VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

NÁVRH DATABÁZOVÉHO SYSTÉMU PRO SPRÁVU AKCÍ
PROPOSAL OF DATABASE SYSTEM FOR EVENT MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Radim Sekula**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jan Luhan, Ph.D., MSc**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Radim Sekula**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Informační management
Vedoucí práce:	**Ing. Jan Luhan, Ph.D., MSc**
Akademický rok:	2016/17

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Návrh databázového systému pro správu akcí

Charakteristika problematiky úkolu:

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

Cíle, kterých má být dosaženo:

Cílem práce je navrhnout databázový systém pro správu akcí zaměřený na konkrétní podmínky vybraného subjektu. Výstup bude realizován s podporou .NET frameworku a dle konkrétních požadavků vybraného subjektu.

Základní literární prameny:

AGARWAL, V. V. a J. HUDDLESTON. Databáze v C# 2008: průvodce programátora. 1. vyd. Brno: Computer Press, 2009. 424 s. ISBN 978-80-251-2309-6.

CONOLLY, T., C. E. BEGG a R. HOLOWCZAK. Mistrovství – databáze: Profesionální průvodce tvorbou efektivních databází. 1. vyd. Brno: Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.

EVJEN, B., S. HANSELMAN a D. RADER. ASP.NET 3.5 v jazycích C# a Visual Basic. 1. vyd. Brno: Computer Press, 2009. 1598 s. ISBN 978-80-251-2069-9.

KROENKE, D., D. J. AUER a J. GONER. Databáze. 1. vyd. Brno: Computer Press, 2015. 496 s. ISBN 978-80-251-4352-0.

SHARP, J. Microsoft Visual C# 2010: krok za krokem. 1. vyd. Brno: Computer Press, 2010. 696 s. ISBN 978-80-251-3147-3.

WATSON, B. C# 4.0: řešení praktických programátorských úloh. 1. vyd. Brno: Zoner Press, 2010. 656 s. ISBN 978-80-7413-094-6.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2016/17.

V Brně, dne 28. 2. 2017

L. S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

doc. RNDr. Bedřich Půža, CSc.	doc. Ing. et Ing. Stanislav Škapa, Ph.D.
ředitel	děkan

ABSTRAKT

Diplomová práce je zaměřena na návrh databázového systému pro správu akcí dle požadavků zadavatele, jímž je společnost ANeT-Advanced Network Technology, s. r. o. V první části práce jsou vymezena teoretická východiska databázových systémů a objektově orientovaného programování se specializací na .NET Framework. Druhá část analyzuje současný stav a konkrétní potřeby společnosti, na základě kterých je ve třetí části vytvořen návrh řešení.

ABSTRACT

The Master's thesis deals with the proposal of database system for event management according to the requirements of the client, which is ANeT-Advanced Network Technology, Ltd. The first part of the thesis summarizes the theoretical bases of database systems and object-oriented programming focusing on .NET Framework. In the second part the current situation and the specific needs of the company are analyzed, and based on this analysis a solution proposal is created in the third part.

KLÍČOVÁ SLOVA

Databázový systém, MS SQL, .NET, C#, Entity Framework, WCF, LINQ

KEY WORDS

Database system, MS SQL, .NET, C#, Entity Framework, WCF, LINQ

BIBLIOGRAFICKÁ CITACE

SEKULA, R. *Návrh databázového systému pro správu akcí*. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2017. 82 s. Vedoucí diplomové práce Ing. Jan Luhan, Ph.D., MSc.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 25. května 2017 ……

Radim Sekula

PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Janu Luhanovi, Ph.D., MSc. za odborné vedení, vstřícný přístup a věcné rady při zpracování této diplomové práce.

Dále bych rád poděkoval vedoucím zaměstnancům společnosti ANeT-Advanced Network Technology, s.r.o. za ochotu, poskytnuté informace a cenné připomínky při zpracování praktické části.

V neposlední řadě bych chtěl také poděkovat mé přítelkyni a rodině za jejich podporu během studia.

OBSAH

ÚVOD

Firemní akce jsou podstatnou součástí firemní komunikace. Při těchto akcích, ať už společenských, kulturních nebo sportovních, dochází k prohlubování a utužování vztahů se stávajícími zaměstnanci, vytváření vztahů se zaměstnanci nově nastupujícími a jejich začlenění do kolektivu. Pro tyto účely je vhodné, aby společnost disponovala přehledným systémem pro správu akcí, kde si každý ze zaměstnanců může vybrat podle svého zájmu a časových možností, které události se chce zúčastnit.

Tato diplomová práce se zaměřuje na návrh databázového systému pro správu firemních akcí pro společnost ANeT-Advanced Network Technology, s. r. o.

První část poskytuje základní přehled o datovém modelování, databázových systémech, objektově orientovaném programování a platformě .NET. Tyto teoretické poznatky jsou důležité pro pochopení návrhové části práce.

V druhé kapitole je analyzován současný stav správy akcí ve společnosti. Tato analýza ukazuje, že současné řešení správy dat firemních akcí je opravdu nepostačující a že toto řešení může snižovat zájem zaměstnanců o firemní akce. Na základě analýzy jsou stanoveny konkrétní požadavky na nové řešení.

Ve třetí části práce je vytvořen návrh řešení s ohledem na konkrétní požadavky společnosti ANeT. Návrh řešení je založen na technologiích, kterými společnost disponuje, a výstupem tohoto návrhu je databázový systém, který představuje základ pro další vývoj ve společnosti.

CÍL PRÁCE A METODIKA

Cílem této diplomové práce je navrhnout databázový systém pro správu akcí zaměřený na konkrétní podmínky společnosti ANeT-Advanced Network Technology, s. r. o. Výstup práce bude realizován s podporou .NET Frameworku a dle konkrétních požadavků vybrané společnosti.

Výchozím bodem práce bude analýza současného stavu správy firemních akcí, jež potvrdí nutnost zavedení nového způsobu správy firemních akcí. Dalším krokem bude stanovení požadavků, které budou kladeny na nové řešení.

S ohledem na používané technologie ve společnosti bude vybrán databázový server. Nezbytným krokem bude také analýza softwarových nástrojů pro objektově relační mapování sloužící k provázání dat z relační databáze do objektově orientovaného jazyku C#. Pro tento účel bude vybrán vhodný objektově relační framework.

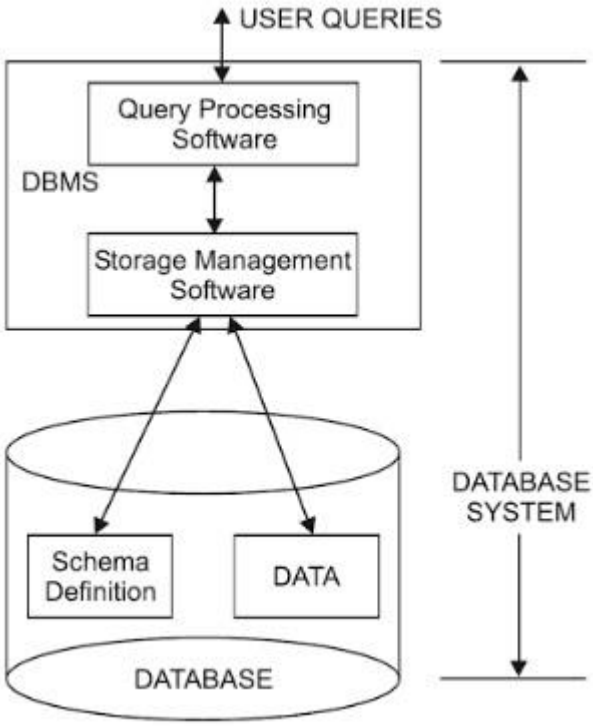
Dále bude vytvořen návrh databázového systému, který bude zahrnovat následující kroky. Na počátku bude implementována databáze na vybraném databázovém serveru. Poté budou propojena data z relační databáze do objektově orientovaného programování použitím vybraného objektově relačního frameworku. Následně bude implementována WCF služba, která bude využívat data z relační databáze. Nakonec bude provedeno unit testování vytvořené WCF služby. Řešení bude taktéž doplněno o mechanismus pro správu uživatelů ASP.NET Identity.

1 TEORETICKÁ VÝCHODISKA PRÁCE

Cílem této části je poskytnout základní přehled o databázových systémech, datovém modelování, objektově orientovaném programování a platformě .NET, a to především jazyku C#, LINQ a technologii Windows Communication Foundation. Tyto poznatky budou tvořit základ pro celkovou tvorbu této diplomové práce.

