprogramování aplikace vyžaduje znalosti několika technologií, které se podílejí na funkčnosti aplikace. Je zapotřebí zvolit programovací jazyk, na kterém je aplikace postavena, prostředek pro tvorbu grafického uživatelského rozhraní, technologii pro uchování dat a vhodné vývojové prostředí pro tvorbu aplikace. Základní principy, popis a využitelnost těchto technologií jsou uvedené v této části práce.

V části praktické vysvětlím nezbytné kroky k vytvoření databázové aplikace, od návrhu po vytvoření funkční aplikace, pomocí využitých technologií představených v teoretické části práce, které spolu musí vzájemně komunikovat.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je zaměřena na problematiku vývoje databázové aplikace v jazyce C#. Pro vytvoření grafického uživatelského rozhraní je vyžita technologie WPF.

Hlavním cílem práce je návrh a implementace databázové aplikace, sloužící k evidenci míst pro trénink parkouru v Praze, pro operační systémy Windows. Dalším cílem je zajištění funkčnosti aplikace jak s funkčním internetovým připojením, tak i bez internetového připojení. Aplikace umožňuje snadné ovládání a podává přehledné informace o tréninkových místech, čímž umožní zlepšení kvality tréninku parkouru ve městě Praha pro její uživatele.

2.2 Metodika

Metodika zpracování teoretické části práce je založena na studiu odborných informačních zdrojů. Na základě zjištěných poznatků budou popsány jednotlivé nástroje a technologie související se zpracováním hlavního cíle práce.

V praktické části práce bude proveden návrh datového modelu a aplikace samotné s využitím běžných postupů softwarového inženýrství. Poté bude na základě návrhu provedena implementace aplikace.

Při tvorbě aplikace bude pro zajištění logiky využit programovací jazyk C# na platformě .NET. Technologie WPF, společně se značkovacím jazykem XAML, poslouží k vytvoření uživatelského rozhraní. Pro uchování dat budou k dispozici relační databázové systémy MySQL a SQL Server Compact Edition.

Výsledná aplikace bude otestována na operačních systémech Windows. Provede se kontrola potencionálních chyb, kompatibility, funkcionality aplikace a budou navrženy možnosti případného dalšího rozšíření.

3 Teoretická východiska

3.1 .NET Framework

.NET Framework je softwarová platforma, vytvořená společností Microsoft, sloužící k vývoji a běhu aplikací. Kromě klasických aplikací spustitelných na operačním systému Windows lze .NET Framework využít i pro vývoj webových aplikací a služeb.

.NET Framework funguje doslova jako substrát, na kterém lze pěstovat software. Jeho jádro je založené na principech objektově orientovaného programování a všechny základní služby zpřístupňuje široké škále programovacím jazykům. .NET Framework automaticky podporuje třídy, metody, vlastnosti, konstruktory, události, polymorfismus atd. Ve výsledném efektu to znamená, že není podstatné, ve kterém programovacím jazyce komponenty vytváříme případně, jaké komponenty používáme (Běhálek, 2007).

3.1.1 **Architektura .NET Frameworku**

Na nejnižší úrovni architektury .NET Frameworku nad operačním systémem Windows je realizována základní infrastruktura. Tato úroveň se nazývá *CLR – Common Language Runtime*. *CLR* zajišťuje kompilaci zdrojových souborů použitého programovacího jazyka (*C#, VB.NET, ...*) na intermediární jazyk *MSIL – Microsoft Intermediate Language*. Další úroveň obsahuje širokou řadu knihoven. Knihovny jsou organizované do jmenných prostorů. První základní a důležitou knihovnou je *Base Class Library*. Tato knihovna obsahuje celou základní sadu knihoven, které se starají o přístup k souborům, manipulaci s řetězci, kolekce nebo například formátování. Dále Framework obsahuje knihovny pro práci s daty (*ADO.NET*) a *soubory XML*. Nad touto úrovní se nacházejí knihovny pro práci s uživatelským rozhraním – *Windows Form, Windows Presentation Formation* nebo *ASP.NET*. Nejvyšší vrstvu tvoří programovací jazyky, které definuje *CLS – Common Language Specification*.

Jestliže budete stavět pouze takové .NET typy, které vystavují CLS kompatibilní funkce, je jisté, že všechny jazyky pracující s .NET tyto typy dokáží přečíst. Naopak, pokud budete používat datový typ nebo programovací konstrukt, který je mimo hranice CLS, nelze garantovat, že každý .NET programovací jazyk dokáže pracovat s vaší .NET knihovnou kódu (Troelsen, 2010).

.NET Framework je podporován širokou škálou programovacích jazyků. Nejpoužívanější programovací jazyky pracující s tímto frameworkem jsou: *C#, C++, VB.NET, J#, F#* a jiné.

3.1.2 Výhody .NET Frameworku

Využití .NET zajišťuje výhody v těchto aspektech:

- **Zpracování chyb** - chyby se zpracovávají pomocí takzvaných výjimek a poté je zajištěno následné zpracování těchto výjimek.
- **Verzování a digitální podpisy knihoven** - řeší tzv. *DLL hell* problém. Tento problém nastává, když více aplikací využívá jednu *DLL knihovnu* v různých verzích. To má za následek přepsání jedné knihovny nejnovější verzí a aplikace, která používá verzi starší, se stává nefunkční.
- **Vylepšení správy paměti** – o správu paměti se stará *Garbage Collector*, jenž monitoruje paměť a zdroje aplikace, které nazpět předává systému, když už pro ně aplikace dále nemá využití.
- **Možnost využití knihoven, tříd, funkcí** – při tvorbě aplikací má programátor možnost využít volání kódu dostupného z .NET Frameworku, namísto zdlouhavého psaní vlastního kódu řešícího stejný problém.

3.1.3 Verze .NET Frameworku

Vydání každé nové verze je spjaté s používaným vývojovým prostředím *Visual Studia*. Verze 1.0 .NET Frameworku byla uvolněna v únoru 2002. Od té doby bylo uvedeno dalších 11 verzí. Nejnovější verzí je .NET Framework 4.6.2, která byla vydána s vývojovým prostředím Visual Studio 2012 Update 1 v srpnu 2016

Do verze 3.0 lze pro každou verzi frameworku používat jen určité vývojové prostředí. To se změnilo s vydáním verze 3.5 a novým Visual Studiem 2008, který podporuje funkci *multitargetting* – možnost psaní aplikací i pro starší verze .NET Frameworku.

Microsoft nabízí .NET Framework na svých oficiálních webových stránkách zdarma ke stažení pro všechny uživatele operačních systémů Windows.

3.2 Programovací jazyk C#

C# (vyslovováno C Sharp) je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft. Tento programovací jazyk byl představen spolu s vývojovým prostředím .NET a později schválený standardizačními komisemi ECMA a ISO. Jazyk je založen na starších programovacích jazycích C, C++ a je také velmi blízký jazyku Java. Vlastnosti jazyka C# vychází z funkcí .NET.

Jazyk C# se využívá ke tvorbě formulářových, databázových nebo webových aplikací a webových služeb.

3.2.1 **Základy jazyka C#**

Syntaxe vychází z jazyka C. C# je case sensitive, což znamená rozlišování mezi velkými a malými písmeny pro pojmenovávání jednotlivých prvků. Pro pojmenovávání prvků jazyka jsou definovány konvence. Konvence pojmenovávání vychází z .NET Frameworku. Jména *rozhraní*, *tříd*, *vlastností*, *namespace*, *assembly* začínají velkým písmenem, *parametry* a *proměnné* začínají malým písmenem, *private* a *protected* proměnné začínají podtržítkem a posléze písmenem malým.

C# je silně typovaný jazyk. Knihovna tříd rozhraní .NET Frameworku definuje sadu s předdefinovanými typy.

Každý typ obsahuje následující informace:

- Úložný prostor pro proměnnou daného typu.
- Minimální a maximální hodnoty.
- Základní typ, ze kterého dědí.
- Povolené operace.
- Místo přidělení paměti pro proměnnou.
- Členy, které obsahuje.

„Proměnná reprezentuje místo uložení, které má modifikovatelnou hodnotu. Proměnná může být lokální proměnná, parametr (hodnota, ref nebo out), atribut (instanční nebo statický) nebo element pole“ (Albahari a Drayton, 2010). Všechny proměnné musí mít nejdříve deklarovaný *datový typ*.

„Datové typy jsou sady (rozmezí) hodnot, které mají podobnou charakteristiku. Například datový typ byte specifikuje sadu integerů v rozmezí [0 … 255]“ (Nakov a Kolev, 2013).

Datové typy se dělí na typy hodnotové a referenční. Hodnotové datové typy se dále dělí na celočíselné datové typy (*byte*, *int*, …) a typy s pohyblivou desetinnou čárkou – desetinná čísla (*float*, *double*, …). Z referenčních datových typů lze uvést datový typ *string*. Pro zjištění pravdy / nepravdy se využívá datový typ *bool*.

```
// proměnné mojePromenna a mojepromenna nejsou stejné, C# je Case Sensitive
string mojePromenna;
mojePromenna = "text";
string mojepromenna;
mojepromenna = "text 2";
```

Obrázek 1: Deklarace proměnné a přiřazení hodnoty

Stejně jako v jiných programovacích jazycích existují i zde *řídící struktury*, které řídí tok programu. Jsou jimi:

- **Podmíněné výrazy** – if, switch
- **Iterace / Cykly** – for, do, while, foreach
- **Skoky** – goto, break, return, yeild

```
// If
int cislo = 0;
string zprava;
if (cislo > 0) { zprava = "Číslo je v intervalu (0; nekonečno>"; }
else if (cislo < 0) { zprava = "Číslo je v intervalu <-nekonečno; 0)"; }
else { zprava = "Číslo se rovná 0"; }

// For cyklus
for (int i = 1; i <= hodnoceni; i++) { mojePromenna++; }
```

Obrázek 2: Řídící struktury

3.2.2 Objektově orientované programování v C#

Objektově orientované programování, zkráceně OOP, je způsob designu a implementace aplikací s důrazem na znuvupoužitelnost. Dochází ke snaze, co nejlépe nasimulovat realitu. Pro tento účel se využívají objekty. Aplikace využívající OOP je díky simulaci reality a objektům,

z pohledu programátorů, lépe čitelná a přirozenější. Vývoj aplikací by měl být snazší. Snazší by měla být i údržba a pomocí OOP by se mělo docílit menší chybovosti.

Základní jednotkou OOP je *objekt*. Který si lze představit jako jakoukoliv věc z naší reality – např.: televize nebo pes. Každý objekt obsahuje své vlastnosti a metody.

„Abychom popsali objekt, soustředíme se na jeho charakteristiky […]. Tyto charakteristiky objektu reálného světa uchováváme v deklaraci třídy ve speciálních typech proměnných. Tyto proměnné se nazývají vlastnosti a drží stavy objektu“ (Nakov a Kolev, 2013). Objekt televize může mít vlastnosti – úhlopříčka, váha, rozlišení a jiné. Tyto vlastnosti jsou reprezentovány v programu daty ve formě *proměnných*, které objekt uchovává.

„Metoda je základní část programu. Dokáže řešit určité problémy a eventuálně brát parametry a vracet výsledek“ (Nakov a Kolev, 2013). Lze si je představit jako schopnosti, které objekt dokáže vykonat.

Objekt je vytvořený podle šablony, předpisu. Této šabloně se říká *třída*, která definuje vlastnosti a schopnosti objektů. Objekt vytvořený podle této šablony se nazývá *instance třídy*. Instance třídy mají stejný předpis, liší se ve svých datech. Při vytváření instance se volá tzv. *konstruktor*. „Konstruktor nevrací žádnou hodnotu (ani void). Každá třída má definovaný implicitně konstruktor bez parametrů a s prázdným tělem. Každý objekt by pak měl být vytvořen pomocí operátoru new“ (Běhálek, 2007).

```
public class Auto // Třída
{
   public string Znacka { get; set; } // Vlastnosti
   public string Nazev { get; set; }
   public Auto(string znacka, string nazev) // Konstruktor
   {
      this.Znacka = znacka;
      this.Nazev = nazev;
   }
   public void VypisAuto() // Metoda
   {
   Console.WriteLine(this.Znacka + " " +  this.Nazev);
   }
}
Auto prvniAuto = new Auto("Škoda", "Octavia"); // Vytvoření instance třídy Auto
```

Obrázek 3: Příklad třídy v jazyce C#

Ačkoliv jsou standardní třídy dle koncepce objektově orientovaného programování všestranným nástrojem pro generování instancí jistého typu, v některých případech se může nutnost zakládání instancí jevit jako těžkopádná nebo nepříliš efektivní. Programátorům se při řešení složitých problémů z oblasti informatiky a softwarového inženýrství nezřídka stává, že potřebují implementovat funkcionalitu, která se ovšem pojí spíše se samotnou třídou než s jejími instancemi. Za těchto okolností se může jevit jako dobrý nápad použít speciální třídu, která definuje pouze statické členy. Takovou třídu nazýváme statickou (Hanák, 2009).

Ve *statické třídě* nelze tvořit instance. Musí obsahovat pouze statické proměnné a metody. Statické metody slouží jako pomocné metody. Např.: metoda pro zaokrouhlení čísla. Nevoláme je pomocí instancí, ale pomocí třídy. Statické proměnné jsou takové proměnné, které se nevážou na instanci třídy, jelikož tu v tomto případě vytvořit ani nelze, ale které ke třídě logicky patří.

```
public static class MojeStatickaTrida
{
    public static string Vlastnost { get; set; }
    public static void MojeMetoda(int parametr) { ... }
}
MojeStatickaTrida.MojeMetoda(1);
string vlastnost = MojeStatickaTrida.Vlastnost;
```

Obrázek 4: Statická třída a přístup k prvkům třídy

„Deklarace třídy, struktury, rozhraní může být nyní umístěna do více fyzických souborů. Teprve kompilátor se postará o jejich spojení. Takto připravená třída je pak rozlišena pomocí klíčového slova Partial“ (Pirkl, 2005). Tyto třídy se chovají stejně jako jedna třída, pouze jsou rozdělené do více souborů.

Důležitou součástí koncepce OOP je *zapouzdření*. To nám zajišťuje skrýt určité vlastnosti a metody objektu, tak aby byly dostupné pouze uvnitř třídy. Uživatel nepotřebuje vědět, jak to uvnitř objektu funguje, pouze potřebuje, aby objekt dělal to, co má dělat. Z toho vyplívá, že objekty se mezi sebou chovají jako uzavřené celky. Mezi sebou se ovlivňují jen instance tříd.

Podle úrovně přístupnosti aplikace k prvkům třídy lze rozčleňovat atributy a metody na *soukromé* a *veřejné*. Zde hovoříme o tzv. *modifikátorech přístupu*. K *soukromým* (*private*) atributů a metodám nemůže aplikace po vytvoření instance přistupovat, nevidí je. Jsou přístupné pouze uvnitř třídy. Na druhé straně *veřejné* (*public*) atributy a metody aplikace vidí, tudíž s nimi

může pracovat. K *protected* atributům může přistupovat sama třída a třídy odvozené pomocí *dědičnosti*. Více o *dědičnosti* v následujícím odstavci. Posledním typem jsou *internal* atributy. K těm lze přistupovat v rámci jedné *assembly* (v rámci projektu / jedné C# .NET aplikace).

Dalším pilířem OOP je *dědičnost*. *Dědičnost* zajišťuje možnost vytvořit jednu třídu na základě druhé, z které může dědit metody a vlastnosti. Při vytváření nové třídy, která obsahuje vše, co obsahuje již existující třída a přidává pouze několik svých vlastních metod a atributů, můžeme dědit z hotové třídy a nemusíme tedy psát celou novou třídu od začátku s duplicitním kódem. V C# může být každá třída potomkem pouze jedné třídy, nelze využít *vícenásobné dědičnosti* - dědit od dvou a více tříd zároveň.