1.1 Databázový systém

Databázový systém se skládá ze dvou hlavních částí – databáze a systému řízení báze dat. Systém řízení báze dat (SŘBD, anglický Database Management System DBMS) jsou speciální programy, které zajišťují řídící činnost a umožňují především spojení uživatele a uložených dat. (9)



Obrázek 1: Databázový systém (9, str. 1)

Jak je vidět na obrázku 1, *DBMS* (systém řízení báze dat) reaguje na *user queries* (dotazy od uživatele) prostřednictvím *Query Processing Software*, což je speciální software, který tyto uživatelské dotazy zpracovává. Ten poté komunikuje se *Storage Management Software*, což je program, který zajišťuje správu ukládání dat. Ten na

základě uživatelského dotazu provede potřebné operace s daty, jež jsou uložené v databázi a k nimž má přístup. (9)

1.1.1 Databáze

„Databáze je kolekce vzájemně souvisejících datových položek, které jsou spravovány jako jediná jednotka." (12, str. 9)

1.1.2 Systém řízení báze dat

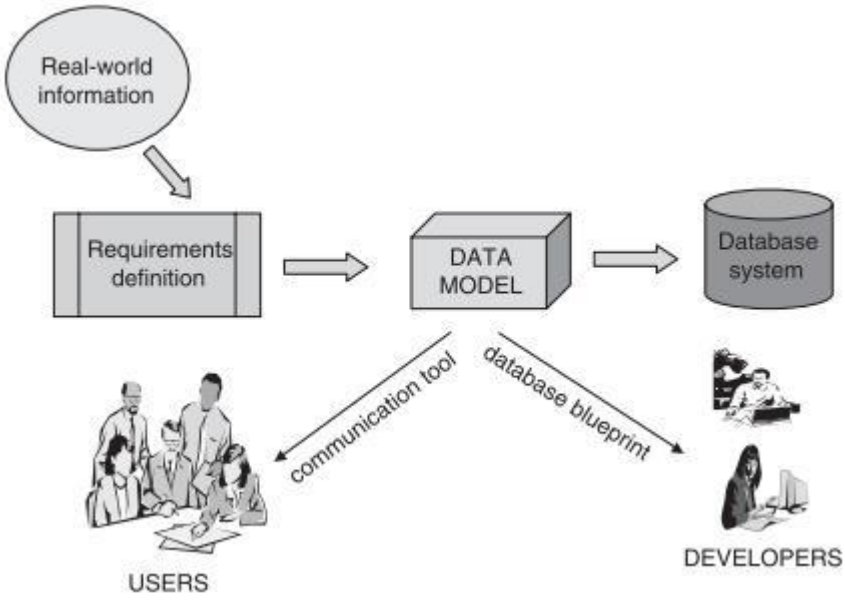
„Systém řízení báze dat (DBMS) je softwarový systém, který uživateli umožňuje definovat, vytvářet a udržovat databázi a poskytuje řízený přístup k této databázi.“ (2, str. 38)

Systém řízení báze dat (DBMS) je software, který interaguje s uživateli, databázovými aplikacemi a s databází. Kromě jiného umožňuje DBMS uživatelům vkládat, aktualizovat, mazat a vyvolávat data z databáze. (2)

V této diplomové práce je jako DBMS použit Microsoft SQL Server 2012.

1.2 Datový model

Datový model je jedním z nejzásadnějších pojmů v oboru databázových systémů. Datový model představuje zachycená a uložená data o reálném světě včetně jejich vzájemných vazeb. Označujeme jím v podstatě architekturu, na základě které se objekty databázovým systémem ukládají do databáze. Tato architektura dané objekty navzájem spojuje. Mezi nejrozšířenější datové modely řadíme hierarchický, síťový, relační, objektově orientovaný či objektově relační model. (4)



Obrázek 2: Datový model v procesu návrhu databázových systémů (13, str. 5)

Na obrázku 2 je možné vidět, jakou roli hraje datový model v procesu navrhování databázového systému. Datový model slouží jako kritický nástroj pro komunikaci s uživateli a taktéž jako blueprint (plán) databázového systému pro vývojáře. Datový model pomáhá uživatelům jasně pochopit databázový systém, který je implementován na základě informačních požadavků. (13)

„Datové modelování poskytuje metody a prostředky pro požadavky na informace reálného světa tak, aby je pochopily zainteresované strany v organizaci. To znamená, že datové modelování umožňuje databázovým expertům použít tyto požadavky na informace a implementovat je do počítačového databázového systému, který bude podporovat podnikání organizace.“ (13, str. 5)

1.2.1 Relační datový model

Relační model byl formálně představen v roce 1970. V současné době je tento model, založený na teorií relací, nejvíce rozšířenou a používanou metodou. Vzniká spojením několika lineárních modelů prostřednictvím relačních klíčů. Data jsou ukládána ve formě dvourozměrných tabulek. Jednoznačným primárním klíčem jsou identifikovány jednotlivé věty tabulek. Spojení mezi tabulkami není trvalé, ale pouze přechodné, trvá po dobu potřebnou k získání dat. V relačním modelu je možné zachycení nejen dat o požadovaných objektech, ale i jejich vzájemných vztahů. (14)

Relační datový model má pět hlavních složek, jmenovitě relaci, atribut, datovou n-tici, doménu a relační databázi.

- **Relace** – tabulka se sloupci a řádky
- **Atribut** – pojmenovaný sloupec relace
- **Datová n-tice** – řádek relace
- **Doména** – množina přípustných hodnot pro jeden nebo více atributů
- **Relační databáze** – kolekce normalizovaných tabulek

V relačním modelu se k uložení informací o objektech, jež je potřeba v databázi reprezentovat, používá relace. Relace představuje tabulku, v níž řádky odpovídají jednotlivých datovým n-ticím a sloupce odpovídají atributům. Atributy se můžou vyskytovat v jakémkoliv pořadí, aniž by šlo o jinou relaci (obsahově bude mít relace stejný význam). Každý atribut v relační databázi je spojen s doménou. Ta může být odlišná pro každý atribut nebo může být dvě či více domén spojeno se stejným atributem. Relační databáze se skládá ze strukturovaných normalizovaných tabulek. (2)

1.2.2 Relační databáze

Relační databáze je databáze založená na relačním modelu. Fyzickou reprezentací dat jsou normalizované dvourozměrné tabulky, které lze spojovat dle potřeby. Kombinace (spojení) tabulek umožňuje vytvářet pohledy, zobrazující se taktéž jako dvourozměrné tabulky. Tato schopnost nezávislého využití tabulek či spojení s ostatními tabulkami, bez jakékoliv předem definované hierarchie nebo posloupnosti, činí relační databázi velmi flexibilní. (12)

1.3 Objektově orientované programování

Předstupeň objektově orientovaného programování je sekvenční programování. Mezi sekvenční programovací jazyky patří například Fortran a Assembler, které vznikly v 50. letech 20. století. Sekvenční programování se provádělo psaním sekvence příkazů, jež byly vykonávány postupně dle pořadí zápisu. Nevýhodou ovšem byla absence objektů, které v současném programování šetří velkou spoustu času.

Skutečnost potřeby předávat řízení v jedné dlouhé sekvenci příkazů a do jisté míry nemožnosti případných úprav či odstraňování chyb vedly ke vzniku strukturovaného programování. Sekvence příkazů byla nahrazena hierarchickou strukturou stromového typu, v které byly zavedeny cykly, podmínky a další nástroje, které usnadňovaly práci programátora. Hlavní rozdíl a výhoda oproti sekvenčnímu programování je, že bloky kódu jsou jednoznačně ohraničeny a nedochází tak ke skokům v sekvenci kódu, kde je úprava či odstraňování chyb podstatně jednodušší záležitostí. Nejznámějším strukturovaným jazykem je programovací jazyk C.