Lze využívat i *polymorfismu*. *Polymorfismus* zaručuje jednotné rozhraní pro práci s různými typy objektů. To znamená, že každá podtřída může překrýt metodu definovanou ve svém předkovi a tím docílit jiného chování. Metoda je definována se stejnou hlavičkou, ale jiným vnitřním kódem. *Překrývání metod* se někdy zaměňuje s *přetěžováním metod*. *Přetěžování metod* znamená vytvoření nové metody se stejným názvem, jako má již existující metoda, ale jiným počtem nebo typem parametrů.

```
public class Zvire { // Třída, ze které se bude dědit
    protected string Jmeno { get; set; } // vlastnosti
    protected string Druh { get; set; }
    protected int Vek { get; set; }
    // konstruktor
    public Zvire(string jmeno, string druh, int vek) {
            Jmeno = jmeno; Druh = druh; Vek = vek;
    }
    // Metody
    public virtual string VypisZvire() {
        return Jmeno + " " + Druh + " " + Vek;
    }
    public string VypisTridu() {
        return "Metoda VypisTridu() z třídy 'Zvire'";
    }
}
public class Tygr : Zvire { // Dědící třída
    private string Smecka { get; set; }
    // konstruktor, doplnění vlastnosti Smecka
    public Tygr(string jmeno, string druh, int vek, string smecka) : base(jmeno,
    druh, vek) {
        Smecka = smecka;
    }
    // využití polymorfismu - přepsání metody
    public override string VypisZvire() {
        return Jmeno + " " + Druh + " " + Vek + " " + Smecka;
    }
    public string VypisSmecku() {
        return "Tento tygr se nachází ve smečce: " + Smecka;
    }
}
// vytvoření instance třídy Zvire a Tygr
Zvire mojeZvire = new Zvire("Pepa", "Zebra", 7);
Tygr mujTygr = new Tygr("Jarda", "Tygr", 4, "Smečka A");
Zvire mujTygr2 = new Tygr("Jarda", "Tygr", 4, "Smečka A");
MessageBox.Show(mujTygr.VypisTridu()); //vypsání třídy této metody
mojeZvire = mujTygr; // uložení tygra do zvirete
MessageBox.Show(mujTygr2.VypisZvire()); // vypsání tygra 2
```

Obrázek 5: Dědění a polymorfismus

Objektově orientované programování umožňuje vytvářet *rozhraní* neboli i*nterface*. „Interface není nic víc, než pojmenovaná sada abstraktních členů“ (Troelsen, 2010). Rozhraní se implementuje pomocí dědění a implementací jeho metod a atributů. Třída dědící z rozhraní musí využít vše, co je v rozhraní definováno. Je možné využít vícenásobné dědění z rozhraní – třída dědí ze dvou a více rozhraní zároveň.

```
interface IZvire {
  int Vaha { get; set; }
  string VypisZvire();
  string VypisTridu();
}
public class Zvire : IZvire {
   public string Vaha { get; set; }
   public virtual string VypisZvire()  {
      return Jmeno + " " + Druh + " " + Vek;
   }
   public string VypisTridu() {
      return "Metoda VypisTridu() z třídy 'Zvire'";
   }
}
```

Obrázek 6: Rozhraní a jeho implementace

Dále se využívají *abstraktní třídy*. Ty si lze představit jako něco mezi rozhraním a třídou. Oproti rozhraní lze v abstraktní třídě definovat pro definovaný prvek i implementaci. Při definici prvků v abstraktní třídě musí před typem prvku následovat slovo *abstract*. Při vytváření prvků v třídě, která dědí z třídy abstraktní, musí tyto prvky obsahovat slovo *override*.

```
public abstract class AbstraktniTrida {
    public abstract int Cislo { get; set; }
    public abstract string Metoda2(string popis);
}
public class MojeTrida : AbstraktniTrida {
    public override int Cislo { get; set; }
    public override string Metoda2(string popis) {
        throw new NotImplementedException();
    }
}
```

Obrázek 7: Abstraktní třáda

Objektům, které reprezentují reference na metody, se říká *delegáti*. Může se jednat jak o metody instanční, tak i o metody statické. Využití delegáta je popsáno na následujícím obrázku.

```
delegate void MujDelegat(); // definování typu delegáta
MujDelegat mojeMetoda = null; // vytvoření objektu delegáta a přiřazení metody
mojeMetoda += VolanaMetoda;
mojeMetoda();// zavolání metody pomocí našeho delegáta

public void VolanaMetoda() // metoda, která bude volaná pomocí delegáta
{
    MessageBox.Show("Moje Metoda");
}
```

Obrázek 8: Definování a vytvoření delegáta

3.3 Technologie WPF

„Windows Presentation Foundation je nezbytnou složkou pro budoucnost vývoje aplikací, umožňuje vývojářům převzít kontrolu nad silou počítače a vytvářet bohaté, interaktivní a médii podporované uživatelské rozhraní“ (Moroney, 2006).

Jedná se o framework umožňující vytváření formulářových aplikací. Je součástí .NET Frameworku od verze 3.0. Podpora WPF je zajištěna na operačních systémech od Microsoftu počínaje od Windows Vista až po Windows 10. Pro starší verze operačních systému (Windows XP) lze WPF doinstalovat.

Pro tvorbu aplikace se využívá celá řada již hotových *ovládacích prvků* (mřížka, tlačítka, tabulky, posuvníky, pole, menu a jiné). Lze vytvářet i vlastní ovládací prvky.

„WPF aplikace se často potřebují adaptovat na různé velikosti oken na různých rozlišeních. Navržení oken pomocí sloupců a řádků mřížky aplikace umožňuje vašemu programu automatické přizpůsobení velikosti okna“ (Stellman a Green, 2007).

WPF využívá lepší způsob oddělení uživatelského rozhraní a logiky aplikace. Uživatelské rozhraní je zajištěno pomocí nově vytvořeného značkovacího jazyka XAML. Více o XAML v kapitole Značkovací jazyk XAML. Logika aplikace je vyjádřena pomocí programovacího jazyku – C#, Visual Basic a jiných. Spojení těchto dvou částí má na starost tzv. *Data Binding*. Více o Data Bindingu v kapitole Data Binding).

3.3.1 **Grafika**

Technologie se pořád vyvíjejí a s nimi jsou i nároky na bohatší grafiku aplikací. Veškerá vytvořená grafika je zajištěna pomocí *Direct3D* knihoven a hardwarové akcelerace grafické karty.

> Grafické karty se liší v jejich podpoře renderování specializovaných funkcí a optimalizací. Toto však není problémem ze dvou důvodů. Zaprvé, většina moderních počítačů obsahuje grafický hardware, který je více než dostačující pro WPF funkce jako jsou 3D kreslení a animace. Toto platí i o noteboocích a počítačích s integrovanou grafikou. Zadruhé, WPF obsahuje softwarovou pojistku pro vše, co dělá. To znamená, že je WPF dostatečně inteligentní pro využívání hardwarové optimalizace, kdekoliv je to možné, avšak dokáže provést stejnou práci pomocí softwarových kalkulací, pokud je to zapotřebí (MacDonald, 2012).

K dispozici jsou pokročilejší grafické schopnosti oproti zastaralejším způsobům vykreslování grafiky. WPF zavádí vektorovou grafiku, novou jednotku délky tzv. DIP (Device Independent Pixel). DIP se vypočítá na základě hodnoty DPI (Dots per inch – počet obrazových bodů na palec) systému uživatele a velikosti 1 DIP, která je stanovena jako 1/96 palce.

Podpora audio a video medií je také zajištěna. Všechny nejpopulárnější formáty videa jsou podporovány. „WPF zahrnuje podporu pro přehrávání jakéhokoli audio nebo video souboru podporovaného Windows Media Playerem, a umožňuje přehrávat více než jeden soubor současně“ (MacDonald, 2012).

K vytváření animací již není potřeba znovu a znovu překreslovat plátno aplikace. WPF vytváří animace pomocí tzv. *Storyboardů*

Více o tvoření uživatelského grafického rozhraní je popsáno v kapitole Značkovací jazyk XAML.

3.3.2 **Porovnání WinForms a WPF**

WPF lze označit za nástupce starší technologie WinForms. V dnešní době se současně využívají oba dva frameworky, avšak WPF je technologicky mnohem vyspělejší. V následujícím seznamu jsou uvedené některé rozdíly a hlavně výhody, některé již dříve zmíněné, WPF oproti WinForms:

- WPF využívá akcelerovanou grafiku a vykreslování pomocí *Direct3D*, které přináší větší výkon.
- WPF zavedlo nové jednotky délky DIP, které se lépe přizpůsobí různým druhům monitorů a obrazovek.
- Tvorba aplikací ve WinForms funguje na principu „naklikávání“ ovládacích prvků pro Windows z nabídky. U WPF se vytváří aplikace téměř z ničeho pomocí XAML kódu a vlastní grafiky, což je složitější, ale lze vytvořit aplikaci více podle představ uživatele i programátora.
- Výhodné je využití Data Bindingu, oddělení uživatelského rozhraní od logiky aplikace u WPF.
- WPF jako novější technologie lépe vyhovuje nynějším standardům.

3.4 Značkovací jazyk XAML

XAML (Extensible Application Markup Language) je rozšiřitelný značkovací jazyk, který slouží k návrhu uživatelského rozhraní aplikací pro technologii WPF. Vychází ze značkovacího jazyku XML. Jazyk XAML byl vyvinut pro usnadnění zápisu. Vše, co lze napsat pomocí programovacích jazyků C# nebo Visual Basic lze velmi přehledně zapsat i pomocí XAML. Využívá komponenty dodávané s .NET Frameworkem nebo s nějakou jinou knihovnou, na kterou se lze napojit. Při kompilaci aplikace je XAML soubor zkompilován do BAML souboru (Binary XAML - binární reprezentace XAML). Veškeré BAML jsou poté vloženy a využívány jako *resources* (*zdroje*) v konečné aplikaci.

3.4.1 Struktura a syntaxe jazyka XAML

XAML soubor, jako všechny XML soubory musí obsahovat právě jeden kořenový element. Proto není překvapením, že element objektu může obsahovat dětské elementy objektů (ne pouze elementy vlastností, které podle XAML nejsou dětmi). Element objektu může mít 3 typy dětí: hodnotu pro vlastnost obsahu, položky kolekce nebo hodnotu, která může být přetypována na element objektu (Nathan, 2010).

Každý dětský vnořený element může, ale nemusí obsahovat další vnořené elementy. Elementy se zapisují do lomených závorek. V závorkách je uveden název elementu. Elementy se dělí na párové a nepárové. Některé lze zapsat obojím způsobem.

```
<TextBlock>Text elementu</TextBlock> // Ukončení párového elementu
<TextBlock Text="Text Elementu" /> // Ukončení nepárového elementu
```

Obrázek 9: Párové a nepárové elementy v XAML

Elementy obsahují atributy / vlastnosti. Např.: Text, Content, FontWeight, Background nebo Click. Tyto vlastnosti mohou popisovat grafický vzhled (FontWeight), přiřazovat obsah (Content, Text) nebo přiřazovat události (Click, o událostech více v kapitole – Události). Atributů existuje celá řada a díky nim lze každý prvek na formuláři nastavit podle představ programátora.

Velmi důležité jsou v XAML *jmenné prostory* (*namespace*). V .NET Frameworku se vyskytuje velký počet tříd a bez použití jmenných prostorů by každá třída musela mít unikátní název.

Očividně není dostatečné poskytovat pouze názvy tříd. XAML parser musí také znát .NET jmenný prostor, ve kterém je třída lokalizována. Například třída Window může existovat na různých místech – může odkazovat na Systém.Windows.Window třídu nebo na třídu Winow komponent třetích stran nebo na třídu vámi definovanou v aplikaci. K vyřešení jakou třídu chcete zvolit XAML parser zkoumá XML namespace, který je aplikovaný na element (MacDonald, 2012).

Jmenné prostory v XAML využíváme například pro používání elementů (*TextBlock*, *Button*, …), definici parametrů nebo pro vytváření vlastních komponent. Jmenné prostory se nastavují v elementu *Window*. Zde se nastaví, jaké komponenty se budou používat a z jakých jmenných prostorů je možné komponenty vybírat. Pro využívání více komponent v jednom formuláři lze komponentám přiřadit tzv. alias a tím je od sebe odlišit.

„Deklarace namespace se vždy aplikuje pro všechny vnořené elementy XAML dokumentu, stejně jako u dokumentu XML. Z tohoto důvodu není nutné deklarovat v XAML dokumentu stejný namespace dvakrát pro užití jeho typů. Avšak je potřeba deklarovat tyto typy s korespondujícím prefixem elementu“ (Garofalo, 2011).

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

Obrázek 10: Defionování jmenného prostoru v XAML

První se uvádí klíčové slov *xmlns*, které znamená XML namespace. Následuje dvojtečka a zkratka pro namespace (zde „x“), za rovnítkem se uvádí v uvozovkách plné jméno jmenného prostoru (nejedná se o URL adresu). Nyní lze tuto zkratku „x“ využívat pro element Window a všechny jeho vnořené elementy. Tento jmenný prostor uchovává definice rozšiřujících parametrů.

```
<Window x:Class="Aplikace.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:Aplikace"
        Title="Název okna" Height="300" Width="300">
    <Grid x:Name="aplikaceGrid">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="40"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Row="0" Grid.ColumnSpan="2">
            <TextBlock Text="První Text - vlevo" FontWeight="Bold"/>
            <StackPanel Margin="0 10">
              <TextBlock Text="Text A – střed" HorizontalAlignment="Center" />
            </StackPanel>
        </StackPanel>
        <Border Background="#000" Grid.Row="1" Grid.ColumnSpan="2" />
        <Button Content="1. tlačítko" Grid.Row="1" Click="Udalost1"/>
    </Grid>
</Window>
```

Obrázek 11: Ukázka struktury a syntaxe XAML kódu

3.4.2 Data Binding

Data Binding je nedílnou součástí WPF. Je to technika, která propojuje dva zdroje dohromady a zachovává synchronizaci mezi těmito zdroji.

„Po identifikování zdroje dat je potřeba založit spojení mezi logikou a uživatelským rozhraním. Jednoduše řečeno to znamená vytvoření spojení mezi daty ve zdroji dat and grafickým elementem. Data Binding je proces navázání tohoto spojení“ (Dalal a Ghoda, 2011).

Prvním způsobem využití je synchronizace zdrojů mezi dvěma elementy. Příklad: Máme dva elementy – TextBox a TextBlock. Pomocí data bindingu svážeme tyto dva elementy. Na základě změny textu TextBoxu se automaticky mění text TextBlocku.

„Cílový objekt bindingu je objekt, kterému chceme poskytnout data. Obvykle se jedná o element jako je TextBox, ListBox nebo DataGrid. Cílová vlastnost by měla zobrazit a umožnit změnu dat“ (Dalal a Ghoda, 2011).

Syntaxe data bindingu se píše do složených závorek. Začíná se klíčovým slovem *Binding*. První vlastnost je *ElementName* a rovná se jménu zdrojového elementu. Vlastnosti se oddělují mezerami. Poté následuje nastavení vlastnosti *Path*. Ta se rovná vlastnosti ve zdrojovém elementu, podle kterého se obsah atributu v cílovém elementu upraví. Pro zajištění *zpětné synchronizace* (zdrojový element se změní po změně cílového elementu) stačí připsat vlastnost *Mode=TwoWay*. Pokud nastane situace, že jeden element se neaktualizuje okamžitě, např. při zpětné aktualizaci při použití vlastnosti *Mode* s hodnotou *TwoWay*, je možné vynutit aktualizaci vlastností *UpdateSourceTrigger= PropertyChanged*.

```
<TextBox Name="zdrojovyTextBox" />
<TextBlock Text="{Binding Path=Text, ElementName=zdrojovyTextBox}" />
```

Obrázek 12: Data Binding mezi dvěma elementy

Druhý způsob se využívá při bindingu objektu, kdy se nejedná o XAML element. Atribut elementu se může vázat na statický zdroj (*static resource* – zdroj, který je načtený pouze jednou) nebo na vlastnost, která zajišťuje přenesení dat z kódu (C#, Visual Basic) do uživatelského rozhraní. Podmínkou pro takovéto využití je, že data pro přenesení musí být uchována ve veřejné vlastnosti (public property). Zde se při data bindingu namísto vlastnosti *ElementName* uvádí vlastnost *Source* nebo *RelativeSource*. Jako hodnota *source* se uvádí zdrojový objekt.

```
C# kód
namespace Aplikace
{
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        private string _testovaciVlastnost;
        public event PropertyChangedEventHandler PropertyChanged;
        public string testovaciVlastnost
        {
            get { return _testovaciVlastnost; }
            set {
                _testovaciVlastnost = value;
                OnPropertyChanged("testovaciVlastnost"); }
        }
        private void OnPropertyChanged(String info)
        {
            PropertyChanged?.Invoke(this,
            new PropertyChangedEventArgs(info));
        }
        public MainWindow()
        {
            InitializeComponent();
            testovaciVlastnost = "Testovací text";
            DataContext = this;
        }
    }
}
XAML kód
<TextBlock Text="{Binding Path=testovaciVlastnost}" />
```

Obrázek 13: Data Binding – přenos dat z kódu C# do XAML elementu

Pomocí RelativeSource se lze odkázat na zdrojový objekt, který má určitý vztah k cílovému objektu. Objekt lze například odkázat sám na sebe nebo na svého rodiče. K tomu se využívá vlastnosti RelativeSource, která ve složených závorkách obsahuje klíčové slovo *RelativeSource* a vlastnost *Mode*.

```
<TextBlock Text="{Binding Path=Width,
RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Window}} }">
```

Obrázek 14: Data Binding s využitím vlastnosti RelativeSource

Pro Binding více objektů na stejný zdroj je výhodné namísto vlastnosti *Source* u každého objektu uvést *DataContext* rodičovského prvku. DataContext je vlastnost, kterou obsahuje každý ovládací prvek ve WPF. Vnořené prvky mohou DataContext dědit od svého předchůdce ve stromové struktuře. Prvky zdědí DataContext a při bindingu již nemusí mít vyplněnou vlastnost Source, ale jen vlastnost Path.

```
StackPanel DataContext="{x:Static SystemFonts.CaptionFontFamily}">
   <TextBlock Text="{Binding Path=Source}" />
   <TextBlock Text="{Binding Path=Source}" />
   <TextBlock Text="{Binding Path=Source}" />
</StackPanel>
```

Obrázek 15: Využití DataContextu – vypsání stejného zdroje do více elementů

3.4.3 Události

Všechny ovládací prvky formuláře obsahují události, na které mohou reagovat. Tyto události vyvolává uživatel aplikace nějakou akcí. Mezi nejčastější události se řadí – Click, MouseDown, MouseEnter, KeyDown, LostFocus, Loaded, SizeChanged a další. Po vykonání této činnosti se spustí přidružená funkce. Událost definovaná v jazyce XAML se vyznačuje klíčovým slovem dané události (např. Click) a po rovnítku následuje jméno spouštěné funkce.

V C# lze využít přiřazení události jen za pomocí kódu, bez využití XAML, což lze využít například pro dynamicky vytvořené elementy. Metody jsou v tomto případě implementovány pomocí delegátů. Delegát je datový typ, který umožňuje referencovat metodu. Metoda obsluhující událost je tzv. *event handler* (*obslužná metoda*). Tato metoda obsahuje parametry, které odpovídají definici delegáta. Prvním parametrem je objekt vyvolávající událost. Druhým parametrem je parametr typu třída *EventArgs* nebo třída odvozená. Tento parametr obsahuje doplňující informace pro obslužnou metodu. Odkaz na obslužnou metodu lze do událost vložit pomocí operátoru „+=“ a odebrat pomocí operátoru „-=“.