Rigidní hierarchie se ukázala jako hlavní nevýhoda strukturovaného programování, protože neodpovídá modelu „reálného světa“, který není hierarchický. Obraz reálného světa se snaží vystihnout objektově orientované programování, které rozbíjí hierarchickou strukturu kódu a je založeno na síti vzájemně komunikujících objektů. (10)

Základní myšlenky objektově orientovaného programování se poprvé objevily v jazyce Simula67. Tento jazyk posloužil jako inspirace pro vznik nového programovacího jazyka Smalltalk, který jako první přinesl samotný termín OOP. Postupem času se techniky OOP rozšířily na univerzity a mezi vědce, avšak tyto techniky byly zřídkakdy využívány mimo akademickou půdu. Zlom nastal až když Bjarne Stroustrup pomocí soustavy maker upravil jazyk C, tak aby umožňoval použití tříd a objektů. Takto vznikl jazyk C with classes, který se díky své efektivitě začal rychle rozšiřovat a v roce 1983 byl přejmenován na C++. Popularita jazyka C++ rozšířila techniky OOP do všech odvětví softwarového inženýrství. (1)

1.3.1 Objekt

Základní jednotkou OOP je objekt, který odpovídá nějakému objektu z reálného světa (např. osoba, kniha, auto). Jednotlivé objekty se mohou skládat z jiných objektů (např. počítač se skládá ze základní desky, procesoru, paměti RAM, grafické karty a HDD). Objekty také můžeme klasifikovat do hierarchických struktur ve smyslu specializace a generalizace (např. objekt osoba je obecnějším (generalizovanějším) objektem než objekty student nebo učitel a podobně objekt míčový sport je

specializovanějším objektem než objekt sport). Každý objekt má své atributy a metody. (3) (10)

- **Atributy** – vlastnosti neboli data, která objekt uchovává (např. u osoby jméno a věk, u knihy název a rok vydání). Jinými slovy se jedná o prosté proměnné, o nichž někdy hovoříme jako o vnitřním stavu objektu
- **Metody** – činnosti, které umí daný objekt vykonat (např. u osoby Pozdrav(), u knihy Vyhledej()). Metody můžou mít parametry a můžou také vracet nějakou hodnotu. (10)

1.3.2 Třída

Objekty mohou být různého typu ve stejném smyslu, jako mohou být i data různého typu. Pojem třída představuje množinu objektů se shodnými charakteristikami. Jinak řečeno třída je prvek objektového počítačového systému, který odpovídá „druhu objektu“. U každého objektu je zřejmé, které třídě náleží, a mezi třídami je vhodným způsobem definována hierarchie postihující jejich vzájemné podobnosti. Třída je vzor, podle kterého se objekty vytváří. (3) (10)

1.3.3 Instance třídy

Třídu je možné nadefinovat a poté vytvářet instance této třídy, což jsou vlastně skutečné objekty. Instance mají stejné rozhraní jako třída, podle které se vytváří, ovšem navzájem se liší svými daty (atributy). (10)

1.3.4 Zapouzdření

Zapouzdření vyjadřuje schopnost spojení atributů a metod. Objekt dané třídy má kromě atributů, které určují jeho vlastnosti a stav, jasně určeno, které operace může provádět. Zapouzdření umožňuje skrýt některé metody a atributy tak, aby zůstaly použitelné jen pro objekt zevnitř. Pro ostatní objekty, které s ním komunikují, by měl být „černou skřínkou“, jejíž vnitřní struktura je dokonale skryta. (10)

1.3.5 Dědičnost

Dědičnost (inheritance) v OOP představuje mechanismus vytváření tříd a podtříd. Podtřída dále specializuje svoji nadtřídu, a to formou dalších datových atributů nebo dalších resp. specializovanějších metod. To znamená, že při sestavování

nové třídy může programátor zvolit jakoukoliv z již existující tříd za nadtřídu. Nová třída potom automaticky zdědí všechny vlastnosti nadtřídy.

Rozlišujeme následující typy dědičnosti:

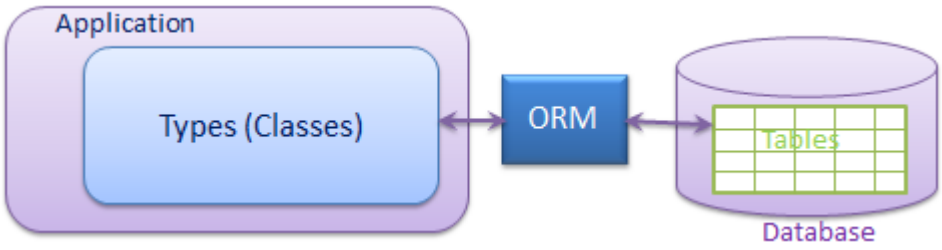
- **Jednoduchá** – třída má jen jednoho předka (rodiče), jde o stromovou hierarchii tříd, kdy třídu v nejvyšší úrovni označujeme jako kořenovou
- **Vícenásobná** – třída může mít více předků
- **Opakovaná** – třída může zdědit vlastnosti vzdálenějšího předka více cestami, vztahy v hierarchii jsou znázorňovány orientovaným acyklickým grafem (DAG), označovaným jako graf příbuznosti tříd (10)

1.3.6 Polymorfismus

Polymorfismus požaduje, abychom mohli jedním a týmž způsobem komunikovat s různými objekty. Úzce souvisí se zapouzdřením – nevidíme-li do černé skříňky, nevidíme, co obsahuje a jak (pokud vůbec) se liší od jiných černých skříněk, je tedy jasné, že se všemi budeme komunikovat týmž způsobem. Naopak jestliže se všemi objekty komunikujeme stejně (tedy polymorfně), není nutné, abychom pátrali po vnitřním obsahu – není problém se zapouzdřením. (10)

1.4 Objektově relační mapování

Relační databáze i objektově orientované programování patří mezi běžné nástroje současného softwarového vývoje. Jelikož relační databáze objektově nefungují a objekty ukládat neumí, nastává mezi těmito nástroji rozpor, kterým je potřeba se zabývat. Jednou z možností, jak se s tímto problémem vypořádat je objektově relační mapování.



Obrázek 3: Objektově relační mapování (16)

Objektově relační mapování (dále jen ORM) je technologie, která umožňuje provázat data mezi relační databází a objekty v objektově orientovaných jazycích. (7)

V této diplomové práci je použitý .NET Entity Framework, který zabezpečuje mapování mezi databázovými tabulkami a nadefinovanými entitami. Blíže je popsán v kapitole 2.5.3.

1.5 .NET Framework

.NET Framework představuje programový model využívaný programátory, kteří mohou psát svůj kód v podstatě jakémkoliv plně objektovém programovacím jazyce. Mezi tyto jazyky se řadí C# .NET, C++ .NET, Visual Basic .NET.

Součástí tohoto frameworku je i vylepšený mechanismus pro správu paměti (Garbage Collector - GC), který monitoruje paměť a zdroje použité v aplikaci a podle potřeby je vrací zpět systému. Toto vylepšení pomáhá v programování v případech, kdy je kód špatně napsán a touto chybou by za normálních okolností došlo k vyčerpání systémových prostředků. Jednoduše řečeno, GC, nám sám uvolní paměťové prostředky nad již nepoužívanými objekty, pokud to programátor „zapomene“ implementovat sám.

Další vylepšení souvisí se zpracováním chybových stavů, které se ve frameworku realizují pomocí výjimek a jejich následnému zpracování (chyba může být zpracována obecně nebo na konkrétní úrovni, například dělení nulou, indexování mimo povolené hranice pole apod.). (17)

.NET Framework poskytuje tyto možnosti:

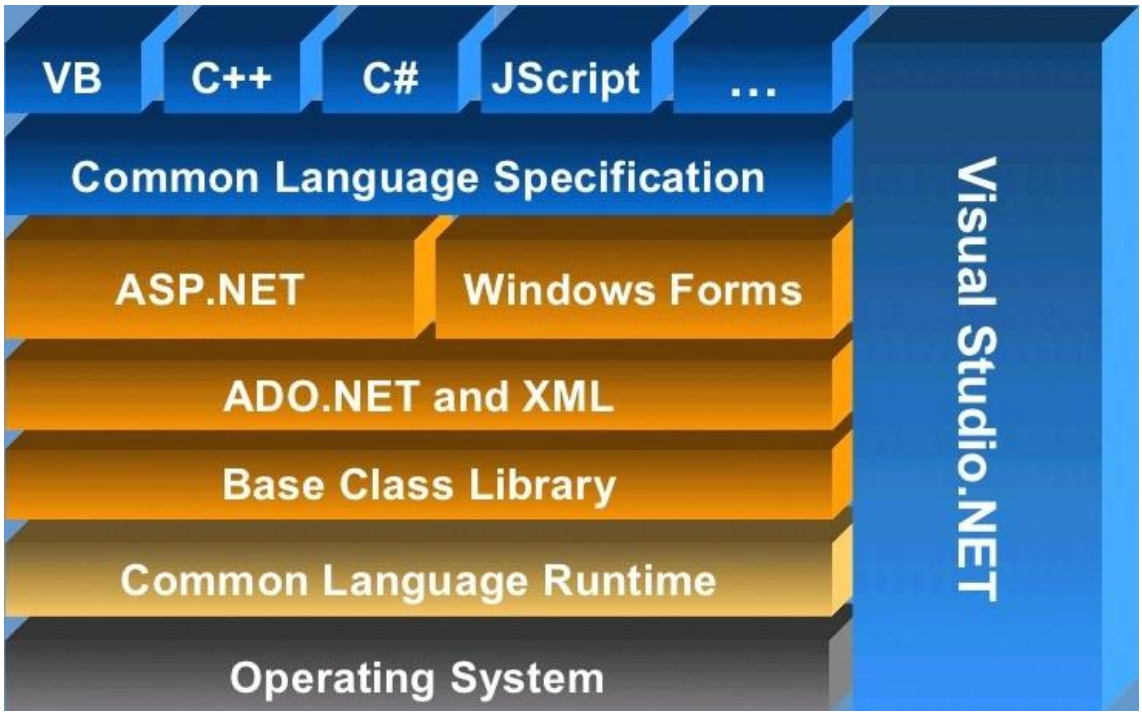
- Slouží jako run-time prostředí, ve kterém běží aplikace (znamená to, že aplikace napsané v tomto prostředí nelze bez tohoto prostředí spustit)
- Nabízí možnost pro tvorbu webových služeb a ASP.NET stránek
- Poskytuje skutečně velmi širokou řadu knihoven obsahující objektově orientované třídy umožňující snadněji řešit řadu úkolů (jako jsou práce s databází a datovými zdroji, bezpečnost, komunikace)
- Umožňuje tvorbu desktopových aplikací na platformě WindowsForms, WPF, Universal, Windows App

- Tvorba Windows Services a aplikací pro mobilní platformy - Xamarin (17)

Aktuálně nejnovější a dostupná verze .NET Frameworku je verze 4.7.