3.4.4 **Styly**

Při vytváření uživatelského grafického rozhraní je většinou snaha sjednotit vzhled výsledné aplikace. Potřebuje se zachovat podobný nebo stejný vzhled pro více ovládacích prvků najednou. „Styly poskytují výhodnou cestu k seskupení sady vlastností (a triggerů) pod jeden objekt, a selektivně (nebo automaticky) tento objekt aplikovat elementům“ (Yosifovich, 2012). Každý styl je pojmenovaný pomocí atributu *x:Key*, jenž představuje jméno stylu a podle jména lze poté ke stylu přistupovat. Nepovinným atributem je *TargetType*, který udává, na jaký ovládací prvek se styl vztahuje. Když není TargetType vyplněný musí se vždy provádět typová kontrola elementu. Atribut x:Key lze vynechat, poté jsou styly aplikované pro všechny elementy uvedené v TargetType.

V elementu style se nachází elementy *Setter*, které udávají, co se bude u ovládacího prvku měnit. Setter obsahuje vlastnost *Property*, která definuje vlastnost prvku, kterou chceme upravit (pozadí, velikost, odsazení, barva textu, …). Hodnota této vlastnosti se nastaví konkrétní hodnotou ve vlastnosti *Value*.

Styly se ukládají do *Resources* (*zdrojů*). Definici stylu lze vytvářet na různých úrovních aplikace. Nejnižší úrovní je definice stylů u příslušného elementu. Styl je tímto způsobem přístupný jen pro tento element a pro všechny jeho vnořené elementy. Pro element StackPanel se styly ukládají do *StackPanel.Resources*. Pro zpřístupnění stylu celému oknu (Window) aplikace je definice stylu provedena na úrovni elementu Window. Styly se definují ve *Window.Resources*.

```
<StackPanel>
   <StackPanel.Resources>
       <Style TargetType="Button">
            <Setter Property="FontSize" Value="15"/>
            <Setter Property="Background" Value="Red"/>
       </Style>
   </StackPanel.Resources>
   <TextBlock Text="Text TextBlocku" />
   <Button Content="Text tlačítka" />
</StackPanel>
```

Obrázek 16: Defionování stylu u elementu StackPanel pro všechny elementy Button

Styly dostupné pro celou aplikaci, tedy pro všechny okna a elementy, se definují v souboru App.xaml v *Application.Resources*.

```
App.xaml

<Application.Resources>
   <Style TargetType="TabItem" x:Key="mujStyl">
      <Setter Property="Background" Value="#0A73BE" />
      <Setter Property="Foreground" Value="White" />
      <Setter Property="BorderBrush" Value="#0A73BE" />
   </Style>
</Application.Resources>

Okno (Window)
<TabItem Header="Název" Style="{StaticResource mujStyl}">
```

Obrázek 17: Definování stylu v souboru App.xaml a přiřazení stylu do elementu

„Pro vytvoření nového stylu lze znovu použít již existující styl a přidat doplňující Setter element je-li to zapotřebí. Atribut BasedOn umožňuje implementaci této funkcionality“ (Dalal a Ghoda, 2011). Zde je potřeba dávat si pozor na umístění těchto dvou stylů. Styl, ze kterého se dědí, musí být nadefinovaný jako první (výše v dokumentu). Parametr *BasedOn* odkazuje na x:Key děděného stylu.

```
<!-- Styl, ze kterého dědíme -->
<Style x:Key="puvodniStyl" TargetType="Button">
     <Setter Property="Background" Value="Black" />
     <Setter Property="Height" Value="30" />
</Style>

<!-- Styl, který dědí -->
<Style x:Key="novyStyl" TargetType="Button"
    BasedOn="{StaticResource puvodniStyl}">
    <Setter Property="Background" Value="Blue" />
    <Setter Property="FontWeight" Value="Bold" />
</Style>
```

Obrázek 18: Dědění stylu

3.4.5 **Templates (Šablony)**

Pro úpravu ovládacích prvků lze využít styly, ale pro přepsání struktury prvku nebo zobrazení dat uvnitř prvku se využívají šablony. „Ovládací prvky jsou vytvořeny pomocí šablon, které je mohou měnit. Šablona se skládá z hierarchie ovládacích prvků použitých k vytvoření zobrazení ovládacího prvku, které může zahrnovat content presenter pro ovládací prvky jako jsou například tlačítka, která zobrazují obsah“ (Perkins, Hammer a Reid, 2016). Šablony jsou typu *Control Template* a *Data Template*. *Control Template* mění pouze vzhled a strukturu elementu. Data Template stanovuje, jak se budou u prvku vykreslovat jeho data. Data Template často využívají prvky: ListBox, ComboBox nebo ListView. Šablony lze definovat stejně jako styly na úrovni elementu, okna nebo aplikace.

Šablony typu Control Template se vytváří v elementu ControlTemplate. Do tohoto elementu se vkládají prvky, ze kterých se měněný prvek bude skládat. Element ControlTemplate může stejně jako element Style obsahovat vlastnosti x:Key a TargetType. Každý prvek obsahuje nějaký obsah (např.: u elementu Button je to text tlačítka). Pro změnu obsahu, lze do šablony uvést tzv. *ContentPresenter*. Ten se mění právě pomocí šablon v XAML a lze ho, po pojmenování, měnit i v kódu C#.

Control Template lze definovat:

- V souboru App.xaml.
- V elementu Style. Zde se vytvoří element Setter s vlastností Property="Template". Do elementu Setter se vloží element Setter.Value, ve kterém je definována strukturu prvku.
- V Resources okna.
- V Resources prvku (např. Button.Resources).
- Uvnitř prvku v elementu Template. Pro element Button je to Button.Template

Přiřazení šablony, pokud není definována přímo v prvku nebo ve stylu, se provádí u příslušného prvku ve vlastnosti *Template*, ve které je definováno jméno šablony. U šablon lze využít i data binding, a tak lze měnit styl prvků bez jejich přesného nadefinování v šabloně.

```
<!-- Vytvoření šablony v elementu Style -->
<Style TargetType="TabItem" x:Key="mojeSablona">
    <Setter Property="Background" Value="#0A73BE" />
    <Setter Property="BorderBrush" Value="#0A73BE" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="TabItem">
                <Border Name="Border" BorderThickness="3" CornerRadius="2">
                   <ContentPresenter x:Name="cpJmeno" ContentSource="Header" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
<TabItem Header="Mapa" Style="{StaticResource mojeSablona}">

<!-- Vytvoření šablony uvnitř prvku v elementu Template -->
<TabItem Header="Mapa">
    <TabItem.Template>
        <ControlTemplate TargetType="TabItem">
            <Border Name="Border" BorderThickness="3" CornerRadius="2">
                <ContentPresenter x:Name="cpJmeno" ContentSource="Header" />
            </Border>
        </ControlTemplate>
    </TabItem.Template>
    ...
</TabItem>

<!-- Vytvoření šablony v Resources okna -->
<Window.Resources>
    <ControlTemplate x:Key="mojeSablona" TargetType="TabItem">
        <Border Name="Border" BorderThickness="3" CornerRadius="2">
            <ContentPresenter x:Name="cpJmeno" ContentSource="Header" />
        </Border>
    </ControlTemplate>
</Window.Resources>
<TabItem Header="Mapa" Style="{StaticResource mojeSablona}">
```

Obrázek 19: Definováná šablony - Control Template

Šablony typu Data Template se vytváří v elementu Data Template, do kterého se zapisuje datová šablona.

Data Template lze definovat:

- V souboru App.xaml.
- V elementu ContentPresenter.ContentTemplate, který je uvnitř šablony Control Template.
- V Resources okna.
- V Resources prvku. Pro element ListBox je to ListBox.Resources
- Uvnitř prvku v elementu ItemTemplate. Pro element ListBox je to ListBox.ItemTemplate.

Pomocí data bindingu se vloží do ovládacího prvku data. Přiřazení šablony, pokud není definována přímo v prvku nebo v ContentPresenteru, lze provést uvedením vlastnosti *ItemTemplate*, která obsahuje jméno šablony.

```
<!-- Vytvoření datové šablony uvnitř prvku v elementu ItemTemplate. -->
<ListBox ItemsSource="{Binding}">
    <ListBox.Resources>
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="{Binding Path=Data1}" />
                <TextBlock Text="{Binding Path=Data2}" />
            </StackPanel>
        </DataTemplate>
    </ListBox.Resources>
</ListBox>

<!-- Vytvoření datové šablony v ContentPresenteru. -->
<Style TargetType="ListBox">
     <Setter Property="FontSize" Value="15"/>
     <Setter Property="Background" Value="Red"/>
     <Setter Property="Template">
         <Setter.Value>
             <ControlTemplate>
                 <ContentPresenter Content="{Binding}">
                 <ContentPresenter.ContentTemplate>
                     <DataTemplate>
                         <StackPanel>
                               <TextBlock Text="{Binding Path=Data1}" />
                               <TextBlock Text="{Binding Path=Data2}" />
                         </StackPanel>
                     </DataTemplate>
                 </ContentPresenter.ContentTemplate>
                 </ContentPresenter>
             </ControlTemplate>
         </Setter.Value>
     </Setter>
</Style>
```

Obrázek 20: Definování šabony - Data Template

3.4.6 **Triggers (Spouště)**

Triggery (triggers, spouště) dokáží reagovat na hodnoty vlastností, události nebo datové hodnoty. Po zaregistrování změny dokáží změnit hodnoty vlastností. Pomocí triggerů se jednoduchým zápisem v XAML nahradí to, co by se muselo udělat v kódu C#. Triggery lze umístit pouze do stylů a šablon. Buď do souboru App.xaml, okna aplikace nebo do daného prvku. To lze obejít vytvořením elementu Control, do kterého se umístí prázdná šablona.

„Aktivní trigger může spouštět akce, avšak někdy potřebuje být trigger složen z více podmínek, které celý trigger aktivují, pokud jsou tyto podmínky splněny. To je přesně to, čeho jsou schopné typy MultiTrigger a MultiDataTrigger“ (Yosifovich, 2012).

```
<Control x:Key="mujTrigger">
    <Control.Template>
        <ControlTemplate TargetType="Button">
            <!-- Zde umístíme kód triggeru -->
        </ControlTemplate>
    </Control.Template>
</Control>
```

Obrázek 21: Vytvoření triggeru v elementu Control

Existují Triggery:

- Property triggers
- Event triggers
- Data triggers

Property triggers reagují na *dependency properties*. To je speciální druh vlastností, které jsou vhodné pro data binding, upozorní nás při změně a dokáží dědit od nadřazených prvků. Při změně vlastností nadřazeného elementu se změní i vlastnost dependency property. Tyto triggery reagují na podmínky, jako jsou: IsMouseOver, IsFocues, IsChecked, a jiné. Property triggery se mohou využít i jako Multi triggery. Poté lze reagovat na více podmínek najednou.

```
<TextBlock Text="Ahoj">
    <TextBlock.Style>
        <Style TargetType="TextBlock">
            <Setter Property="Foreground" Value="Black"></Setter>
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Foreground" Value="Blue" />
                    <Setter Property="TextDecorations" Value="Underline" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </TextBlock.Style>
</TextBlock>
```

Obrázek 22: Nastavení Property triggeru

Event triggers jsou využívány k vytváření animací, které vznikají po vyvolání událostí.

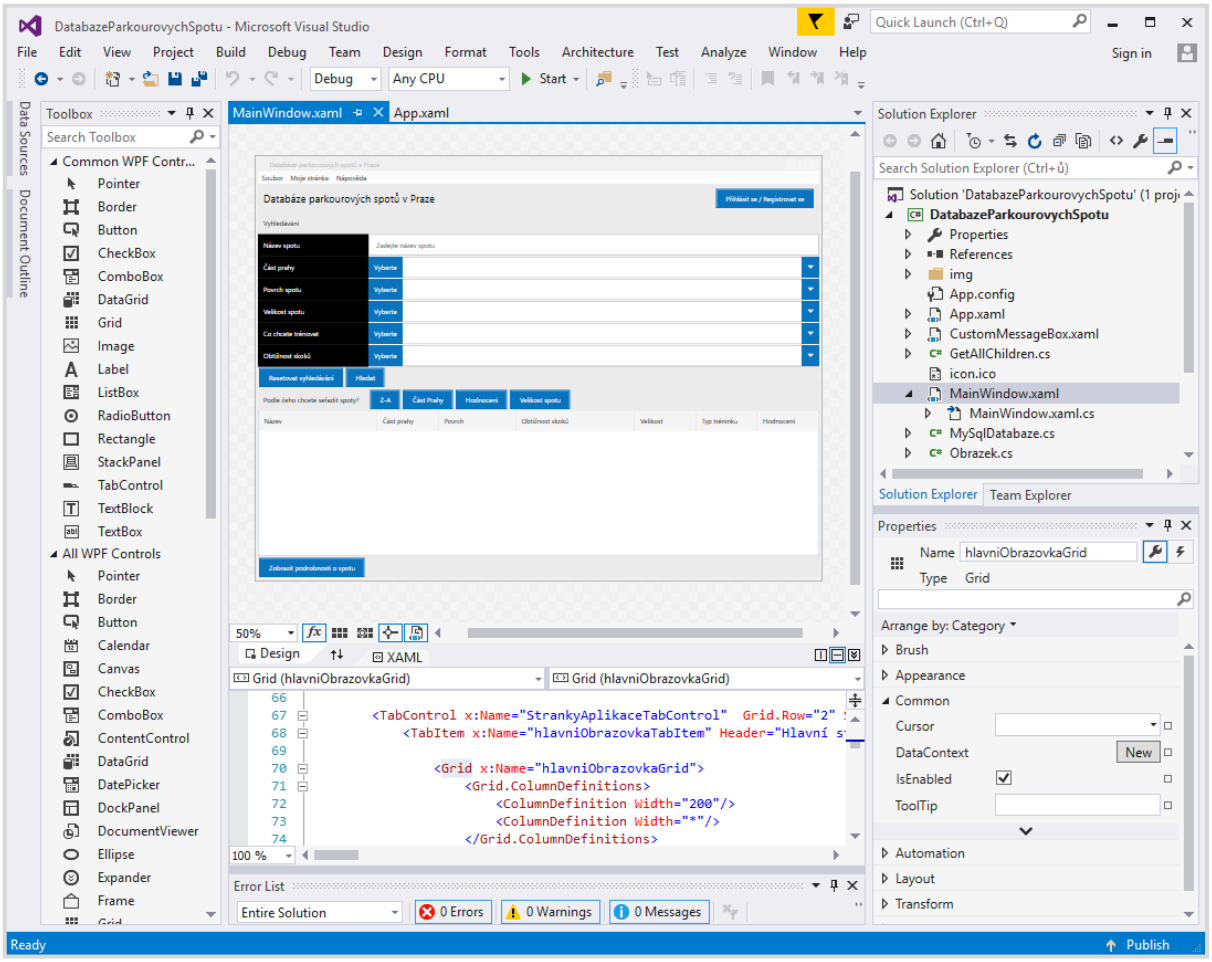
Posledním typem jsou Data triggers. Využívají se i na jiné vlastnosti, než jsou dependency properties. V data triggeru se vytvoří data binding na nějakou vlastnost, a při změně této vlastnosti se mění hodnoty zadané v triggeru. Tímto způsobem lze svázat dva ovládací prvky. Pokud se hodnota jednoho prvku rovná zadané hodnotě, poté se změní vlastnosti druhého prvku. Data trigger mohou být definovány také jako MultiDataTrigger.

```
<StackPanel>
    <CheckBox Name="mujCheckBox" Content="Změnit text TextBlocku" />
    <TextBlock> <TextBlock.Style>
            <Style TargetType="TextBlock">
                <Setter Property="Text" Value="CheckBox není zaškrtnutý" />
                <Style.Triggers>
                    <DataTrigger Binding="{Binding ElementName=mujCheckBox,
                   Path=IsChecked}" Value="True">
                    <Setter Property="Text" Value="CheckBox je zaškrtnutý." />
                  </DataTrigger>
                </Style.Triggers>
            </Style>
    </TextBlock.Style> </TextBlock>
</StackPanel>
```

Obrázek 23: Příklad Data triggeru

3.5 Microsoft Visual Studio

Visual Studio je vývojové prostředí (IDE) od firmy Microsoft, které se využívá pro vytváření konzolových aplikací, aplikací s uživatelským grafických rozhraním, webových aplikacích, webových stránek a webových služeb. Programovací jazyky jsou podporovány na základě jazykových služeb. Vestavěnými jazyky jsou C#, VB.NET, C, C++. Jiné jazyky lze doinstalovat. Visual Studio podporuje přidávání rozšíření od vývojářů ve formě balíčků, maker a rozšíření.



Obrázek 24: Uživatelské rozhraní Visual Studia 2015

3.5.1 Editor kódu

Editor kódu ve Visual Studiu podporuje zvýraznění syntaxe a automatické dokončování kódu za použití *IntelliSense*, které našeptává kód z knihovny .NET Frameworku, případně ještě z jiných dostupných knihoven, a ze všech zdrojových kódů v projektu. Zmáčknutím klávesové zkratky *Ctrl* + *mezerník* se zapne našeptávání. IntelliSense dokončuje kód při psaní proměnné,

funkce, metody a také cyklů a dotazů. V editoru je zahrnuta podpora záložek, sbalování kódu nebo také *refaktorování*. Refaktorování je provádění změn v kódu, které nemají vliv na chování ostatního vnějšího kódu. Při refaktorování se zlepšuje kvalitu, čitelnost a čistota kódu. Na pozadí je spuštěna kompilace kódu pro upozorňování na syntaktické chyby.

3.5.2 **Designer**

„Jednou z věcí, která je integrována do Integrovaného vývojového prostředí, je možnost upravovat soubory graficky, někdy nazýváno grafické vývojové prostředí nebo designer. […] Ve zkratce je designer táhni-a-pusť komponenta Visual Studia.“ (Sempf, David a Sphar, 2010). Visual Studio podporuje různé designery, podle typu vytvářené aplikace. K dispozici je: WinForms designer, WPF designer, Web designer, Designer dat, Designer mapování, Designer tříd

WPF designer byl do Visual Studia přidán ve verzi 2008 a zvládá všechny funkce Windows Presentation Foundation. Grafické rozhraní lze vytvářet pomocí ručního psaní XAML kódu v samostatném okně nebo pomocí funkce *drag-and-drop*, která slouží pro volení ovládacích prvků z nabídky. Po „nakliknutí“ ovládacího prvku Visual Studio vyprodukuje XAML do okna s XAML kódem. Ovládací prvky jsou k dispozici v okně *Toolbox*. Po vybrání prvku je nabídnuto okno *Properties* (editor vlastností), ve kterém lze měnit grafický vzhled, vlastnosti a události ovládacího prvku.

3.5.3 **Debugger**

Debugger je softwarový nástroj pro hledání chyb při vývoji aplikace. Pro zjišťování chyb se využívají *breakpointy*, které zastaví běh programu v určitém místě. Je možné využít i *podmíněné breakpointy.* Ty se aktivují po splnění určité podmínky. Další možností sledování chyb je použití *watch*. Ve watch se zobrazují hodnoty proměnných během chodu aplikace. Dále se využívá *krokování*. Při krokování se kód po každém řádku zastaví a další kód se neprovede, dokud nezačneme krokovat dále. Ve Visual Studiu je během *debuggingu* možné, díky funkci *Edit and Continue*, kód upravovat.

3.5.4 **Další nástroje**

Visual Studio má k dispozici řadu dalších nástrojů, funkcí, doplňků a oken. Za zmínku stojí:

- **Průzkumník řešení (Solution Explorer) –** „Kdykoliv vytvoříte nebo otevřete aplikaci nebo samostatný soubor, Visual Studio použije koncept řešení k provázání všeho dohromady. Typicky je řešení vytvořené z jednoho nebo více projektů, kde každý projekt obsahuje položky s ním asociované“ (Johnson, 2014). V řešení jsou obsaženy všechny soubory, ze kterých se aplikace sestaví. Jsou zde třídy, okna, interface, databáze, reference a další soubory.
- **Document Outline** – Okno pro výběr ovládacích prvků. Prvky jsou seřazeny do stromové struktury.
- **Průzkumník objektů (Object Browser) –** Průzkumník jmenných prostorů, tříd, metod v projektu.

3.6 Databáze

Při vytváření aplikací je zapotřebí ukládat a načítat data. Jedním možným způsobem je využití databáze. „Velmi jednoduše řečeno, databáze je kolekce strukturovaných informací. Databáze jsou navrženy specificky ke správě velkého počtu informací a uchovávají data organizovaným a strukturovaným způsobem, který umožňuje uživatelům navrhnout a získat tyto data, když je potřebují“ (Agarwal, Huddleston a Raghuram, 2008). Pro práci s databázemi se využívá software nazývaný Systém řízení báze dat (SŘBD, databázový systém). Existuje celá řada databázových modelů: Hierarchická databáze, objektová databáze, relační databáze, objektově relační databáze a jiné.

Relační databáze se zakládá na relačním modelu. *Záznamy* jsou chápány jako řádky a sloupce vyjadřují *atributy*. Záznamy se nacházejí v tabulkách, které jsou mezi sebou propojeny pomocí *relací*. Pro jednoznačné odkázání na záznam se využívají *primární klíče*. „Primární klíč je atribut (sloupec) nebo kombinace atributů (sloupců), jejichž hodnoty jednoznačně identifikují záznam v entitě“ (Agarwal, Huddleston a Raghuram, 2008). Pomocí *cizích klíčů* se propojují jednotlivé tabulky mezi sebou. „Cizí klíč je atribut, který dokončuje vztah identifikováním rodiče entity. Cizí klíč poskytuje možnost zajištění integrity dat (nazvané referenční integrita) a navigaci mezi rozdílnými instancemi entity“ (Agarwal, Huddleston a Raghuram, 2008). Každý sloupec v tabulce má nějaký datový typ. Mezi základní typy patří: *Integer* (celé číslo), *Varchar* (krátký text), *Text* (dlouhý text), *Boolean* (logická hodnota), *Date* (datum). Relační databáze využívají k práci s daty SQL jazyk. Pomocí tohoto jazyku lze získávat data, vkládat nové záznamy, upravovat nebo mazat záznamy, vytvářet a mazat databáze, tabulky a provádět spoustu dalších operací.

3.6.1 **Lokální databáze – SQL Server Compact Edition (CE)**

Microsoft SQL Server Compact Edition je *embedded* (portable, přenosná) databáze, která je obsažená pouze v několika DLL souborech. Na disku je databáze uložena jako jeden SDF soubor, který může mít velikost až 4 GB. Po přiložení těchto souborů společně s aplikací bude databáze fungovat na jakémkoliv počítači, i když se zde nevyskytuje Microsoft SQL Server. Tato databáze oproti Microsoft SQL Serveru obsahuje méně funkcí, ale na využití do desktopových aplikací menšího rozsahu postačí. SQL Server CE oproti SQL Serveru neobsahuje: *pohledy*, *uložené procedury*, *triggery*, operaci *DISTINCT* v příkazu *SELECT*, *integritní omezení*. Pro práci s daty se využívá jazyk SQL.

3.6.2 **Externí databáze – MySQL**

MySQL databáze je jedním z nejoblíbenějších databázových systémů. Mezi výhody MySQL patří rychlost, malá velikost, jednoduchost, snadná implementace, multiplatformnost výkon a volná šiřitelnost databáze. Komunikaci zajišťuje jazyk SQL. Mezi podporované vlastnosti patří: *triggery*, *pohledy*, *uložené procedury*, práce s *metadaty*, *primární* a *cizí klíče*, *transakce*, *kurzory, časování události*, *poddotazy* a další. Databáze obsahuje klientskou (program pro práci s MySQL Serverem) a serverovou část (MySQL Server).

Architektura se skládá ze tří vrstev. První horní vrstva obsluhuje klient / server nástroje. Druhá vrstva zajišťuje funkcionalitu MySQL - funkce, optimalizace, analýza. Třetí vrstva uchovává úložné *enginy*. Enginy se starají o data, jejich ukládání, načítání a odpovídají na požadavky serveru.

Po připojení je klientovi přiděleno samostatné vlákno, kde se vykonávají dotazy. Vlákna jsou uchována v *cache paměti*, a proto se nemusí vytvářet a mazat pro každém nové připojení. Pro připojení k databázi je vyžadováno uživatelské jméno, heslo a jméno hostitele. MySQL poskytuje různé druhy databázových úložišť – *InnoDB*, *MyIsam*, *CSV* a další.

4 Vlastní práce

4.1 Návrh aplikace

Tato část práce, na rozdíl od přechozí části, ve které jsou uvedena veškerá teoretická východiska použitých nástrojů a technologií potřebných k vytvoření aplikace, se zaměřuje na návrh samotné databázové aplikace – Databáze parkourových spotů v Praze. Do návrhu aplikace patří požadavky, případy užití, funkcionalita aplikace, využité třídy, ovládací prvky a databázové systémy. Aplikace byla tvořena ve vývojovém prostředí Visual Studio 2015, které je díky svým možnostem a funkcím jedním z nejlepších vývojových prostředí pro tvorbu aplikací.

4.1.1 **Požadavky, funkcionalita aplikace a diagram případů užití**

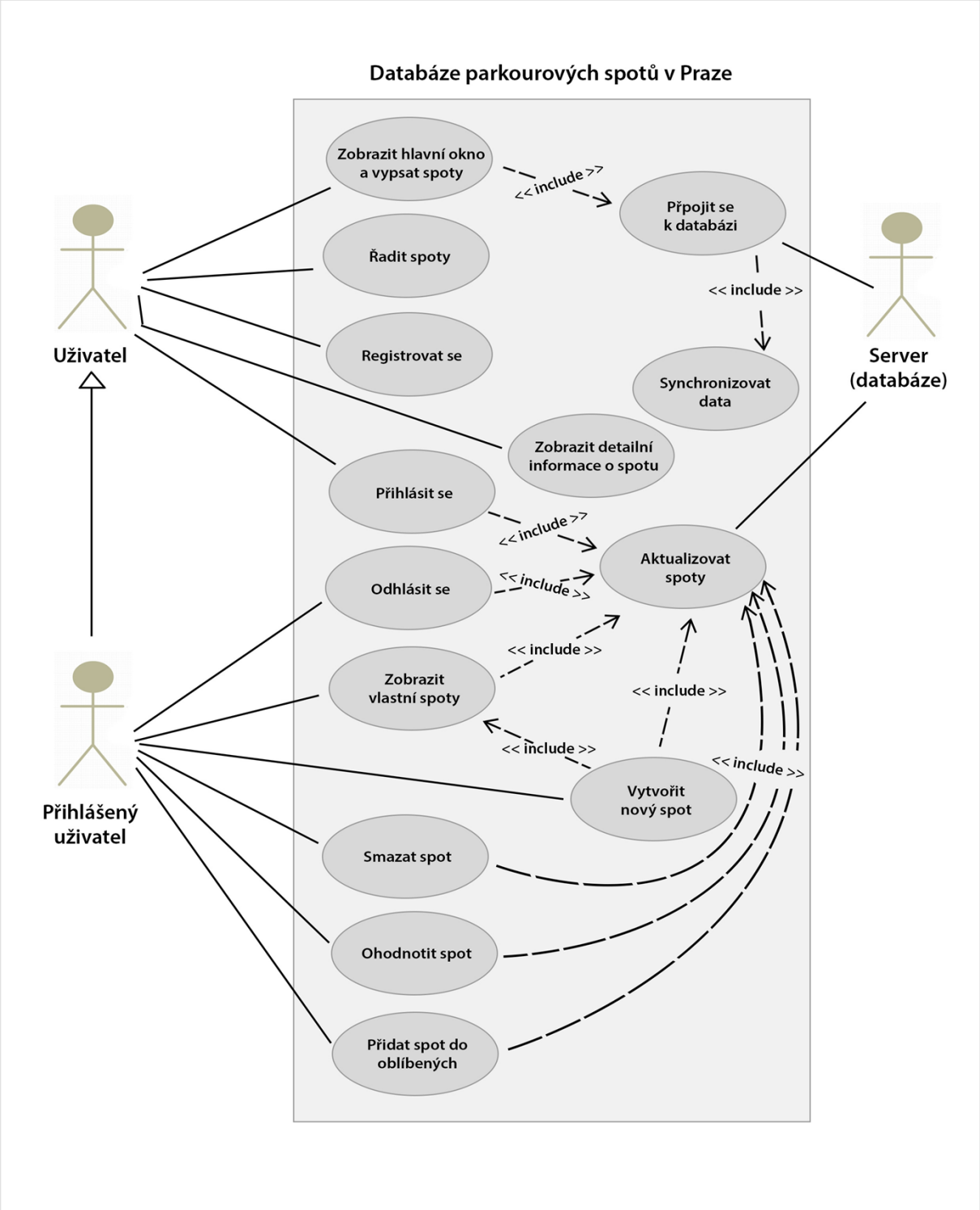
Aplikace je primárně zaměřena na parkourovou komunitu a na příznivce tohoto sportu. Jedním z velkých problémů parkouristů v České republice, obzvlášť v Praze, kde je parkour nejrozšířenější, je zvolení vhodného místa na trénink. Tato tréninková místa se nazývají spoty. Databáze parkourových spotů v Praze má za úkol usnadnit parkouristům orientaci v Praze, nalezení vhodného tréninkového místa vzhledem k jejich současné úrovni a schopnostem. Aplikace by měla poskytovat jednotné místo, odkud všichni její uživatelé mohou spravovat, zobrazovat a vzájemně vylepšovat povědomí o spotech a možnostech tréninku. Aplikace má následující požadavky:

- Připojení k databázovým systémům a vzájemná synchronizace dat.
- Uživatel má možnost využívat aplikaci jak s přístupem k internetu, tak i bez přístupu k internetu.
- Uživatel má možnost pomocí ovládacích prvků aplikace třídit, vyhledávat a procházet spoty.
- Uživatel může vytvářet vlastní spoty s vlastními informacemi a fotografiemi.
- Při splnění všech podmínek aplikace lze u každého spotu zobrazit galerii s fotografiemi.
- Aplikace obsahuje pochopitelné a jednoduché grafické rozhraní.

Při načítání se aplikace připojuje ke dvěma databázím. Nejdříve se snaží připojit k MySQL databázi, ke které se lze připojit pouze s funkčním internetovým připojením a poté se aplikace připojuje k SQL Server Compact Edition databázi, která nevyžaduje přístup k internetu. Aplikace je desktopovou verzí webové stránky dpsc.cz – Databáze parkourových spotů v Praze, s kterou spolupracuje. Pokud je internetové připojení k dispozici proběhne synchronizace dat v aplikaci, aby data odpovídala datům na uvedené webové stránce. Po načtení je zobrazeno úvodní okno aplikace s databází parkourových tréninkových míst. Uživatel má možnost procházet, vyhledávat a řadit dostupné spoty, zobrazit detailní informace o spotech a prohlédnout si galerii fotografií u jednotlivých spotů. Každý uživatel může využívat aplikaci jako registrovaný nebo neregistrovaný uživatel. Pro registraci nebo přihlášení slouží samostatné okno aplikace dostupné po kliknutí na příslušné tlačítko. Po registraci může uživatel využívat jak desktopovou aplikaci, tak webovou stránku. Přihlášený uživatel získává možnost hodnotit spoty, přidávat spoty do seznamu oblíbených spotů, přidávat vlastní soukromé spoty nebo je mazat. Spoty může filtrovat v závislosti na vlastní aktivitě, lze tedy zobrazit všechny uživatelem ohodnocené spoty, oblíbené spoty a své soukromé spoty. Aplikace při interakci s uživatelem kontroluje stav připojení k internetu a na tomto základě pracuje s příslušnou databází.

Aplikace využívá programovací jazyk C#, .NET Framework, Windows Presentation Foundation společně s XAML jazykem a databázové systémy MySQL a SQL Server Compact Edition.

Diagram případů užití (*use case diagram*) popisuje chování systému z pohledu uživatele a znázorňuje, co je od systému očekáváno. Avšak postup splnění těchto očekávání již není účelem diagramu případů užití. Případ užití se dá představit jako sada akcí. Po splnění těchto akcí je dosáhnuto cílů uživatele. Případy užití by měli pokrývat funkcionalitu navrženého systému. Pro jeden systém lze vytvořit více případů užití. Důležitou součástí případů užití je aktér (*actor*), který komunikuje s případy užití. V této databázové aplikaci je aktivním aktérem uživatel aplikace a roli pasivního aktéra má databázový server. Na obrázku č. 29 je zobrazen diagram případů užití pro aplikaci – Databáze parkourových spotů v Praze.



Obrázek 25:Diagram případů užití - Databáze parkourových spotů v Praze

4.1.2 **Použité třídy**

Aplikace obsahuje celkem 13 tříd.

Třídy pro grafické rozhraní – Window. Tyto třídy se starají o propojení grafického uživatelského rozhraní a logiky aplikace. Pomocí těchto tříd aplikace reaguje na události, volá ostatní třídy a při potřebě mění vzhled aplikace.

- **MainWindow** – Třída hlavního okna aplikace, která zajišťuje zobrazení hlavního okna, okna s detaily spotu a okno galerie.
- **SplashScreen** – Třída zobrazující načítání aplikace.
- **PrihlaseniRegistrace** – Třída zajišťující obsluhu okna pro přihlášení a registraci uživatele.
- **NovySpot** – Třída pro obsluhu okna pro vytvoření nového spotu.
- **CustomMessageBox** – Třída pro vytvoření vlastního dialogu pro upozornění uživatele.

Komunikaci mezi aplikací a serverem zajišťují dvě databázové třídy. První třída obsluhuje MySQL databázi, druhá třída obsluhuje SQL CE databázi.

- **MySqlDatabaze** – Třída sloužící pro komunikaci s relační online databází MySQL.
- **SqlCeDatabaze** – Třída sloužící pro komunikaci s relační offline databází SQL CE.

Pro uchovávání dat načtených z databáze se využívá pěti tříd, přičemž každá uchovává záznamy z jedné tabulky. Třídy uchovávající data:

- **Spot** – Třída uchovává data o všech spotech v databázi.
- **Obrazek** – Data o všech obrázcích patřícím ke spotům.
- **Hodnoceni** – Data o všech hodnocení spotů.
- **Oblibene** – Data se seznamy oblíbených spotů uživatelů.
- **Uzivatel** – Data o registrovaných uživatelích.

Poslední třídou aplikace je statická třída obsahující rozšíření.

- **Extensions** – Třída obsahuje rozšiřující a pomocné metody, které se využívají v aplikaci.

4.1.3 Použité ovládací prvky

Ovládací prvky neboli komponenty zajišťuje WPF Framework. Pro tvorbu aplikace jsou využity následující komponenty:

- **Window** – Okno aplikace.
- **Grid** – Komponenta pro vkládání obsahu do okna.
- **Border** – Slouží pro vykreslení pozadí nebo rámečku.
- **TabControl** – Rozděluje rozhraní na rozdílné oblasti.
- **StackPanel** – Kontejner pro další komponenty. Komponenty se ve StackPanelu řadí vertikálně nebo horizontálně za sebe
- **WrapPanel** – Kontejner lišící se od StackPanelu v zalamování elementů. Elementy ve WrapPanelu se při dosáhnutí okraje podle své orientaci zalamují buď do dalšího řádku, nebo sloupce.
- **DockPanel** – Kontejner, který umisťuje své vnitřní elementy na určitou pozici (vlevo, vpravo, nahoru, dolů nebo vyplní kontejner).
- **Menu** – Klasické menu aplikace.
- **TextBlock** – Prvek zobrazující text.
- **Label** – Prvek zobrazující text s některými odlišným vlastnostmi oproti TextBlocku.
- **TextBox** – Vstupní pole pro zápis textu.
- **ComboBox** – Výběrové pole obsahující položky.
- **PasswordBox** – Vstupní pole pro zadání hesla.
- **Button** – Klasické tlačítko, kdy se po kliknutí provede příslušná akce.
- **Image** – Prvek zobrazující obrázek.
- **DataGrid** – Zobrazuje data v tabulkovém formátu.
- **ScrollViewer** – Posuvník obsahu. Pokud je obsah větší než samotné okno, tak pro navigaci a posouvání obsahu slouží právě ScrollViewer.
- **ProgressBar** – Indikátor průběhu nějaké akce (v mé aplikaci pro zobrazení načítání obrázku).