1.5.1 Architektura .NET Framework

Na obrázku 4 se nachází architektura .NET Framework.



Obrázek 4: Architektura .NET Framework (17)

Na nejnižší úrovni se nachází *CLR - Common Language Runtime* realizující základní infrastrukturu, nad kterou je framework vybudován. Nad CLR se nachází několik hierarchicky umístěných knihoven. Ty jsou rozděleny do jmenných prostorů. Základem je knihovna *BCL - Base Class Libary*. Nad ní je knihovna pro přístup k datům a práci s XML soubory. Poslední vrstvou je sada knihoven usnadňující práci s uživatelským rozhraním. Je rozdělena do dvou skupin: pro usnadnění vytváření webových aplikací a pro vytváření klasických aplikací. Poslední vrstvu tvoří nelimitovaná množina programovacích jazyků. Jejich základní vlastnosti definuje *CLS – Common Language Specification*. Celá tato architektura je podpořena verzí vývojového nástroje Visual Studio .NET. (17)

Existuje více než 50 jazyků, které lze použít pro psaní aplikací v prostředí .NET – například Eiffel, Smalltalk, COBOL, Haskell, Pizza, Pascal, Delphi, Oberon, Prolog a Ruby. Každý jazyk má své výhody i nevýhody, některé věci se v jednom jazyku

provádějí jednodušeji a v jiném zase obtížněji. Třídy v .NET jsou vždy stejné, ovšem v syntaxi konkrétního jazyka je kladen důraz na různé vlastnosti této platformy.

Nejčastěji používané jazyky .NET od Microsoftu jsou C# a Visual Basic. C# byl nově vytvořen pro .NET a přebral myšlenky z C++, Javy, Pascalu a dalších jazyků. Právě v této diplomové práci je použitý jazyk C#, který je blíže popsán v následující kapitole.

1.6 Jazyk C#

Jazyk C# je výkonný, objektově orientovaný programovací jazyk vycházející z jazyků C/C++ a jazyku Java. První zmínky o tomto jazyce se dostali mezi veřejnost již v roce 2000. Oficiálního uvedení na trh proběhlo až v roce 2002 a to společně s celým vývojovým prostředím .NET.

Označení C# není jediné, se kterým se lze setkat. Někde lze narazit i na zápis C # a v některých naučných textech i na zápis v textovém přepisu C Sharp. Význam tohoto označení nebyl nikdy oficiálně publikován, a proto si lze jen domýšlet, co autoři za něj schovali.

Mezi hlavní rysy jazyka C# patří fakt, že je čistě objektově orientovaný a je součástí vývojových prostředí Visual Studio .NET. Seznam vlastností jazyka ukazuje následující přehled. (1)

- **Jednoduchou dědičnost** – každá třída muže být potomkem pouze jedné třídy, avšak naopak každá třída muže mít jednoho nebo více potomků.
- **Násobná implementace rozhraní** – každá třída může implementovat jedno nebo více rozhraní a každé rozhraní může být implementováno jednou nebo více třídami.
- **Garbage collector** – automaticky určuje, která část paměti již není používána a následně ji připraví pro znovu použití. Samotné uvolnění paměti je označováno jako *garbage collecting*.

- **Členská data** – jsou privátní položky uvnitř třídy a nelze s nimi pracovat mimo danou třídu.
- **Vlastnosti** – poskytují mechanismus pro čtení a zápis v rámci privátních položek třídy.
- **Metody** – jsou funkce definované v rámci nějaké třídy. Pomocí přístupových modifikátorů *public, protected* a *private* určíme, zda lze metodu použít i mimo třídu. Dalšími modifikátory jsou *static* a *abstract*.
- **Události** – umožňují třídě upozornit jinou třídu nebo více tříd, že v ní došlo ke změnám. Ostatní třídy pak mohou tuto událost zachytávat a příslušným způsobem na ní zareagovat.
- **Zpracování chyb pomocí výjimek** – je objekt nesoucí informaci o chybě, která v rámci provádění programu nastala a zachycením tohoto objektu lze na daný chybový stav patřičným způsobem reagovat.
- **Zajištění zpětné kompatibility v binární podobě** – program je schopen pracovat i se starší verzí knihoven bez potřeby nového překladu.
- **Zajištění zpětné kompatibility v podobě zdrojových kódu** – je vyžadován nový překlad, ale není třeba změn v kódu. (5)

1.6.1 Jmenné prostory

Při importu jmenných prostorů v C# se používá klíčové slovo *using*. (1)

```
using System.Linq;
```

1.6.2 Definice typů

.NET rozlišuje referenční typy a hodnotové typy. V C# se referenční typy definují pomocí tříd a hodnotové typy pomocí struktur. Následující přehled také ukazuje, jak definovat rozhraní (referenční typ) a výčet (hodnotový typ).

- **Referenční typy** – při deklaraci se používá klíčové slovo *class*, v C# je třída uzavřena do složených závorek {}, na rozdíl například od jazyka Visual Basic, který používá na konci třídy výraz *End Class*.

```
public class MyClass
```

```
{
}
```

Při použití referenčního typu, je potřeba deklarovat proměnnou a vytvořený objekt musí být umístěn na řízené haldě. V C# začíná deklarace proměnné referenčním typem, za nímž následuje název proměnné, přiřazovací operátor =, příkazem *new* a typem objektu je vyhrazena paměť na řízené haldě.

```
MyClass obj = new MyClass();
```

- **Hodnotové typy** – k deklaraci se používá výraz *struct*

```
public struct MyStruct
{
}
```

- **Dovozený typ** – pomocí výrazu *var* je možné definovat lokální proměnnou bez explicitní deklarace datového typu. Typ se dovozuje z počáteční hodnoty přiřazené této proměnné.

```
var x = 3;
```

- **Rozhraní** – definice rozhraní používá klíčové slovo *interface*

```
public interface IDisplay
{
	void Display();
}
```

V implementaci rozhraní se používá dvojtečka za názvem třídy, za níž následuje název rozhraní. Implementují se tak metody definované v rozhraní.

```
public class Person: IDisplay
{
	public void Display();
	{
	}
}
```

- **Výčty** – definují se pomocí klíčového slova *enum*

```
public enum class Color
{
	Red, Green, Blue
}
```

(1) (5)

1.6.3 Metody

Metody se vždy definují uvnitř třídy.

```
public class MyClass
{
	public void Foo()
	{
	}
}
```

Parametry metod a návratové typy

V C# se parametry předávané metodám definují v závorkách. Typ parametru se zapisuje před jméno proměnné. Metoda, která vrací nějakou hodnotu, se definuje s návratovým typem jiným než *void*.

```
public class MyClass
{
	public int Foo(int i)
	{
		return 2 * i;
	}
}
```

Modifikátory parametrů

Ve výchozím nastavení se hodnotové typy předávají hodnotou a referenční typy se předávají odkazem. Pokud chceme ve volající metodě změnit hodnotový typ předávaný jako parametr, můžeme v C# použít modifikátor parametru *ref*.

```
public class ParameterPassing
{
	public void ChangeVal(ref int i)
	{
		i = 3;
	}
}
```