- **GroupBox** – Viditelně sdružuje obsah do jednoho pole.
- A další

4.1.4 Databázové systémy, tabulky

Aplikace pro svou činnost potřebuje získávat uložená data. Tyto data jsou uložená pomocí databází. Jsou využívány 2 relační databáze. MySQL databáze je pojmenována dpsc a slouží jak pro webovou stránku *dpsc.cz*, tak i pro C# aplikaci. Databáze obsahuje celkem 20 tabulek, avšak aplikace využívá jen 5 tabulek, které jsou uvedené v tabulce níže. SQL CE databáze je pojmenována dpscdb a obsahuje celkem 5 tabulek. Názvy tabulek a jejich funkce jsou identické pro obě databáze, přičemž některé tabulky v SQL CE databázi jsou lehce upravené. Více o databázových systémech a tabulkách v kapitole Implementace databázových systémů. Tabulky v databázích:

Název tabulky	Popis (obsah)
spoty	Tréninková místa pro parkour.
obrazky	Fotografie spotů.
hodnoceni	Hodnocení spotů.
oblibene	Uživatelovi oblíbené spoty.
uzivatele	Registrovaní uživatelé.

Tabulka 1: Seznam databázových tabulek

4.2 Implementace databázových systémů

Aplikace využívá dva databázové systémy, jimiž jsou MySQL a SQL CE. Implementace zahrnuje vytvoření databází, tabulek, naplnění těchto tabulek daty, vytvoření tříd pro práci s databázemi a připojení se k těmto databázím.

4.2.1 Implementace externí databáze MySQL

Tato kapitola se zaměřuje na implementaci externí relační databáze MySQL. Zabývá se vytvořením databáze a připojením aplikace k databázi.

Vytvoření relační databáze MySQL

Pro vytvoření MySQL databáze je nutné získat přístup k MySQL serveru. Jelikož je aplikace spjata s webovou stránkou *http://www.dpsc.cz*, tak lze využít přístupu k MySQL databázi pomocí webhostingu. Webhosting znamená pronájem prostoru pro webové stránky na cizím serveru. Webhosting kromě prostoru pro webové stránky obsahuje i jiné funkce. Jednou z těchto funkcí je možnost vytvoření MySQL databáze. Webové stránky a také aplikace využívají databázi poskytovanou webhostingem *Endora*.

Naplnění MySQL databáze daty

Databáze musí obsahovat tabulky, která uchovají data. *Adminer* je nástroj pro správu databází, který je uložený jen v jednom PHP souboru. PHP soubor Admineru stačí vložit na webové stránky a přejít na jeho URL adresu v prohlížeči. Po zadání přihlašovacích údajů lze začít upravovat databázi.

Pro účely aplikace jsou zavedeny tabulky hodnoceni, oblibene, obrazky, spoty a uzivatele. Databáze obsahuje ještě další tabulky, které však slouží pro webovou stránku http://www.dpsc.cz. Struktura tabulek je uvedena na následujícím obrázku.

hodnoceni

Sloupec	Typ	Komentář
id	int(11) *Auto Increment*	
id_spotu	int(11)	
obrazek	varchar(100)	
nahled	varchar(100)	

oblibene

Sloupec	Typ	Komentář
id	int(11) *Auto Increment*	
id_spotu	int(11)	
id_uzivatele	int(11)	
zverejneni	datetime	

uzivatele

Sloupec	Typ	Komentář
id	int(11) *Auto Increment*	
url	text	
uzivatel	varchar(60)	
email	varchar(100)	
heslo	text	
obrazek	varchar(100)	
datum	datetime	
aktivace	text	
stav	bit(1)	

spoty

Sloupec	Typ	Komentář
id	int(11) *Auto Increment*	
url	text	
nazev	varchar(100)	
popis	longtext	
praha	varchar(30)	
adresa	varchar(60)	
metro	mediumtext	
bus	mediumtext	
tramvaj	mediumtext	
mapa	varchar(50)	
streetview	varchar(50)	
panorama	varchar(50)	
velikost	varchar(30)	
povrch	varchar(30)	
obtiznost	varchar(50)	
typ_treninku	varchar(30)	
zverejneni	datetime	

obrazky

Sloupec	Typ	Komentář
id	int(11) *Auto Increment*	
id_spotu	int(11)	
obrazek	varchar(100)	
nahled	varchar(100)	

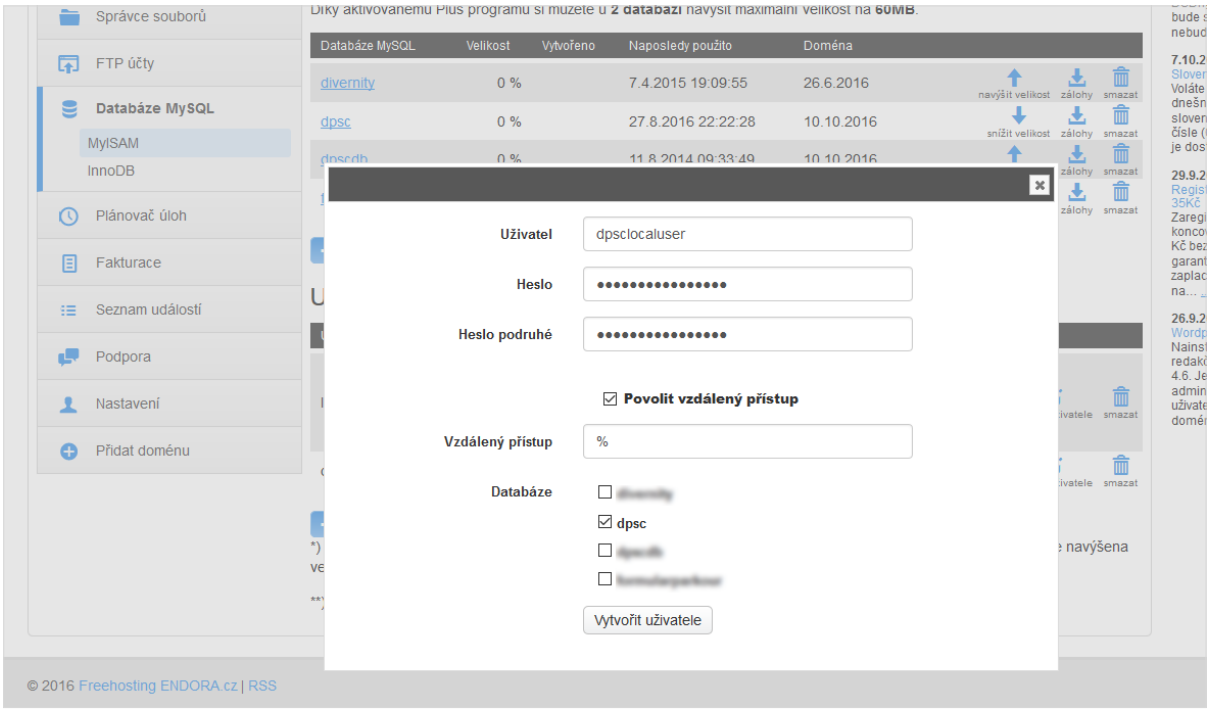
Obrázek 26: Struktura MySQL tabulek

Naplnění tabulek daty lze provést několika způsoby. Prvním způsobem je přidávání záznamů do tabulky pomocí rozhraní Admineru. Druhým způsobem je přidávání záznamů pomocí jazyka SQL a příkazu INSERT. Třetí možností je přidávat záznamy z formuláře webové administrace stránky http://www.dpsc.cz. Data této databáze byla přidána pomocí třetí možnosti. Je však možné využít libovolný typ přidávání záznamů do databáze.

Záznamy do tabulek spoty, obrazky se přidávají z formuláře webové administrace stránky. Do tabulky uzivatele přidávají záznamy návštěvníci webové stránky pomocí registračního formuláře nebo uživatelé aplikace. Záznamy do tabulek hodnoceni, oblibene přidávají přihlášení uživatelé pomocí ovládacích prvků u příslušných spotů na webové stránce nebo v aplikaci.

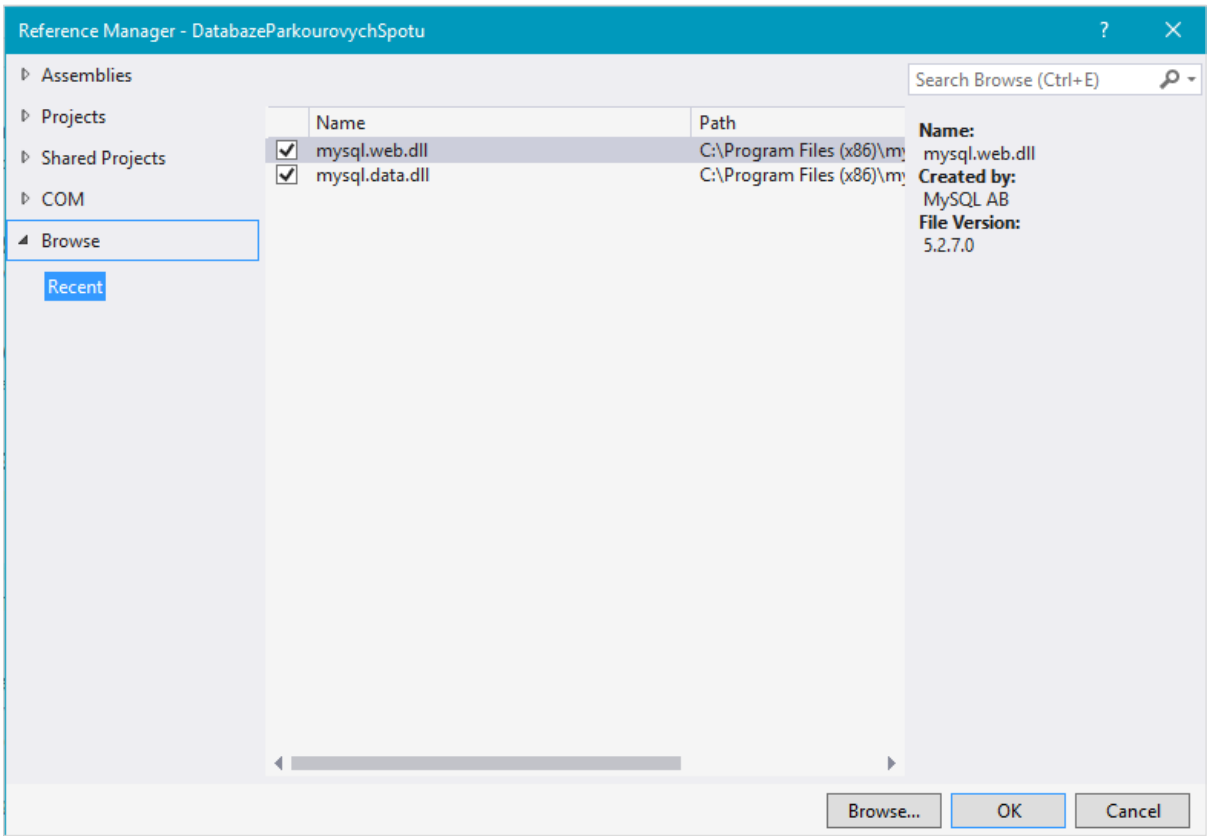
Připojení k databázi

Aplikace se připojuje k MySQL Serveru pomocí jednotných uživatelských údajů, jednoho uživatele. Tento uživatel má povolený vzdálený přístup k této databázi, jinak by nebylo možné aplikaci připojit k MySQL databázi.



Obrázek 27: Vytvoření uživatele MySQL databáze

Pro připojení aplikace k databázi využívá Visual Studio 2015 ovladač *Connector/Net,* který lze stáhnout na oficiálních stránkách MySQL. Tento ovladač pracuje s *Reference Managerem*, který obsahuje reference na .DLL MySQL soubory.



Obrázek 28: Přidání MySQL referencí do Visual Studia

Pro připojování aplikace k databázi slouží třída MySqlDatabaze. Následující výňatek ze třídy představuje proces připojení.

```
Třída MySQLDatabaze

        #region Proměnné, vlastnosti, konstanty
        private MySQLConnection _pripojeni;
        private const string Server = "sql.endora.cz";
        private const string Port = "3309";
        private const string Database = "dpsc";
        private const string Uid = "dpsclocaluser";
        private const string Pwd = "kW5A35X_d5DS71aA";
        #endregion

        #region Inicializace
        public MySQLDatabaze() : this(Server, Port, Database, Uid, Pwd) { }

        public MySQLDatabaze(string server, string port, string database,
        string uid, string pwd)
        {
            _pripojeni = VytvorPripojeni(server, port, database, uid, pwd);
        }

        private MySQLConnection VytvorPripojeni(string server, string port,
        string database, string uid, string pwd)
        {
            string pripojeniString = "Server=" + server + ";Port=" + port +
            ";Database=" + database + "; Uid=" + uid + ";Pwd=" + pwd +
            ";charset=utf8; convert zero datetime=True";
            return new MySQLConnection(pripojeniString);
        }
        #endregion
```

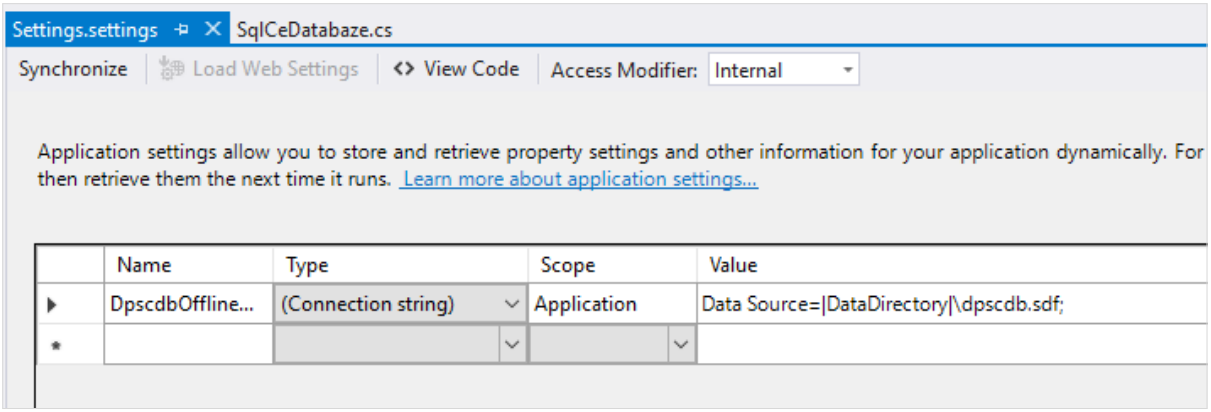
Obrázek 29: Databázová třída pro připojení aplikace k MySQL databázi

Třída MySqlDatabaze obsahuje privátní konstanty, ve kterých jsou uloženy informace potřebné k připojení a proměnná, ve které se uchovává připojení. V konstruktoru se předávají parametry pro připojení. Při volání prázdného konstruktoru se předají již zmíněné konstanty, jinak se využijí parametry předané v *konstruktoru*. Poté se volá metoda *VytvorPripojeni()* a

předává se do proměnné *_pripojeni*. V privátní metodě *VytvorPripojeni()* se vytvoří *řetězec pro připojení* a nové připojení pomocí závislé na tomto řetězci. Aplikace se připojí k MySQL databázi vytvořením instance této třídy.

4.2.2 Implementace lokální databáze SQL Server Compact Edition

SQL CE databáze zajišťuje funkčnost aplikace i bez internetového připojení. Jako první je nutné získat řetězec pro připojení (*connection string*), který obsahuje informace potřebné pro připojení k databázi. Řtězec pro připojení k databázi musí obsahovat jméno, typ řetězce, místo kde lze řetězec použít a hodnotu, která obsahuje samotnou aplikaci uloženou v místním souboru.

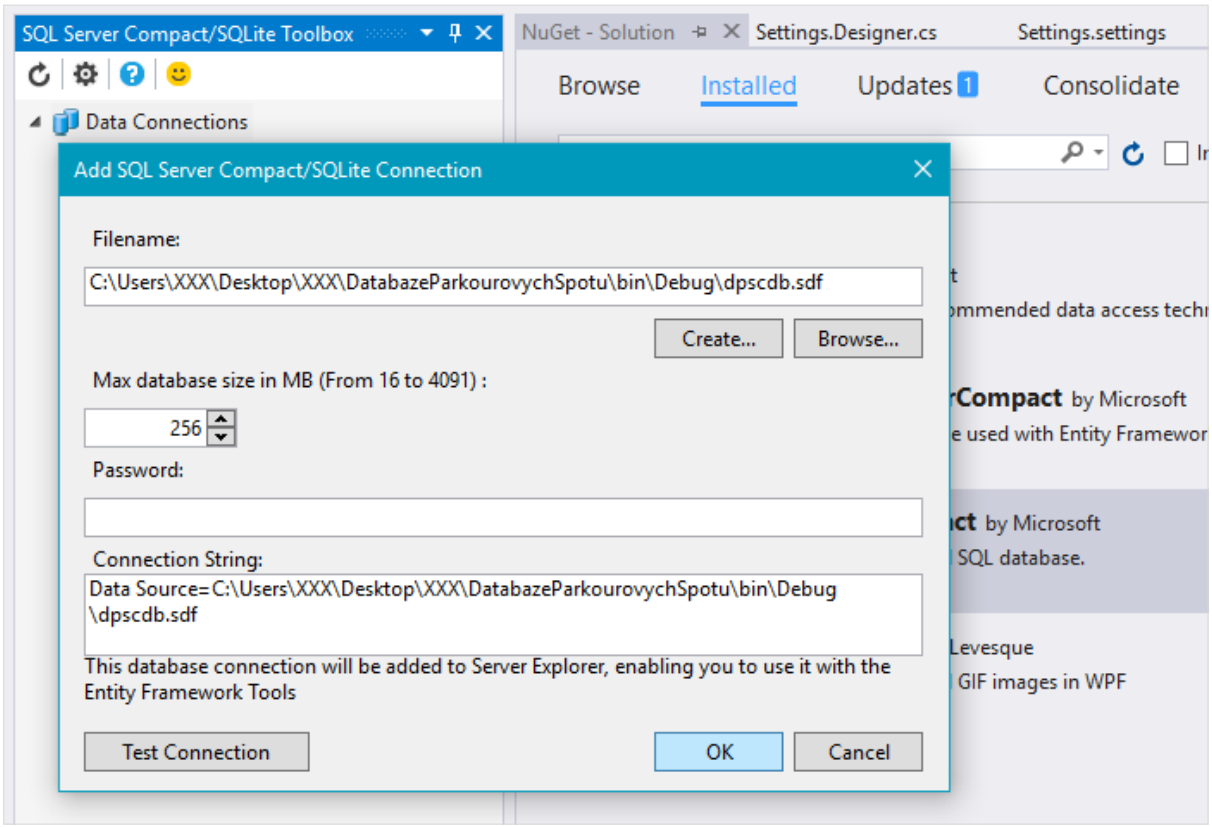


Obrázek 30: Connection string pro připojení k SQL CE databázi

Microsoft Visual Studio 2015 již nepodporuje lokální databázi SQL CE. Možnost tuto databázi využít i v této verzi zajišťuje modul *Manage NuGet Packages for Solution*.

Vytvoření relační databáze SQL CE

Pro vytvoření databáze je nutné mít stáhnutý a nainstalovaný *SQL Server Compact/SQLite Toolbox*, což je nástroj, pomocí kterého lze databázi vytvořit a editovat. Při zakládání databáze se musí zvolit místo pro uložení, název databáze, maximální velikost databáze, heslo (nepovinné) a připojovací řetězec. Po založení databáze se vytvořena prázdná SQL Server Compact 4.0 databáze s koncovkou .SDF. Tato databáze je přenosná, uložená v jednom souboru ve zdrojích aplikace.



Obrázek 31: Založení nové SQL CE databáze

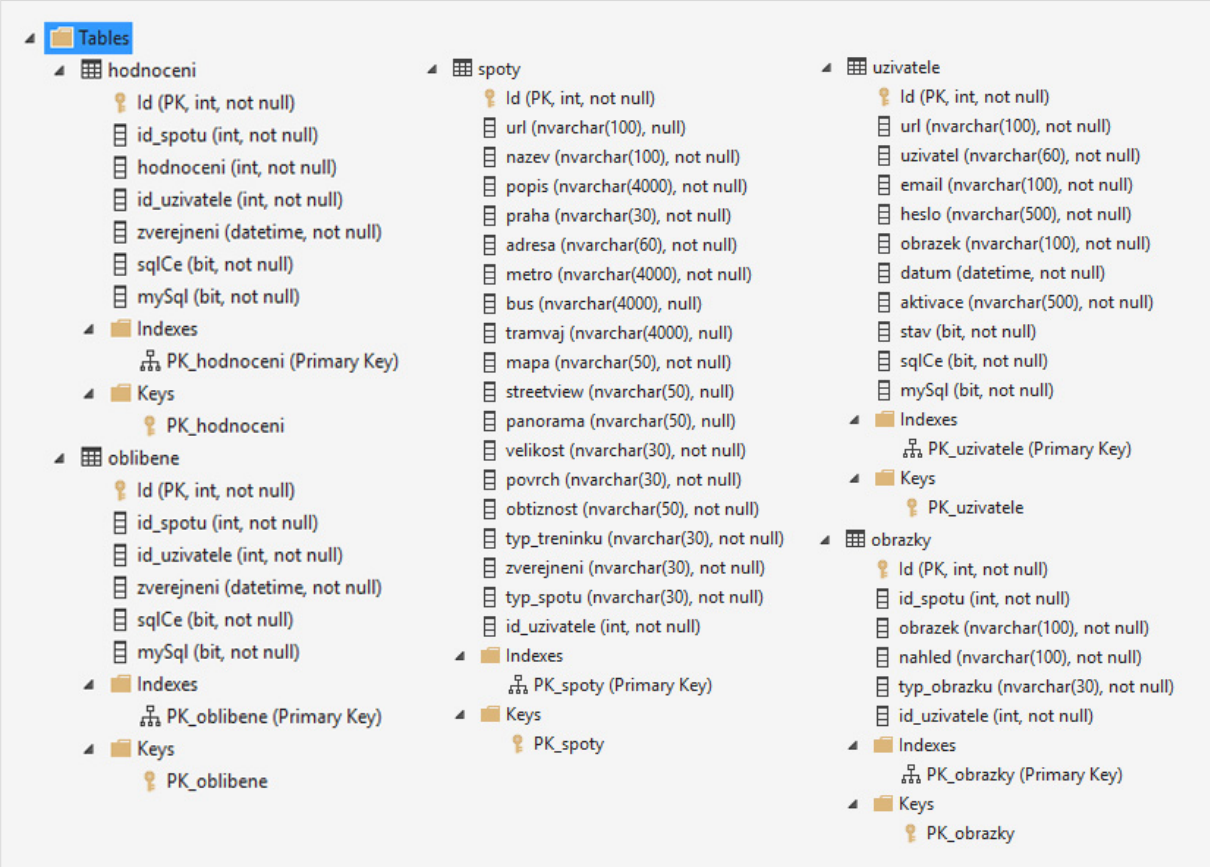
Naplnění SQL CE databáze daty

Založení tabulek lze provést dvěma způsoby. Prvním způsobem je „naklikání“ jednotlivých sloupců a jejich datových typů v editoru toolboxu. Druhý způsob je využít dotaz pro vytvoření databáze v jazyce SQL. V následujícím kódu je pro představu jazyka SQL uveden dotaz pro vytvoření tabulky uzivatele.

```
DROP TABLE [uzivatele];
GO
CREATE TABLE [uzivatele] (
  [Id] int NOT NULL
, [url] nvarchar(100) NOT NULL
, [uzivatel] nvarchar(60) NOT NULL
, [email] nvarchar(100) NOT NULL
, [heslo] nvarchar(500) NOT NULL
, [obrazek] nvarchar(100) NOT NULL
, [datum] datetime NOT NULL
, [aktivace] nvarchar(500) NOT NULL
, [stav] bit NOT NULL
, [sqlCe] bit NOT NULL
, [mySql] bit NOT NULL
);
GO
ALTER TABLE [uzivatele] ADD CONSTRAINT [PK_uzivatele] PRIMARY KEY ([Id]);
GO
```

Obrázek 32: Vytvoření SQL CE tabulky pomocí jazyka SQL

V SQL CE databázi je nutné vytvořit tabulky hodnoceni, oblibene, obrazky, spoty a uzivatele. Struktura tabulek jsou následující:



Obrázek 33: Struktura SQL CE tabulek

Záznamy lze vytvořit různými technikami. Jednou možností naplnění je vytvořit záznamy ručně pomocí příkazů INSERT, což je v případě této databáze zdlouhavé, jelikož podporuje pouze vkládání jednoho záznamu najednou. Záznamy musí odpovídat záznamům v MySQL databázi, která je již naplněná, a proto aplikace disponuje možností synchronizace záznamů. Pokud jedna databáze obsahuje záznamy navíc oproti databázi druhé, zavolá se metoda *SynchronizaceDat()* obsažena v obou databázových třídách aplikace, která tuto synchronizaci provede. Kontrola synchronizace dat je volána po každém spuštění aplikace, pokud je k dispozici internetové připojení.

Připojení k databázi

Připojení aplikace k SQL CE zajišťuje třída SqlCeDatabaze. Následující výňatek ze třídy představuje proces připojení.

```
Třída SqlCeDatabaze

        #region Proměnné, vlastnosti
        private SqlCeConnection _pripojeni;
        private string _pripojeniString =
        Properties.Settings.Default.DpscdbOfflineConnectionString;
        #endregion

        #region Inicializace
        public SqlCeDatabaze()
        {
            _pripojeni = VytvorPripojeni();
        }

        private SqlCeConnection VytvorPripojeni()
        {
            return new SqlCeConnection(_pripojeniString);
        }
        #endregion
```

Obrázek 34: Databázová třída pro připojení aplikace k SQL CE databázi

Třída SqlCeDatabaze nejdříve vytvoří potřebné proměnné, ve které se uchovává připojení a proměnná s řetězcem pro připojení k databázi. V konstruktoru se volá metoda *VytvorPripojeni()* a předává se do proměnné *_pripojeni*. V privátní metodě *VytvorPripojeni()* se předává řetězec pro připojení a vytvoří se nové připojení pomocí tohoto řetězce. Vytvořením instance této třídy, například ve třídě okna aplikace, zajistí připojení aplikace k SQL CE databázi.

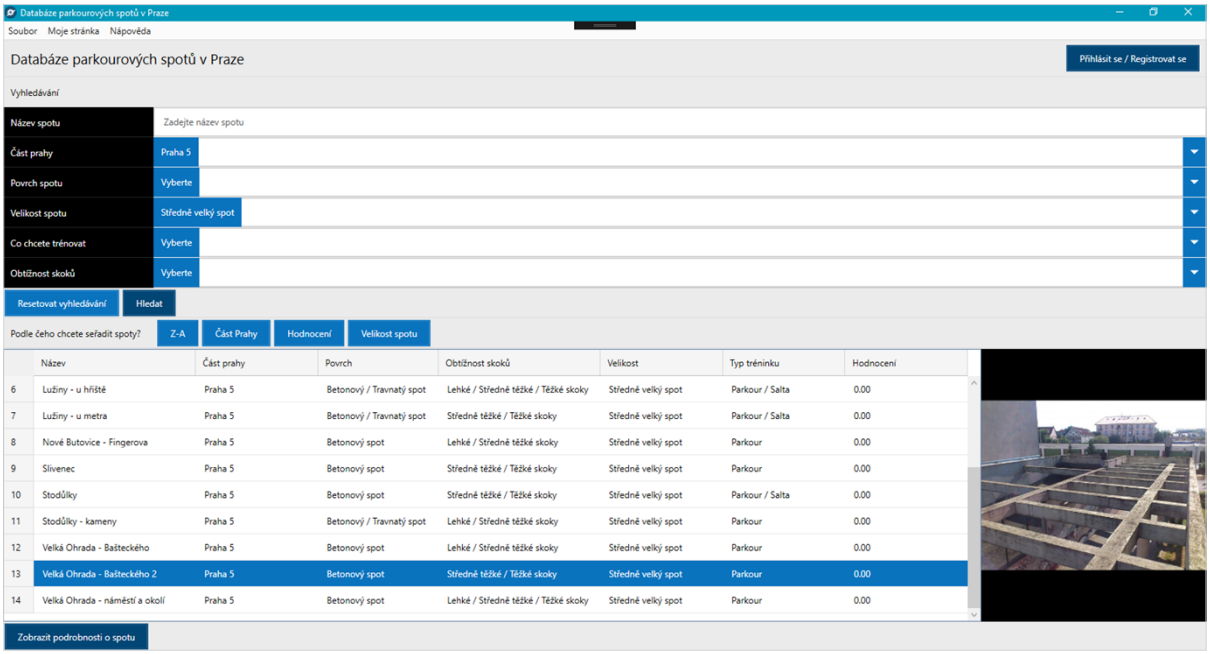
4.3 Implementace aplikace

Při vytváření aplikace je nutná implementace všech jejích funkcí. V první části této kapitoly je uvedeno vše spojené s grafickým uživatelským rozhraním aplikace a jsou popsána jednotlivá okna, ze kterých se aplikace skládá, pro lepší přehled v příští části kapitoly. Poté je vysvětlena funkčnost a implementaci aplikace pomocí tříd, jejich metod a událostí.

4.3.1 Uživatelské rozhraní

Uživatelské rozhraní se skládá z oken (window). V každém okně jsou umístěny pro uživatele ovládací prvky, kterými se aplikace ovládá. Těchto oken obsahuje aplikace celkem pět – MainWindow, PrihlaseniRegistrace, NovySpot, SplashScreen a CustomMessageBox. Veškerá okna jsou vytvořena pomocí XAML jazyka. Styl a šablony pro ovládací prvky jsou definovány globálně pro celou aplikaci v souboru App.xaml.

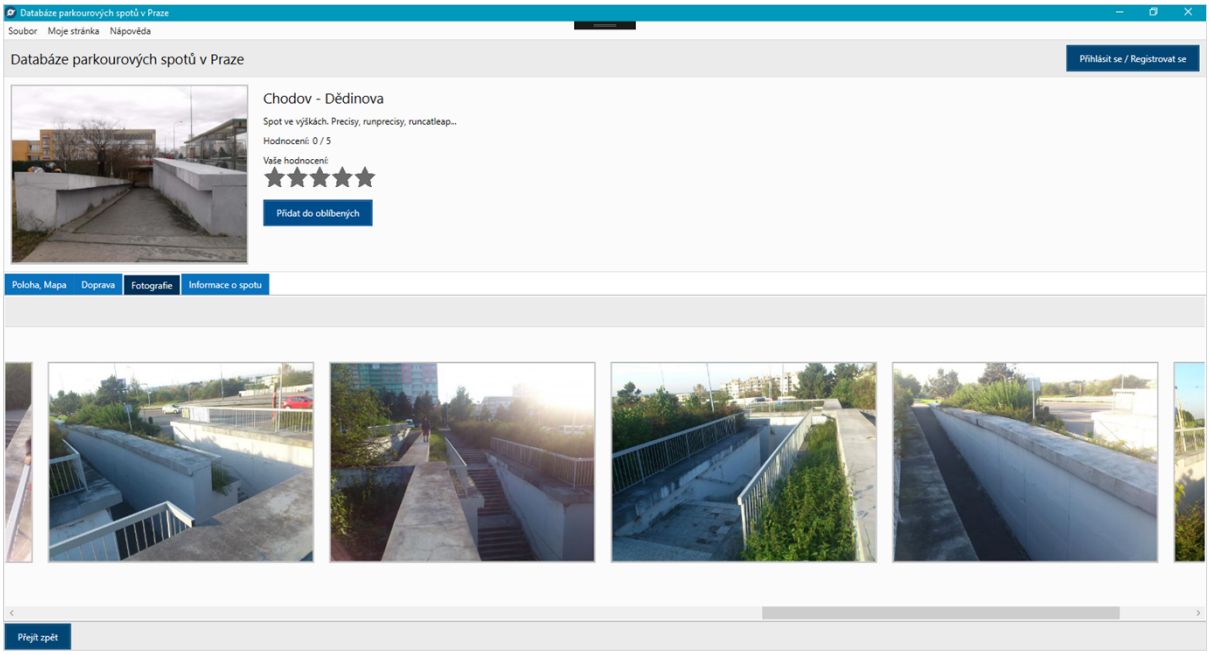
MainWindow



Obrázek 35: Hlavní okno aplikace

Jedná se o nejdůležitější okno aplikace. Toto okno se zobrazí po načtení aplikace. V horní část okna je menu aplikace. Menu aplikace je fixní a nachází se ve všech oknech. Nachází se zde možnost ovládání aplikace včetně nápovědy k aplikaci a informací o aplikaci. Dalšími fixními prvky jsou nadpis a tlačítko pro přihlášení / registraci či odhlášení uživatele. Okno

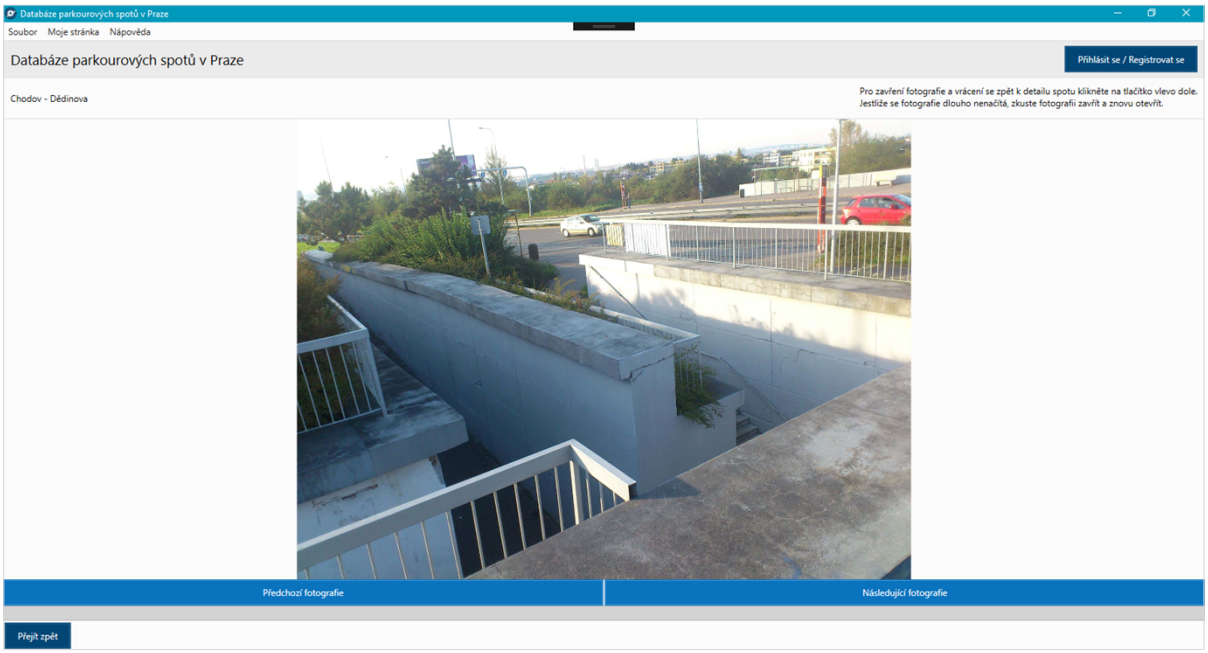
(window) je dále rozděleno pomocí elementu TabControl na tři podokna – databáze spotů, detail spotu a fotogalerie spotů. V prvním podokně, databáze spotů, jsou zobrazeny v tabulce (DataGrid) pražské spoty. Pomocí tlačítek (Button), textových polí (TextBox) a výběrových polí (ComboBox) je zajištěna možnost pracovat s aplikací – vyhledávat, řadit spoty a přepínat se na další okna aplikace. V tomto podokně se zobrazují základní informace o každém spotu včetně náhledu spotu.