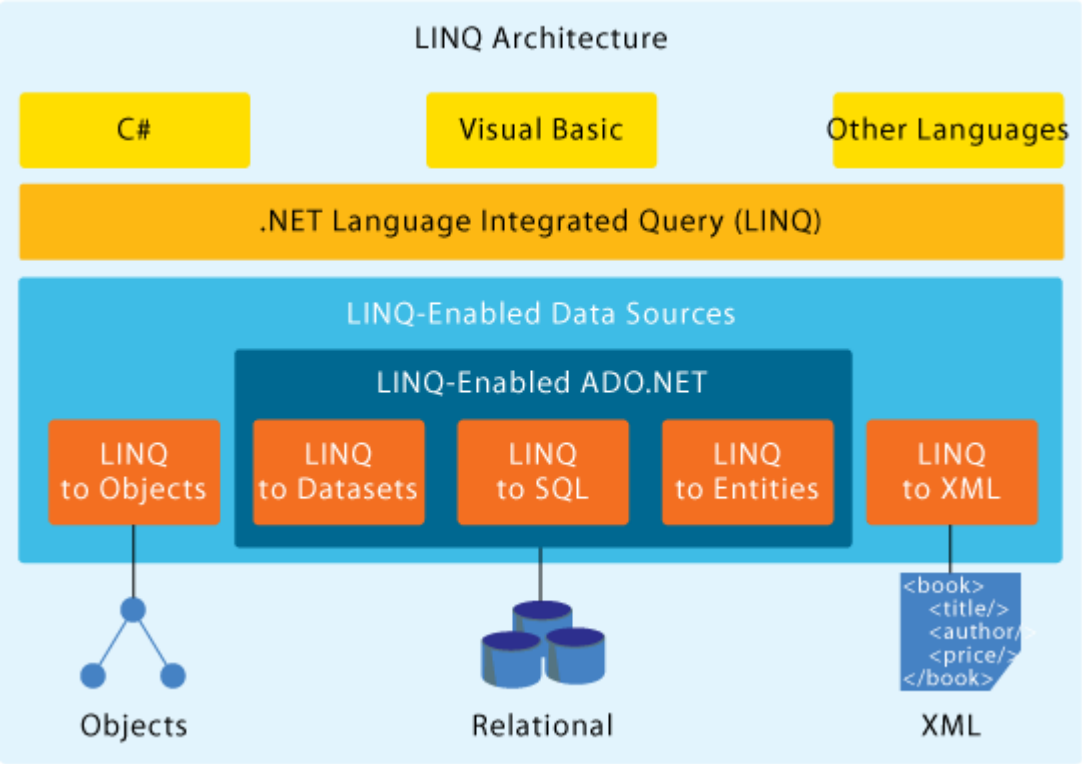
(1) (6)

1.7 LINQ

Language Integrated Query (LINQ) je programovací model pro práci s daty, který zavádí dotazy jako prvořadý princip do všech jazyků Microsoft .NET. Jedna z velkých výhod LINQu je možnost dotazovat se stejným způsobem nad různými

kolekcemi dat. Stejný dotaz lze použít nad kolekcí objektů, databází nebo XML dokumentem. Existuje několik různých implementací LINQ. (8)

- **LINQ to Objects** – pro dotazování se nad standardními kolekcemi nacházející se v paměti
- **LINQ to XML** – možnost využití LINQ dotazů nad XML soubory
- **LINQ to DataSet –** pro dotazování nad ADO.NET datasety
- **LINQ to SQL** – rozhraní pro komunikaci mezi objektovými daty a relačním datovým modelem, podpora pouze pro MS SQL Server
- **LINQ to Entities** – pro psaní dotazů v rámci konceptuálního modelu technologie .NET Entity Framework (8)



Obrázek 5: Architektura LINQ (18)

Na obrázku 5 je zobrazena třívrstvá architektura LINQ, ve které je nejvyšší vrstva tvořena jazykovými rozšířeními a spodní vrstva sestává ze zdrojů dat. Data, nad kterými jsou prováděny dotazy, mohou být ve formě XML *(LINQ to XML)*, formě relačních databází (*LINQ-enabled ADO.NET: LINQ to SQL, LINQ to DataSet a LINQ*

to Entities) a samozřejmě také ve formě objektů *(LINQ to Objects)*. LINQ je také vysoce rozšiřitelný a umožňuje vytvářet vlastní poskytovatele dat podporující LINQ (např. LINQ to Amazon, LINQ to NHibernate, LINQ to LDAP). (18)

1.7.1 Klíčová slova

LINQ umožňuje podporu dotazování přímo do .NET programovacího jazyku. Pomocí této technologie lze pracovat téměř s libovolnými daty prostřednictvím různých klíčových slov, podobně jako u klasického dotazování v SQL. Zde jsou uvedena některá z nich.

- **From** – definuje datový zdroj dotazu či poddotazu a proměnnou intervalu, jež určuje všechny jednotlivé elementy ve zdroji, na něž má dotaz směřovat
- **Where** – udává filtrovací podmínku, jež se aplikuje na datový zdroj
- **Select** – jde o projekci, která určuje, co se má vybírat z výsledků všech předchozích klauzulí a výrazů
- **GroupBy** – lze použít na seskupení výsledků podle určitého klíče
- **OrderBy, OrderByDescending** – umožňuje vzestupně či sestupně třídit výsledky dotazu
- **Join** – umožňuje spojovat různé datové zdroje na základě členů zdrojů, u nichž lze zjišťovat rovnost
- **Into** – pro ukládání výsledků příkazů *select*, *group* či *join* do dočasné proměnné
- **SelectMany** – umožňuje výběr více hodnot najednou (např. pole)
- **First, Last** – výběr prvního nebo posledního prvku z kolekce
- **Count** – počet prvků v kolekci
- **ElementAt** – výběr prvku podle udaného indexu
- **Count** – počet prvků v kolekci
- **Union, Intersect, Except** – definice množinových operací sjednocení, průnik a rozdíl
- **Sum, Min, Max, Average** – vrací součet, minimální, maximální či průměrnou hodnotu z dané kolekce
- **Concat** – spojí dvě kolekce téhož typu dohromady
- **Reverse** – vypisuje všechny položky ze zdrojové sekvence v opačném pořadí
- **OfType** – výběr pouze těch prvků, které jsou specifikovaného typu (8)

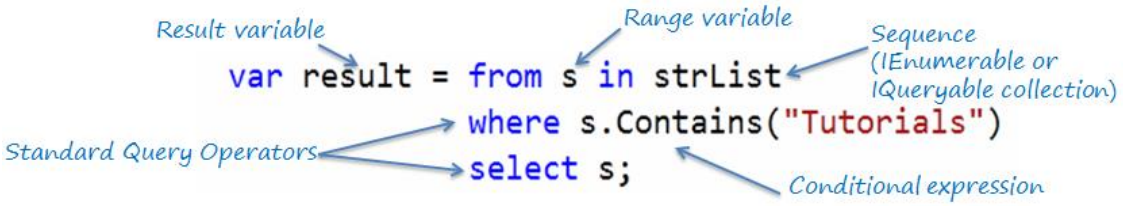
1.7.2 Syntaxe LINQ dotazu

Existují dva základní způsoby, jak psát LINQ dotazy a to *Query Syntax* a M*ethod Syntax or Fluent Syntax* (v češtině někdy označována také jako *výrazová*).

Query syntax je velice podobná SQL, začíná klíčovým slovem *from* a končí klíčovým slovem *select*. Následující ukázka jednoduchého LINQ dotazu vrací kolekci řetězců, které obsahují slovo *„Tutorials“*.

```
IList<string> stringList = new List<string>()
{
        "C# Tutorials",
        "VB.NET Tutorials",
        "Learn C++",
        "MVC Tutorials",
        "Java"
};

var result = from s in stringList
             where s.Contains("Tutorials")
             select s;
```



Obrázek 6: LINQ Query syntax (19)

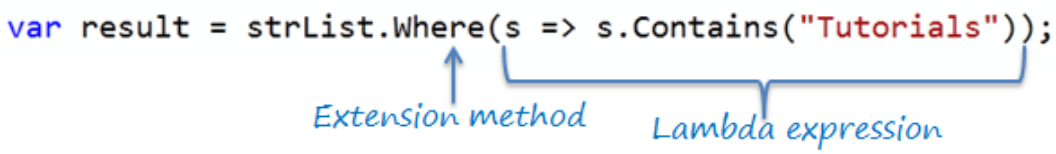
Dotazovaný výraz začíná klauzulí *from*, která deklaruje tzv. rozsahovou proměnnou (range variable). Klauzule *from* je strukturována jako *„from rangeVariableName in sequence“*. Za klauzulí *from* následuje v dotazu řada klauzulí, v nichž je možné používat nejrůznější dotazové operátory pro odfiltrování dat reprezentovaných rozsahovou proměnnou. V příkladu na obrázku je použit operátor *where*, za nímž následuje podmínka. Výraz je na konci uzavřen klauzulí *select*, která představuje operátor projekce. Když v dotazovaném výrazu provedeme projekci, je obvykle vytvořena jiná kolekce informací (či jediná informace), jež je transformovanou verzí kolekce procházené pomocí rozsahové proměnné. Je možné vybrat celé objekty jako takové nebo jen jejich vlastnosti. Ve výše uvedeném příkladu byly vybrány všechny výsledné prvky kolekce. (19)

Method syntax (známá také jako **fluent syntax**) používá tzv. rozšířené metody (extension methods) obsažené v *Enumerable* nebo *Queryable* statické třídě, která poskytuje sadu statických metod pro dotazování objektů implementujících rozhraní *IEnumerable<T>*. Toto rozhraní podporuje jednoduchou iteraci přes zadaný typ kolekce. (20)

Níže je ukázka LINQ dotazu v *method syntax*, který vrací kolekci řetězců, jež obsahuje slovo *"Tutorials"*.