Obrázek 36: Okno detailu spotu

Podokno detailu spotu v horní části obsahuje úvodní fotografii spotu, krátký popis spotu a možnost ohodnotit spot nebo přidat spot do seznamu oblíbených spotů. Dolní část je rozdělena do čtyř záložek pomocí elementu TabControl. Jsou jimi:

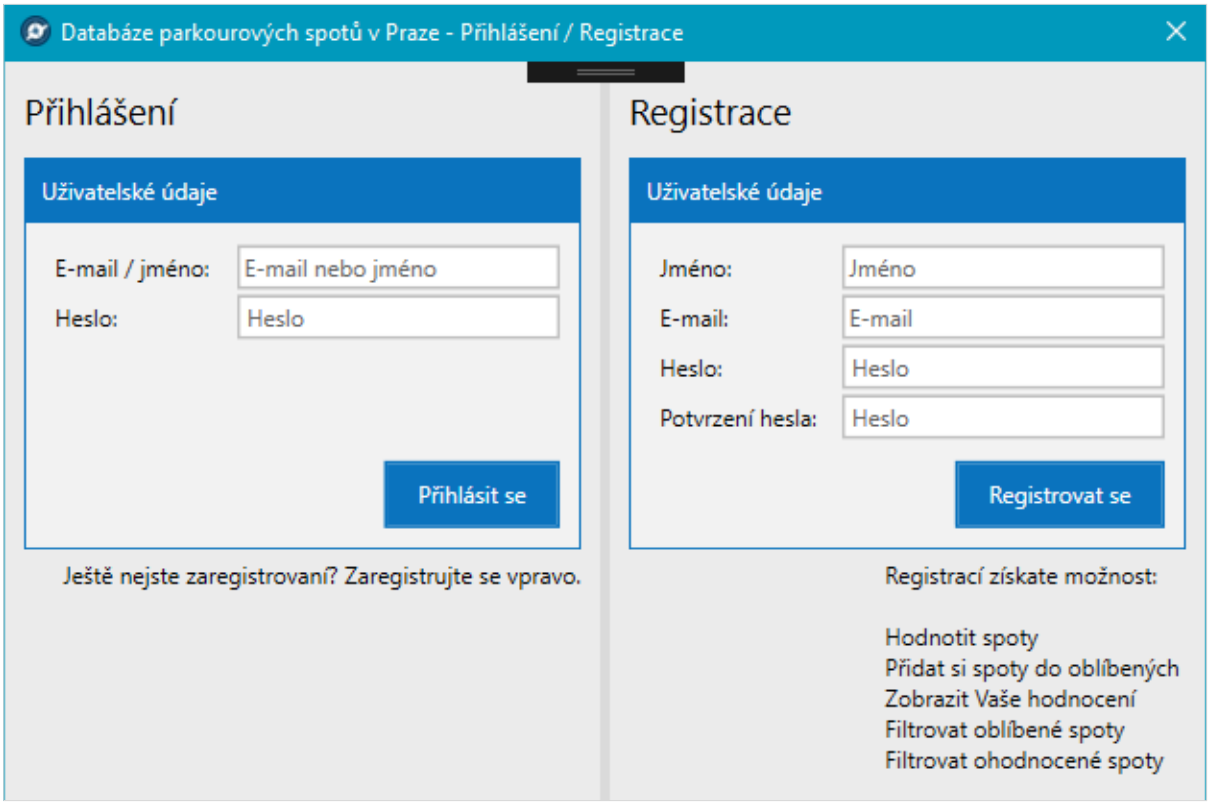
- **Poloha, Mapa** – Zde je uvedena poloha spotu definována adresou a částí prahy, poté jsou zde odkazy pro zobrazení spotu na mapách.
- **Doprava** – Obsahuje možnosti a čas dopravy ke spotu za využití hromadné dopravy (metro, tramvaje a autobusy).
- **Fotografie** – Náhledy fotografií spotu. Po kliknutí na náhled se fotografie zobrazí v záložce galerie, pokud je dostupné internetové připojení.
- **Informace o spotu** – Doplňující informace o spotu, jimiž jsou velikost spotu, obtížnost skoků, typ tréninku a povrch spotu.



Obrázek 37: Okno galerie spotu

Galerie je jednoduchá a intuitivní pro ovládání. Vlevo nahoře je k dispozici název spotu, na pravé horní straně jsou doporučené informace o ovládání galerie. Ve středu okna se nachází samotná fotografie, která se přizpůsobí velikosti okna. Pod fotografií se nachází tlačítka pro přechod mezi fotografiemi, pod těmito tlačítky je ukazatel průběhu načítání fotografie. Nakonec v levém dolním rohu se nachází tlačítko pro zavření fotografie a přechod zpět na detail spotu.

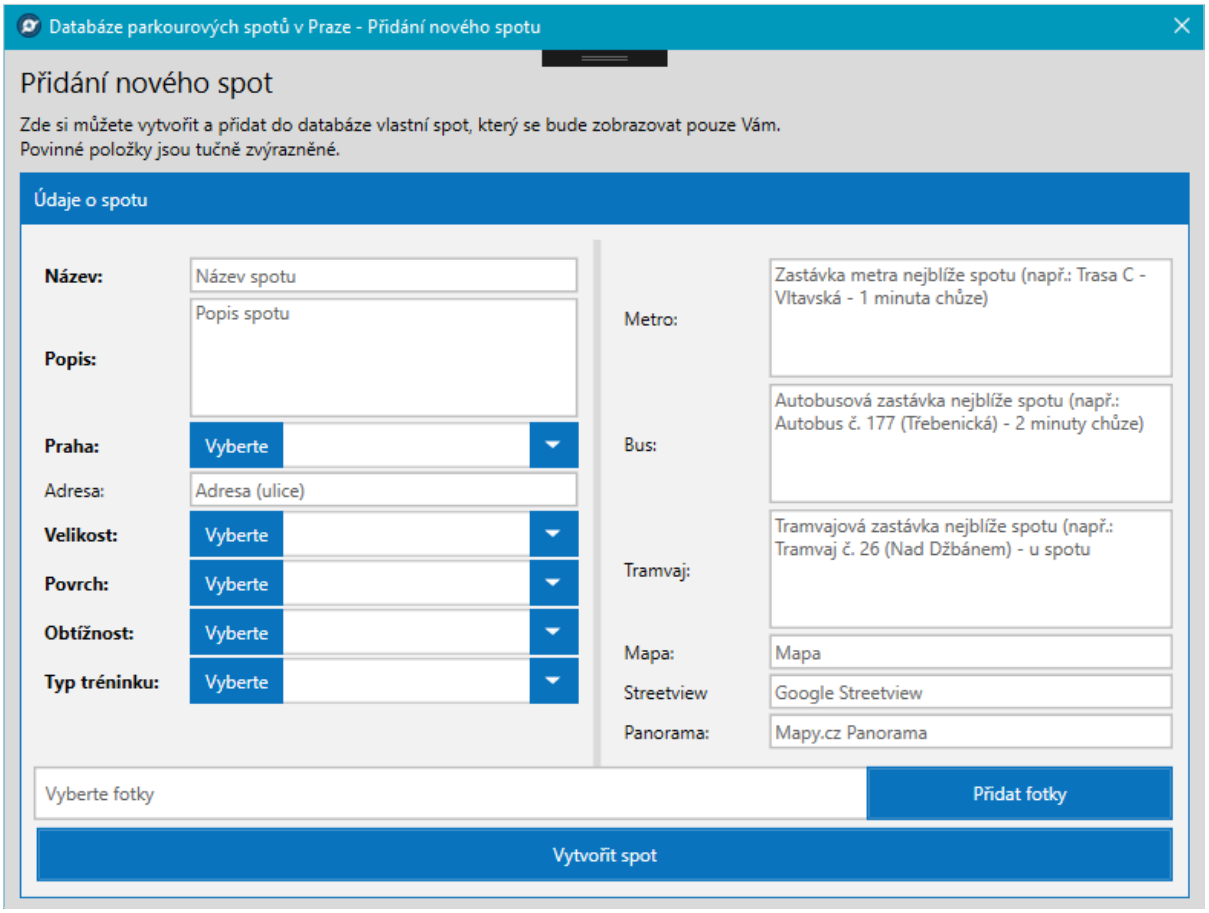
PrihlaseniRegistrace



Obrázek 38: Okno pro přihlášení a registraci uživatele

Okno PrihlaseniRegistrace obsahuje dva formuláře. Jeden pro přihlášení a druhý pro registraci. Na levé straně se nachází možnost přihlášení se do aplikace pomocí již existujícího účtu. K přihlášení stačí zadat E-mail nebo jméno účtu a přináležející heslo. Po vyplnění lze formulář odeslat tlačítkem *Přihlásit se*. Na pravé straně je k dispozici formulář pro zaregistrování nového účtu. Po zadání uživatelských údajů se lze registrovat za pomoci tlačítka *Registrovat se*. V dolní části každého formuláře jsou uvedeny doplňující údaje.

NovySpot



Obrázek 39: Okno pro vytvoření nového spotu

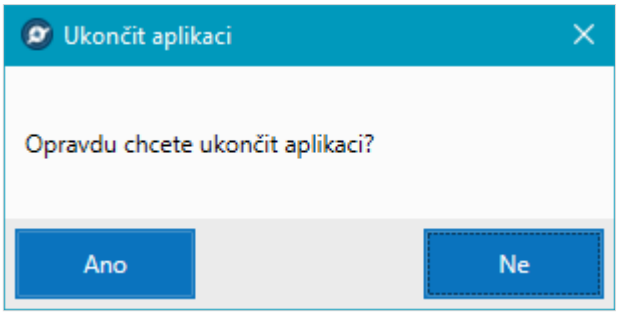
Přidání nového vlastního spotu má na starosti okno s názvem NovySpot. V horní části jsou textové informace, poté následuje samotný formulář. Formulář obsahuje poměrně hodně údajů k vyplnění, a to z důvodu souladu spotů vytvořených administrátorem a spotů definovaných uživateli aplikace. Pomocí popisů každého prvku a možnosti prohlížení existujících spotů lze nový spot vytvořit celkem snadno. K vlastnímu spotu lze přidávat taktéž fotografie.

SplashScreen

Obrázek 40: Okno načítání aplikace

Toto okno nedovoluje uživateli provádět žádnou akci, slouží pouze jako informativní okno. Okno je složeno z obrázku na pozadí a textových položek (TextBlock). Okno se spouští při zapnutí aplikace, pomocí informativních textů zobrazuje momentální akci, kterou provádí aplikace před samotným zavedením aplikace. Po skončení načítání se zobrazí okno MainWindow.

CustomMessageBox



Obrázek 41: Okno pro zobrazení zprávy

CustomMessageBox obsahuje hlavičku okna, text upozornění pro uživatele a tlačítka pro obsluhu formuláře. Obsah těchto ovládacích prvků se mění v závislosti na provedené akci uživatele. Například při pokusu zavřít aplikaci okno zobrazí text s dotazem o zavření aplikace

a tlačítka Ano, Ne. Okno může obsahovat tlačítka Ano a Ne nebo tlačítko OK. CustomMessageBox je vlastní okno pro zobrazení informativních zpráv, kterými aplikace komunikuje s uživatelem. Okno se spouští při akcích jako je zavření aplikace, upozornění uživatele na chyby nebo pro zobrazení nápovědy.

4.3.2 Funkcionalita tříd a událostí

Základní popis jednotlivých tříd je popsán v kapitole 4.1.2 Použité třídy.

Inicializace aplikace

Po spuštění aplikace jsou postupně využity tyto třídy: MainWindow, SplashScreen, MySqlDatabaze, SqlCeDatabaze. Databázové třídy načítají data s pomocí tříd uchovávajících data - Hodnoceni, Oblibene, Obrazek, Spot a Uzivatel. Pro vypsání zpráv uživateli je využívána třída CustomMessageBox.

Inicializaci aplikace zahajuje třída MainWindow, která je nejrozsáhlejší třídou v celé aplikaci a kromě inicializace řeší události hlavního okna a obsahuje proměnné, vlastnosti a pomocné metody pro události. Zavedení aplikace probíhá v konstruktoru třídy. Vytvoří se instance třídy SplashScreen pro zobrazení úvodního načítání aplikace. Poté se pomocí metody *PripojeniKDatabazi()* aplikace pokusí připojit k dříve zmíněným databázím za využití databázových tříd SqlCeDatabaze a MySqlDatabaze. Postup připojení aplikace k databázím je popsán v kapitole 4.2 Implementace databázových systémů. Získaná data z databází se uloží do privátní proměnné *_dataSet*. Tabulka s daty v proměnné _dataSet se nastaví jako *DataContext* pro element DataGrid v hlavním okně aplikace, čímž se vypíše seznam spotů z databáze do okna aplikace. Pokud dojde k chybám na základě nedostupnosti databází, k čemuž může dojít z důvodu nedostupného internetového připojení, aplikace uživateli tento problém zobrazí pomocí třídy CustomMessageBox. Tímto je aplikace inicializována a čeká na vstup od uživatele.

Vyhledávání spotů

Na základě hodnot zadaných v příslušných ovládacích prvcích se pomocí události zavolané kliknutím na tlačítko *Hledat* spustí metoda *NaplneniDataGriduVyhledavani()*. V této metodě se pracuje s databázovými třídami, kterým se předají hodnoty z ovládacích prvků.

Pomocí SQL příkazu SELECT se vyberou příslušné spoty z databází. Tyto data se zobrazí v DataGridu stejně jak tomu bylo při inicializaci aplikace.

Řazení spotů

Funkcionalita řazení je odlišná oproti vyhledávání spotů. K tomuto úkolu již nejsou využívány databázové třídy. Řadit spoty lze abecedně, podle části prahy, podle velikosti spotů a na základě ohodnocení spotů. Abecední řazení a řazení hodnocení je zajištěno pomocí zavolání předdefinované vlastnosti *sort* na objektu *DataView*, který je propojený s tabulkou dat *DataTable*. Tato vlastnost seřadí data podle příslušného sloupce v DataGridu. Vlastnost *sort* lze využít pro jednoduché řazení dat, avšak pro složitější řazení, jako je řazení podle části prahy a podle velikosti spotů, již tuto vlastnost využít nelze. Při řazení spotů podle velikosti se prochází cyklem *foreach* objekt DataView a podle velikosti spotu se daný spot importuje do nového objektu DataTable. Tabulka s daty se nastaví jako DataContext elementu *DataGrid*. Řazení spotů podle části prahy je zajištěno filtrováním načtených spotů v DataGridu pomocí dotazovacího jazyku LINQ.

Podrobnosti o spotu

Zobrazení podrobností o spotu se provede kliknutím na tlačítko *Zobrazit podrobnosti o spotu* v dolní části okna aplikace s evidencí spotů, dvojklikem na příslušný spot nebo stisknutím klávesy Enter při označení spotu. Tím se zavolá událost *podrobnostiOSpotuButton_Click()*. V této události se nejdříve zkontroluje internetové připojení, a zda je vybraný spot, o kterém chce uživatel získat informace. Následuje přepnutí ovládacího prvku *TabItem*, čímž se zobrazí požadované okno aplikace. Další příkazy zajišťují nastavení příslušných informací o vybraném spotu do ovládacích prvků a vybrání náhledů fotografií ke spotu.

Galerie fotografií

Ke galerii obrázků se přistupuje ze záložky *Fotografie* na straně detailu spotu. Tyto fotografie jsou načteny již při zobrazení podrobností o spotu. Pokud je uživatel připojený k internetu je možné fotografii zvětšit kliknutím na náhled fotografie. To zajišťuje událost MouseUp volaná na příslušném obrázku. Aplikace se přepne na okno galerie spotu a získá se zdroj fotografie, který je umístěný na webové stránce http://www.dpsc.cz. Galerie fotografií

umožňuje přepínat mezi jednotlivými fotografiemi spotu za pomocí navigačních šipek, tvořených ovládacími prvky Button.

Registrace a přihlášení uživatele

Přístup k registraci a přihlášení uživatele je zajištěn ze třídy MainWindow, která volá třídu PrihlaseniRegistrace.

Po kliknutí na tlačítko ‚Přihlásit se / Registrovat se' v hlavním okně se spustí událost *nazevAplikaceButton_Click()*, pomocí které se vytvoří instance třídy PrihlaseniRegistrace, inicializuje se tato třída a tím se zobrazí okno pro registraci a přihlášení. V této třídě jsou dvě události, jedna se stará o přihlášení uživatele a druhá se stará o registraci uživatele. Událost zajišťující přihlášení se nazývá *prihlaseniButton_Click()* a kontroluje uživatelem zadaná data a pomocí databázových tříd se pokouší uživatele přihlásit. Druhou událostí je *registraceButton_Click()*, jenž také kontroluje vstupy uživatele, poté porovnává údaje v databázi s údaji zadanými uživatelem a v případě úspěchu načítá uživatele, registruje ho a zavírá okno s registrací, čímž se uživatel dostane zpět do hlavního okna aplikace.

Vytvoření nového spotu

Okno pro vytvoření nového spotu je spouštěno taktéž třídou *MainWindow* z hlavního okna, a to pomocí tlačítka *Vytvořit nový vlastní spot*, které volá po kliknutí událost *vytvoreniSpotuButton_Click()*.