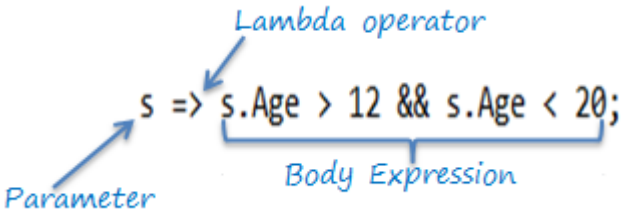
```
IList<string> stringList = new List<string>()
{
        "C# Tutorials",
        "VB.NET Tutorials",
        "Learn C++",
        "MVC Tutorials",
        "Java"
};

var result = stringList.Where(s => s.Contains("Tutorials"));
```



Obrázek 7: LINQ Method syntax (20)

Jak je vidět ve výše uvedeném obrázku, *Method syntax* zahrnuje **rozšířenou metodu** a **lambda výraz**. Rozšířená metoda *Where()* je definována ve třídě *Enumerable*. Chcete-li vytvořit lambda výraz, zadáte vstupní parametry na levé straně lambda operátoru => a zadáte výraz nebo blok příkazů na druhou stranu (obrázek 8). (20)



Obrázek 8: Struktura lambda výrazu (23)

1.8 Windows Communication Foundation

Windows Communication Foundation (WCF) je základním technologickým pilířem pro tvorbu distribuovaných aplikací orientovaných na služby využívající filozofie a principů servisně orientované architektury (SOA).

WCF představuje jednotný programovací model pro psaní aplikací typu klient-server. Zároveň nabízí obrovské možnosti ve výběru různých komunikačních protokolů pro komunikaci mezi aplikacemi. WCF nabízí jakési sjednocení, standardizování a univerzálnost psaní distribuovaných aplikací předem předurčených pro přenos informací. Propojení WCF s .NET je na opravdu vysoké úrovni, tyto technologie jsou velice úzce spjaty a zároveň se doplňují. WCF je spolu s dalšími zajímavými technologiemi k dispozici v .NET Frameworku ve verzi 3.0 a vyšší.

WCF byla navržena podle principů orientace na služby a sjednotila existující komunikační technologie jako .NET Remoting a webovou službu ASMX do jediného programovacího modelu a konzistentní architektury, která poskytuje vysokou úroveň funkčnosti, interoperability a rozšiřitelnosti. (21)

1.8.1 Endpoint

Typická služba WCF může vystavit jeden nebo více *endpointů* (koncových bodů). Obecně platí, že koncové body poskytují klientům přístup k funkčnosti nabízené službou WCF. Každý koncový bod se skládá ze tří vlastností, jak je znázorněno na následujícím obrázku:

Obrázek 9: Komunikace ve WCF (22)

- **Address** (adresa) - určuje, kde lze najít koncový bod
- **Binding** (vazba) - určuje, jak může klientská aplikace komunikovat s koncovým bodem
- **Contract** (kontrakt) - identifikuje operace vystavené koncovému bodu

Address

Každý endpoint musí mít vlastní adresu určující přesné místo, na kterém se služba nachází a kde je možné se s ní spojit. Tato adresa musí být vždy jedinečná, pak funguje jako jednoznačné URI (Uniform Resource Identifier) endpointu, neboli jako jednotný identifikátor zdroje.

Struktura stavby adresy je vcelku jednoduchá a v podstatě je nám tento způsob tvorby adresy známý třeba z webových stránek. Skládá se z definice transportního protokolu, jména počítače či serveru, portu a umístění, kde konkrétně na serveru služba běží. Například *http://localhost:8091/MojeSluzba*. (21)

Binding

Binding (vazba) je definice způsobu komunikace, zabezpečení přenosu a šifrování dat. WCF disponuje celou řadou již předdefinovaných bindingů, které v naprosté většině případů bohatě postačují nárokům vývojáře.

Příklady bindingů implementovaných přímo v .NET Frameworku: *BasicHttpBinding, WSHttpBinding, WebHttpBinding, CustomBinding atd.* (21)

Kontrakty

Kontrakt definuje funkce nabízené službou a funkce, které může použít klient. Může být na implementaci služby zcela nezávislý. WCF disponuje nejrůznějšími typy kontraktů definované pomocí atributů, které mohou obsahovat různá další nastavení pomocí parametrů.

Kontrakty definované ve WCF se dělí do tří základních typů:

- **Datový kontrakt** – definuje data, která jsou službou přijímána a vracena. Atribut *[DataContract]* se píše před hlavičku definující třídu, která bude využita

pro přenos. Atribut *[DataMember]* se píše před jednotlivé členy, které struktura obsahuje.

```
[DataContract]
public class Customer
{
	[DataMember]
	public string fullName;

	[DataMember]
	public string phoneNumber;
}
```

- **Kontrakt služby** - definuje operace, které může služba provést. Atribut *[ServiceContract]* se používá v rozhraních nebo třídách, když je třeba definovat kontrakt služby. Na metody nabízené takovou službou se aplikuje atribut *[OperationContract]*.

```
[ServiceContract]
public interface IService1
{
	[OperationContract]
	string GetData(string value);

	[OperationContract]
	void Delete(int code);
}
```

- **Kontrakt zprávy** – pokud je potřeba mít nad zprávou SOAP plnou kontrolu, atribut *[MessageContract]* určuje kontrakt zprávy, hlavička a tělo zprávy SOAP označujeme atributy *[MessageHeader]* a *[MessageBodyMember]*.
(21)

1.8.2 WCF služba

WCF vyžaduje pro vytvoření webové služby minimálně dva typy objektů. Prvním typem je třída, která obsahuje implementaci všech metod, které mají být reprezentovány jako metody webové služby.

```
public class Service : IService
{
	public string GetData(string value)
	{
		return string.Format("You entered: {0}", value);
	}
	...
}
```

Dalším typem je rozhraní s jednoduchým nastavením, jež má zmíněná třída implementovat.

```
[ServiceContract]
public interface IService
{
	[OperationContract]
	string GetData(string value);
	...
}
```

Vedle třídy a rozhraní, které definují strukturu každé webové služby, musí být nastaveny i další vlastnosti, které závisí na konkrétních požadavcích na výslednou webovou službu. Mezi tyto požadavky patří mimo jiné nastavení koncových bodů, které definují například protokoly, prostřednictvím kterých bude probíhat komunikace. Dalšími důležitým nastavením je bezpečnostní politika, kterou je možné definovat pomocí konkrétních způsobů autentizace a autorizace. (21)

1.9 Unit testování

Unit testování se zaměřuje na relativně malé částí kódu (tzv. units, jednotky), aby se prokázalo, že tyto jednotky fungují správně. Je prováděno většinou přímo vývojáři, kteří se podílí na sestavení dané dílčí části. Úkolem unit testů je kontrola všech možných průchodů a správná reakce na uměle vytvořené případy, které by mohly v ostrém provozu aplikace nastat. Nevýhodou této metody je neschopnost detekování integračních chyb nebo rozsáhlejších chyb v systému. (11)

2 ANALÝZA SOUČASNÉHO STAVU

V této kapitole je nejprve stručně představena společnost ANeT-Advanced Network Technology, s. r. o. Dále je provedena analýza současného nevyhovujícího stavu správy akcí v dané společnosti, na základě které jsou stanoveny požadavky na nové řešení. Taktéž jsou představeny softwarové nástroje pro objektově relační mapování, jež se v současnosti nejvíce využívají ve vývoji na .NET platformě, z nichž je vhodně vybrán jeden, vyhovující dané situaci.

2.1 Představení společnosti

Společnost ANeT-Advanced Network Technology, s. r. o. poskytuje komplexní služby v oblasti identifikačních systémů od návrhu až po vlastní realizaci. Firma ANeT je předním dodavatelem identifikačních systémů pro podnikatelskou i státní sféru v České a Slovenské republice.

Obrázek 10: Logo společnosti ANeT (24)

Firma byla založena roku 1993. Dynamický rozvoj firmy vedl v roce 2003 k zavedení systému jakosti podle ISO 9001:2001. Špičková technologie, komplexní výroba, specializovaný tým pracovníků na vysoké profesionální úrovni. Tyto předpoklady umožňují vytvářet hodnotné a kvalitní produkty. Komponenty vyráběné a dodávané společností ANeT jsou v souladu s protokolem o shodě a jsou certifikovány NBÚ na stupeň přísně tajné. Vlastní vývojové, výrobní i realizační kapacity umožňují garantovat vysokou kvalitu prováděných prací a poskytovat trvalý záruční i pozáruční servis na realizované projekty. (24)

2.1.1 Produkty

Systém *ANeT−Work* slouží pro automatický sběr dat ve výrobě a monitoruje plnění zakázek. Umožňuje přidělování zakázek jednotlivým pracovníkům, sleduje čas strávený na jednotlivých zakázkách a stav jejich dokončení. Získáte přehled o stavu

prací a snadno optimalizujete zdroje, čímž dosáhnete zvýšení efektivity práce. Systém je možno přímo propojit s ERP systémem.

Systém *ANeT−Plan* je moderní nástroj pro plánování lidských zdrojů. Pomocí tohoto systému efektivně naplánujete směny, absence a pohotovostí pro Vaše pracovníky. Systém velmi úzce spolupracuje s docházkovým systémem ANeT-Time a tak zajišťuje vazbu mezi plánováním a realitou.

Docházkový systém *ANeT−Time* je primárně určen k evidenci odpracované doby, nahradí tedy ruční způsoby evidence docházky. Získáte rychlé, přesné a automatické vyhodnocení odpracované doby a příplatků v návaznosti na nastavení (např. směnný provoz). Pomáhá při kontrole dodržování legislativy (zákoník práce). ANeT-Time je komplexní vstupní systém pro přípravu dat pro mzdové a vyšší informační systémy.

Stravovací systém *ANeT-Kredit* nahrazuje doposud užívané způsoby prodeje a výdeje jídel a doplňkového sortimentu. Platby za jídlo a sortiment realizujete bezhotovostně odečtem ze stravovacího účtu, srážkou ze mzdy, bankovním převodem nebo platbou v hotovosti.

Systém kontroly vstupu ANeT−Guard spolehlivě sleduje a řídí (časově i prostorově) průchody jednotlivých osob prostřednictvím snímačů a přístupových mechanismů (dveře, turniket, závora, branka apod.). Pomocí tohoto systému ochráníte hmotný i nehmotný majetek firmy, získáte přehled o pohybu osob a kontrolu přístupu ke chráněným informacím. (24)

2.2 Současný stav správy akcí

Společnost v současnosti nemá žádný systematický postup, kterým by pokrývala oblast správy firemních akcí. Informace o těchto akcích (ať už sportovních či kulturních) jsou ukládány do souborů tabulkového procesoru EXCEL, jež jsou k dispozici všem zaměstnancům na firemním disku, či jsou vyvěšeny na nástěnkách ve společnosti.

Soubory EXCELu jsou nazvány dle druhů akcí (fotbal, tenis, koncerty, oslavy, bowling apod.). Jednotlivé soubory spravují pověření zaměstnanci, kteří mají na starosti danou zájmovou oblast, a dle počtu potvrzených účastníků poté rezervují konkrétní prostory na daný termín konání.

Tento způsob má ovšem mnoho nevýhod, mezi hlavní určitě patří možnost změny souborů a tím i celistvosti informací o plánovaných akcích kterýmkoliv zaměstnancem. „Správci“ souborů tak nemají nad soubory plnou kontrolu. Dalším problémem je, že zaměstnanci nemají jednoduchý přehled o tom, na které akce již potvrdili či nepotvrdili účast. Jsou nuceni procházet tabulkové soubory a u každého termínu akce kontrolovat seznam účastníků, což je značně neefektivní. Další nevýhodou je také to, že již proběhlé akce se ze souborů mažou, tím pádem není uchovávána žádná historie o firemních akcích ani o tom, kteří zaměstnanci se jich zúčastnili. To znamená, že data není možné ani jakkoliv analyzovat a vyhodnocovat.

Z důvodu nevyhovujícího současného stavu vznikl ve společnosti požadavek na vytvoření databázového systému pro správu firemních akcí. Tento systém bude uchovávat informace týkající se firemních akcí a zaměstnancům společnosti bude tyto informace poskytovat a bude jim umožňovat se na tyto akce přihlašovat či se z nich odhlašovat. Zaměstnanci společnosti budou také moci k jednotlivým akcím přidávat příspěvky, čímž by se měla zlepšit komunikace a v konečném důsledku také zpětná vazba. Vedení společnosti bude taktéž moci analyzovat a vyhodnocovat proběhlé akce. Systém bude vytvořen s podporou .NET Frameworku a dle požadavku zadavatele bude založen na databázovém serveru MS SQL Server, kterým společnost disponuje.

2.3 Funkční požadavky

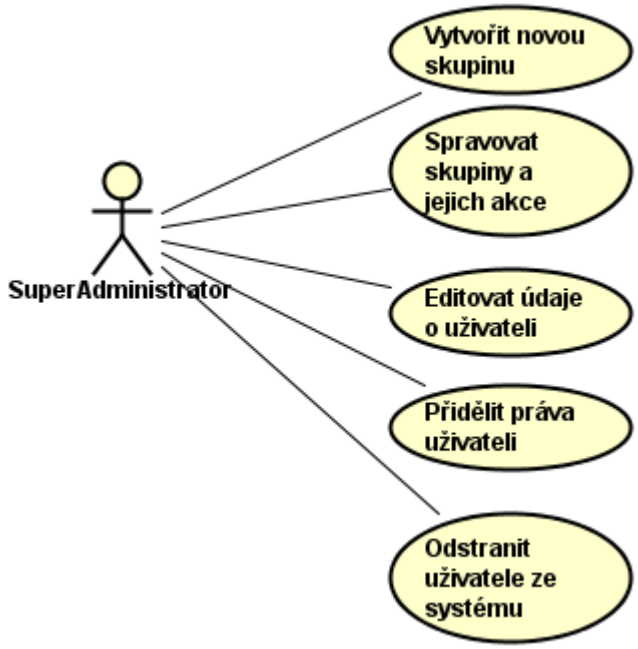
Systém pro správu akcí by měl propojit informace o jednotlivých zaregistrovaných uživatelích, kteří mohou patřit do jedné nebo více skupin, s informacemi o jednotlivých sériích akcí pro tyto skupiny.

Každý uživatel, patřící do nějaké skupiny se bude mít možnost přihlásit/odhlásit z akce, nebo může delegovat tuto činnost i na jiné uživatele.

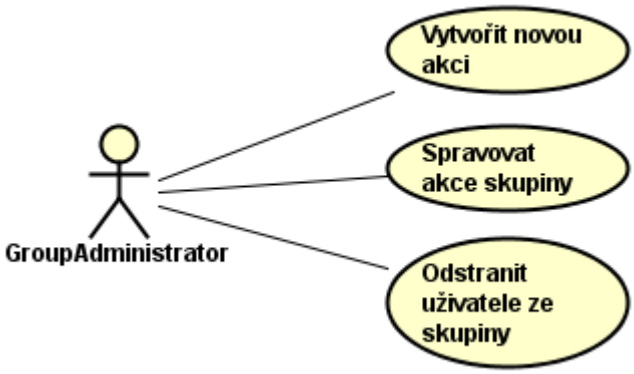
Uživatelé budou také patřit do různých rolí a tím jim bude také umožněno provádět různé činnosti, jako např. *SuperAdministrator* bude mít možnost definovat nové skupiny a jejich akce, rušit uživatele apod., *GroupAdministrator* bude mít možnost spravovat jen akce a další konfigurace dané skupiny a pak *User* s nejnižšími právy, což je jen přihlašování a odhlašování.

2.3.1 Use case diagramy

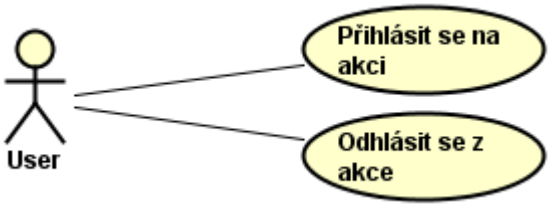
Na základě přiděleného oprávnění mají jednotliví uživatelé pravomoc na provádění určitých činností. To zobrazují níže uvedené diagramy případů užití (use case diagramy).



Obrázek 11: Use case diagram pro roli SuperAdministrator (vlastní zpracování)



Obrázek 12: Use case diagram pro roli GroupAdministrator (vlastní zpracování)



Obrázek 13: Use case diagram pro roli User (vlastní zpracování)

2.4 Softwarové nástroje pro objektově relační mapování

Požadavkem zadavatele bylo, aby jako databázový server byl zvolen Microsoft SQL Server. Z hlediska dat zbývá vyřešit jediné – jaký model pro interakci s databází použít. Na řešení tohoto problému poskytuje .NET Framework mnoho technologií. Za účelem vhodného výběru pro realizaci databázového systému budou v následující části představeny tři objektově-relační frameworky, které se při vývoji v .NET Frameworku využívají v posledních letech nejčastěji.