V tělu události se pouze inicializuje třída NovySpot a otevře se příslušné okno. Hlavní funkci v této třídě zajišťuje událost *vytvoreniSpotuButton_Click()*. Událost zajišťuje kontrolu a validaci formuláře pro vytvoření spotu a prostřednictvím databázových tříd vytváří spot společně s obrázky náležejícími k vytvářenému spotu.

Metody a události tříd

V tabulkách číslo 2 až 8 jsou uvedeny nejdůležitější metody a události daných tříd. Třídy oken obsahují události volané uživateli a také pomocné metody, které komunikují s jinými třídami nebo mění uživatelské grafické rozhraní. Databázové třídy obsahují metody, které zajišťují logiku při práci s databázemi. Poslední třídou, která obsahuje metody je třída Extensions. Třída obsahuje různorodé pomocné metody, které se využívají v celé aplikaci.

MainWindow

Metoda nebo Událost	Popis
PripojeniKDatabazi()	Připojuje aplikaci k databázím.
nazevAplikaceButton_Click()	Otevírá okno pro registraci a přihlášení nebo odhlašuje uživatele.
vytvoreniSpotuButton_Click()	Otevírá okno pro vytvoření spotu.
podrobnostiOSpotuButton_Click()	Přepíná se do okna s podrobnostmi o spotu.
VyhledavaniButton_Click()	Pomocí databázových tříd vyhledává spoty dle zadaných kritérií.
ZmenaPripojeniInternetu()	Kontroluje změnu internetového připojení pro zajištění funkčnosti aplikace.
NaplneniDataGridu()	Naplňuje DataGrid daty.
NacteniZaznamu()	Načítá záznamy z databáze.
Image_SourceChanged()	Mění zdroj a načítá obrázek do příslušného prvku aplikace.

Tabulka 2: Nejdůležitější metody a události třídy MainWindow

PrihlaseniRegistrace

Metoda / Událost	Popis
registraceButton_Click()	Pokouší se registrovat uživatele.
SqlCeDatabazeUzivateleRegistrace()	Validuje uživatelské jméno a email uživatele při registraci a porovnává je vůči SQL CE databázi.
MySqlDatabazeUzivateleRegistrace()	Validuje uživatelské jméno a email uživatele při registraci a porovnává je vůči MySQL databázi.
prihlaseniButton_Click()	Pokouší se přihlásit uživatele.

Tabulka 3: Nejdůležitější metody a události třídy PrihlaseniRegistrace

NovySpot

Metoda / Událost	Popis
vytvoreniSpotuButton_Click()	Pokouší se vytvořit uživatelův spot.
pridatFotkyButton_Click()	Umožňuje vybrat fotografie spotu.

Tabulka 4: Nejdůležitější metody a události třídy NovySpot

SplashScreen

Metoda / Událost	Popis
Timer_Tick()	Zobrazuje animaci načítání aplikace a hlídá načtení aplikace.
Window_MouseDown()	Umožňuje uživateli posouvání okna načítání aplikace.

Tabulka 5: Nejdůležitější metody a události třídy SplashScreen

CustomMessageBox

Metoda / Událost	Popis
Show()	Zobrazuje okno MessageBoxu.
anoButton_Click()	Nastavuje chování okna po stisknutí tlačítka ‚Ano‘.
neButton_Click()	Nastavuje chování okna po stisknutí tlačítka ‚Ne‘.
OKButton_Click()	Nastavuje chování okna po stisknutí tlačítka ‚OK‘.
Window_Loaded()	Nastavuje vzhled MessageBoxu.

Tabulka 6: Nejdůležitější metody a události třídy CustomMessageBox

MySqlDatabaze

Metoda / Událost	Popis
VytvorPripojeni()	Vytváří a vrací připojení k MySQL databázi.
NacteniListu()	Načítá data z databáze do jednotlivých listů.
SynchronizaceDat()	Synchronizuje data v databázi s lokální databází.
VybraniVysledku()	Vrací SELECT pro získání dat z databáze.
NacteniZaznamu()	Načítá záznamy a vrací je v podobě DataSetu.

Tabulka 7: Nejdůležitější metody a události třídy MySqlDatabaze

SqlCeDatabaze

Metoda / Událost	Popis
VytvorPripojeni()	Vytváří a vrací připojení k SQL CE databázi.
NacteniListu()	Načítá data z databáze do jednotlivých listů.
SynchronizaceDat()	Synchronizuje data v databázi s externí databází.
VytvoreniNahledu()	Vytváří náhledy pro spoty uživatele.
SmazaniPrazdnychSlozek()	Spravuje složky vytvořené programem pro uchovávání obrázků.
VybraniVysledku()	Vrací SELECT pro získání dat z databáze.
NacteniZaznamu()	Načítá záznamy a vrací je v podobě DataSetu.
ZiskaniId()	Získá ID záznamu z příslušné tabulky.

Tabulka 8: Nejdůležitější metody a události třídy SqlCeDatabaze

4.4 Publikace aplikace

Pro zpřístupnění aplikace uživatelům je nutné vybrat vhodný typ a místo zpřístupnění aplikace.

4.4.1 Zpřístupnění projektu

Aplikace je přístupná jako projekt spustitelný ve vývojovém prostředí Visual Studio. Pro zpřístupnění tohoto projektu jako samotné aplikace je nutné získat spustitelný .EXE soubor se všemi potřebnými zdroji aplikace. Jedním způsobem je vytvoření instalátoru aplikace pomocí nástroje Visual Studia. Tím se vytvoří soubor, z kterého bude možné nainstalovat aplikaci do počítače s operačním systémem Windows pomocí průvodce instalací. Druhým způsobem je získání přenosné tzv. „portable“ verze aplikace z projektu. Přenosnou verzi aplikace lze přenášet do libovolného počítače s OS Windows bez nutnosti instalace aplikace pomocí průvodce. Portable verze poté obsahuje všechny zdrojové soubory aplikace v jedné složce. Vše potřebné pro běh aplikace je přiloženo společně se spustitelným souborem .EXE. Právě tímto způsobem jsem se rozhodl aplikaci publikovat.

Veškeré zdrojové soubory Visual Studio odkládá do složky s projektem. Mezi těmito soubory se nachází složka *bin*. Tato složka obsahuje podsložku *Debug*, ve které se nachází všechny soubory potřebné pro běh aplikace. Tuto složku stačí zkopírovat a dále rozšiřovat mezi ostatní uživatele, kterým poskytne přenosnou verzi aplikace. Aplikace je spustitelná pomocí souboru .EXE.

4.4.2 Distribuce aplikace

Jelikož aplikace slouží především pro parkouristy, rozhodl jsem se aplikaci distribuovat na internetové stránce http://www.dpsc.cz, která je mezi pražskými parkouristy známá a často navštěvovaná. Aplikace je zpřístupněna pro všechny registrované uživatele zdarma v jejich účtu. Aplikaci lze stáhnout jako zkomprimovaný .ZIP soubor – databaze_parkourovych_spotu_v_praze.zip, který stačí rozbalit. Tento .ZIP soubor obsahuje složku databaze_parkourovych_spotu_v_praze. V této složce se nachází složka *Resources* a zástupce aplikace. Aplikace se spustí kliknutím na soubor zástupce aplikace – Databáze parkourových spotů v Praze. Složka Resources obsahuje data potřebná ke správnému spuštění aplikace. Celková velikost aplikace je 87,8 MB.

5 Výsledky a diskuse

5.1 Výsledky

Cílem práce bylo navrhnout a vytvořit funkční databázovou aplikaci poskytující evidenci míst pro trénink parkouru v Praze s využitím programovacího jazyka C# a technologie WPF.

5.1.1 **Kompatibilita**

Aplikace Databáze parkourových spotů v Praze byla vyvinuta pro operační systémy Windows. Testování aplikace probíhalo na několika verzích operačního systému Windows, těmito systémy byly Windows XP, Windows 7 a Windows 10, na kterých aplikace běžela bez problémů. Jediným omezením je minimální rozlišení okna 800x600. Dále byla aplikace testována ve dvou režimech. V režimu s funkčním internetovým připojením a v režimu offline, tedy bez internetového připojení. Oba režimy zobrazují evidenci tréninkových míst v předpokládaném rozsahu v závislosti na internetovém připojení.

5.1.2 **Funkcionalita**

Hlavním dosažením cílem je zobrazení evidence tréninkových míst v Praze. Pro snadnější přístup a přehlednost k jednotlivým tréninkovým místům, tzv. spotům, byla vyvinuta možnost vyhledávání pomocí několika klíčových kritérií spotů a možnost řazení zobrazených spotů.

Každý spot obsahuje kromě samotného jména spotu doplňující informace, které poskytnou uživatelům všechny důležité informace včetně fotogalerie, polohy a dopravy k danému místu.

Pro zlepšení funkcionality je aplikace propojená s webovou prezentací http://www.dpsc.cz, která je webovou obdobou mé aplikace. Aplikace čerpá pomocí databázového systému data z webové prezentace, která poskytuje svá tréninková místa aplikaci. Nad rámec těchto funkcí byla zajištěna možnost registrace vlastního účtu, která zpřístupňuje uživateli další funkcionalitu aplikace. Po registraci je vytvořen účet jak k aplikaci, tak k webové prezentaci. Po přihlášení je získán přístup ke správě vlastních spotů. Uživateli je poskytnuta možnost vytváření vlastních soukromých spotů, které nebyly zveřejněny v databázi aplikace. Dále má uživatel možnost hodnotit spoty a vytvářet si vlastní seznam oblíbených spotů.

5.2 Diskuze

Tato kapitola se věnuje zhodnocení dosažených výsledků při tvorbě aplikace, uvádí potencionální chyby vytvořené aplikace a uvádí možnosti rozšíření.

Aplikace byla vyvinuta pomocí technik objektově orientovaného programování v jazyce C#, které zajišťují znuvupoužitelnost kódu, dobrou čitelnost kódu, možnost využití objektů a další výhody, které zmiňuje ve své knize Sempf (2010). Tyto techniky napomohly k vytvoření stabilní a snadno udržovatelné aplikace. Grafické uživatelské rozhraní bylo postaveno na technologii WPF, která přináší možnost vytvářet bohaté a interaktivní aplikace a je nezbytná pro budoucí vývoj aplikací, jak sám uvádí *Moroney (2006)*. Jednotlivé části aplikace byly rozděleny do samostatných tříd. Třídy dědící z rodičovské třídy *Window* propojují logiku aplikace s grafickým uživatelským rozhraním. *Stellman (2007)* upozorňuje na možnost využití Data Bindingu pro propojení logiky aplikace s uživatelským rozhraním u aplikací postavených na technologii WPF. Má aplikace se touto informací řídila a na vhodných místech byl Data Binding využit. Uchování dat bylo vyřešeno pomocí dvou relačních databázových systémů. Jako externí databáze byla využita MySQL databáze, která je v dnešní době využívaná většinou uživatelů, kteří volí externí databáze pro svá data. Interní databáze aplikace byla postavena na SQL Server Compact Edition. Mnoho vývojářů, kteří pro své aplikace používají SQL Server, již tuto databázi nevyužívá. Mnohem častěji využívají SQL Server Express, která poskytuje více funkcí. Já jsem databázi SQL CE i přes tuto skutečnost zvolil díky její snadné přenositelnosti a jednoduchosti. Výhodami těchto relačních databází jsou také rychlost, správa, kompaktnost. Tyto výhody potvrzuje ve své knize *Agarwal (2008)*.

Pomocí těchto technik vznikla databázová aplikace umožňující uživateli zobrazovat evidenci parkourových tréninkových míst v Praze, přičemž aplikace obsahuje přes 200 míst. Další tréninková místa může uživatel po registraci a přihlášení vytvářet pomocí přehledného formuláře vytvořeného značkovacím jazykem XAML a technologií WPF. Tréninková místa se synchronizují s externí databází MySQL a tím poskytují uživateli stále nová data.

Z důvodu využívání externí databáze je potencionální hrozbou aplikace právě závislost aplikace na této databázi. V případě krátkodobé nefunkčnosti externího databázového serveru by mohlo dojít k nemožnosti využívat online zdroje dat. V takovém případě však stále existuje interní databáze zajišťující funkčnost aplikace. Pokud by došlo k úplnému smazání externí databáze je jako jednou z možností založit novou externí databázi a poskytnout uživatelům

novou verzi aplikace. Dalším teoretickým místem pro vznik chyb je synchronizace dat mezi interní a externí databází, avšak při testování nevykázala aplikace v tomto ohledu žádné chyby.

Jedním směrem možného rozšíření aplikace je zavedení více informací o jednotlivých evidovaných místech. Toto rozšíření je v budoucnu velmi pravděpodobným krokem pro zlepšení funkčnosti aplikace. S odstupem času a dobré odezvy uživatelů lze uvažovat o rozšíření aplikace mimo město Praha. S rozšířením aplikace i na jiná města je ovšem zapotřebí spolupráce s parkouristy z ostatních měst České republiky. Takovéto rozšíření aplikace by mohlo vést k užívání aplikace uživateli v celé České republice, což by s největší pravděpodobností přispělo ke zlepšení a ucelení tohoto sportu v České republice.

5.3 Závěr

Tato bakalářská práce se zaměřovala na problematiku vývoje databázové aplikace pro evidenci parkourových tréninkových míst v Praze pomocí programovacího jazyka C# s využitím technologie WPF.

V teoretické části byla provedena analýza technologií potřebných k vytvoření práce. Na začátku byla popsána platforma .NET Framework. Následoval programovací jazyk C# od základů jazyka k objektově orientovanému programování. Dalším bodem byla technologie WPF a značkovací jazyk XAML. Zde se práce zaměřovala na grafiku, strukturu a syntaxi těchto technologií. Bylo nahlédnuto do vývojového prostředí Visual Studio. Poslední část řešila vhodné databázové systémy pro ukládání dat.

V praktické části této práce byly popsány jednotlivé kroky od návrhu k publikaci databázové aplikace. Na základě syntézy poznatků z teoretické části práce s přihlédnutím k potřebám pražských parkouristů byly stanoveny požadavky funkcionality aplikace a vytvořen use case diagram. Na tomto základě byl proveden návrh aplikace, byly navrhnuty jednotlivé třídy, uživatelské rozhraní a databáze. Tyto části aplikace byly implementovány pomocí technologií uvedených v teoretické části práce. V závěru praktické části byla uvedena distribuce aplikace mezi požadovanou skupinu uživatelů.

Aplikace je plně kompatibilní s operačními systémy Windows od verze Windows XP. Funkcionalita aplikace poskytuje evidenci pražských spotů s podrobnými informacemi, jejich vyhledávání a řazení, ukládání seznamů spotů a tvoření vlastních spotů. Veškeré uvedené požadavky aplikace byly úspěšně splněny.

6 Seznam použitých zdrojů

1. AGARWAL, V.V -- HUDDLESTON, J. -- RAGHURAM, R. Beginning C# 2008 Databases From Novice to Professional. New York, 2008. ISBN13: 978-1-59059-900-6

2. BĚHÁLEK, Marek. Programovací jazyk C# [online]. VŠB-TU Ostrava, 2007 [cit. 2016-05-03]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf

3. HANÁK, Ján. *Praktické objektové programování v jazyce C# 4.0*. Brno: Artax, 2009. Microsoft (Artax). ISBN 978-80-87017-07-4.

4. MACDONALD, M. Pro WPF 4.5 in C# Windows Presentation Foundation in .NET 4.5. New York, 2012. ISBN: 978-1-4302-4365-6

5. MAMTA DALAL, Ashish Ghoda. *XAML developer reference*. Sebastopol, Calif: O'reilly Media, 2011. ISBN 9780735658967.

6. NAKOV, S -- KOLEV, V. Fundamentals of Computer Programming with C#. Sofia, 2013. ISBN: 978-954-400-773-7

7. PIRKL, J. Řešené příklady v C sharp -- aneb C# skutečně prakticky. 2005. ISBN: 80-7232-265-6
8. SEMPF, B. -- DAVIS, S.R. -- SPHAR, C. C# 2010 All-in-One For Dummies. Hoboken, 2010. ISBN: 978-0-470-56348-9

9. STELLMAN, Andrew. a Jennifer GREENE. *Head first C#*. Sebastopol, CA: O'Reilly, c2007. ISBN 0596514824.

10. YOSIFOVICH, P. Windows Presentation Foundation 4.5 Cookbook 2012. Birmingham, UK, 2012. ISBN: 978-1849686228

11. TROELSEN, Andrew W. Pro C# 2010 and the .NET 4 platform. 5th ed. New York, NY: Apress, c2010. ISBN: 9781430225492.

12. ALBAHARI, Joseph., Ben. ALBAHARI a Peter. DRAYTON. C# 4.0 in a nutshell. 4th ed. Sebastopol: O'Reilly, c2010. In a nutshell (O'Reilly & Associates). ISBN: 9780596800956.

13. NATHAN, Adam. WPF 4 unleashed. Indianapolis, Ind.: Sams, c2010. ISBN 0672331195.

14. GAROFALO, Raffaele. Applied WPF 4 in context. New York, c2011. ISBN 1430234709.

15. PERKINS, B. -- HAMMER, J. – REID, J, Beginning C# 6 Programming with Visual Studio 2015, Indianapolis, Indiana, 2016. ISBN: 978-1-119-09668-9¨

16. JOHNSON, Bruce. Professional visual studio 2013. Indianapolis, Indiana, 2016. ISBN: 1118832043.

17. MORONEY, Laurence. Foundations of WPF: an introduction to Windows Presentation Foundation. New York, c2006. ISBN: 1590597605.

7 Přílohy

Příloha A: Obsah přiloženého CD

Na přiloženém CD v adresáři ‚A) APLIKACE' se nachází vytvořená aplikace spustitelná pomocí souboru – *Databáze parkourových spotů v Praze.exe*. V adresáři ‚B) ZDROJOVÉ SOUBORY' se nachází celý projekt aplikace, se všemi zdrojovými kódy, spustitelný ve vývojovém prostředí Visual Studio. V adresáři ‚C) TEXT' se nachází tato bakalářská práce ve formátu PDF – *xkoll009.pdf*.