2.4.1 NHibernate

NHibernate je řešení poskytované pod licencí GNU Lesser General Public License. Je vyvíjené od roku 2006. Aktuální stabilní verze má pořadové číslo 4.1.1 a poskytuje i částečnou podporu dotazovacího jazyku LINQ.

NHibernate se vyvíjel dávno před dobami jazyka LINQ, má tudíž několik jiných způsobů, jak vytvářet dotazy. V podpoře LINQ má určité slabiny (jeho provider LINQ to NHibernate nepodporuje např. klauzuli order by, left join apod.) NHibernate proto definuje vlastní dotazovací jazyk, který je možné použít v situacích, kdy LINQ nebo

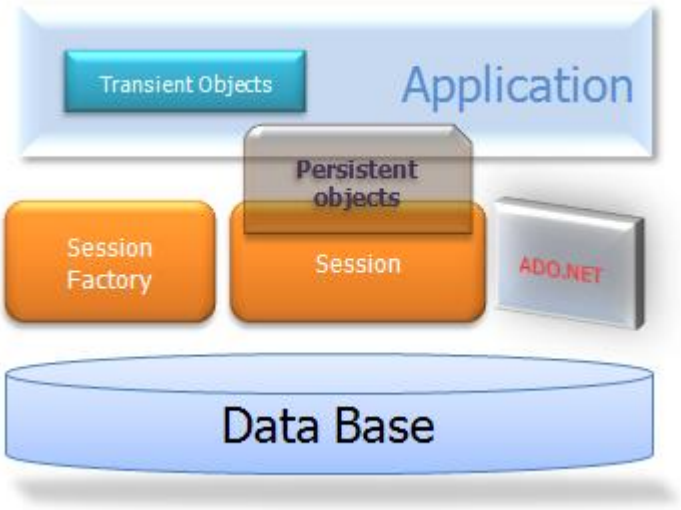
jiné přístupy nestačí. Konkrétně jde o dotazovací jazyk zvaný HQL, který se používá především pro psaní složitých dotazů. Syntakticky jde velmi zjednodušeně o dialekt jazyka SQL, který navíc umožňuje pracovat s entitami daného frameworku.

NHibernate nabízí ještě dva další dotazovací mechanismy – Criteria API a Query Over. Criteria API je staré API s kompilovanou syntaxí a bez statické kontroly při překladu. Mechanismus QueryOver je silný nástroj využívající lambda funkce. Na druhou stranu bývá NHibernate terčem kritiky za přílišné množství dotazovacích mechanismů, kdy není schopen kvalitně podporovat standardní LINQ a místo toho přidává nové API.

NHibernate podporuje MS SQL Server, Oracle, DB2, MySQL, SQLite, PostgreSQL a Firebird.

Dokumentace NHibernate má své slabé stránky a není zcela vyčerpávající, může se stát, že podrobný popis určité funkcionality tam nenajdeme a budeme muset hledat porůznu na webu.

Další nevýhodou je, že neexistuje podpora NHibernate ve Visual Studiu, ale je potřeba pořídit některý z placených designerů (např. Mindscape NHibernate Designer, Devart NHibernate Designer). (25)



Obrázek 14: Architektura NHibernate (26)

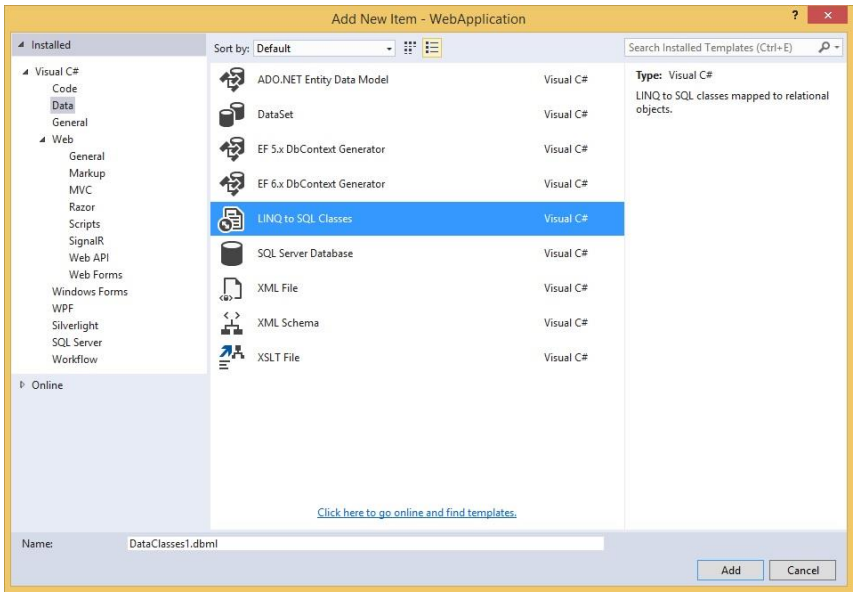
2.4.2 LINQ to SQL

LINQ to SQL je komponentou .NET Frameworku, která poskytuje infrastrukturu pro správu relačních dat jako objektů.

V LINQ to SQL je datový model relační databáze mapován na objektový model vyjádřený v objektově orientovaném programovacím jazyce. LINQ to SQL překládá při běhu aplikace LINQ dotazy v objektovém modelu do SQL jazyka a posílá je do databáze k vykonání. Pokud databáze vrátí výsledky, tak je LINQ to SQL převede zpět na objekty, s kterými je možné pracovat v daném programovacím jazyce.

Vývojáři využívající Visual Studio obvykle používají Object Relation Designer, který poskytuje uživatelské rozhraní pro implementaci mnoha funkcí LINQ to SQL.

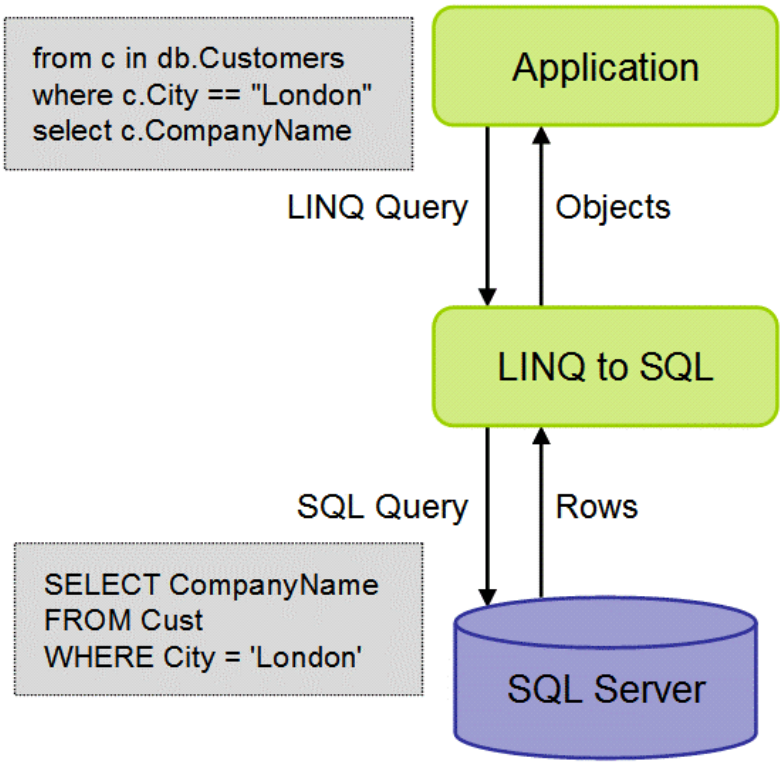
Pro použití LINQ to SQL v aplikaci je potřeba vytvořit soubor DBML, který bude obsahovat zdrojový kód, jež nám umožní psát LINQ dotazy pro získání dat z SQL databáze. Stačí do projektu ve Visual Studiu přidat novou položku LINQ to SQL Classes. (27)



Obrázek 15: Přidání LINQ to SQL do projektu ve Visula Studiu (vlastní zpracování)

U LINQ to SQL je vždy mapována jedna tabulka na jeden objekt. Nad databází vzniká tedy silně typovaná vrstva, jedna ku jedné. U tohoto mapování tedy neexistuje nic jako konceptuální model.

V případě LINQ to SQL nám však může jakákoliv změna v databázi způsobit nefunkčnost celé aplikace. Avšak kombinace zmíněného mapování a Microsoft SQL Serveru nabízí mnoho automaticky generovaných funkčností, které by bylo např. při použití NHibernate potřeba doprogramovat. (27)

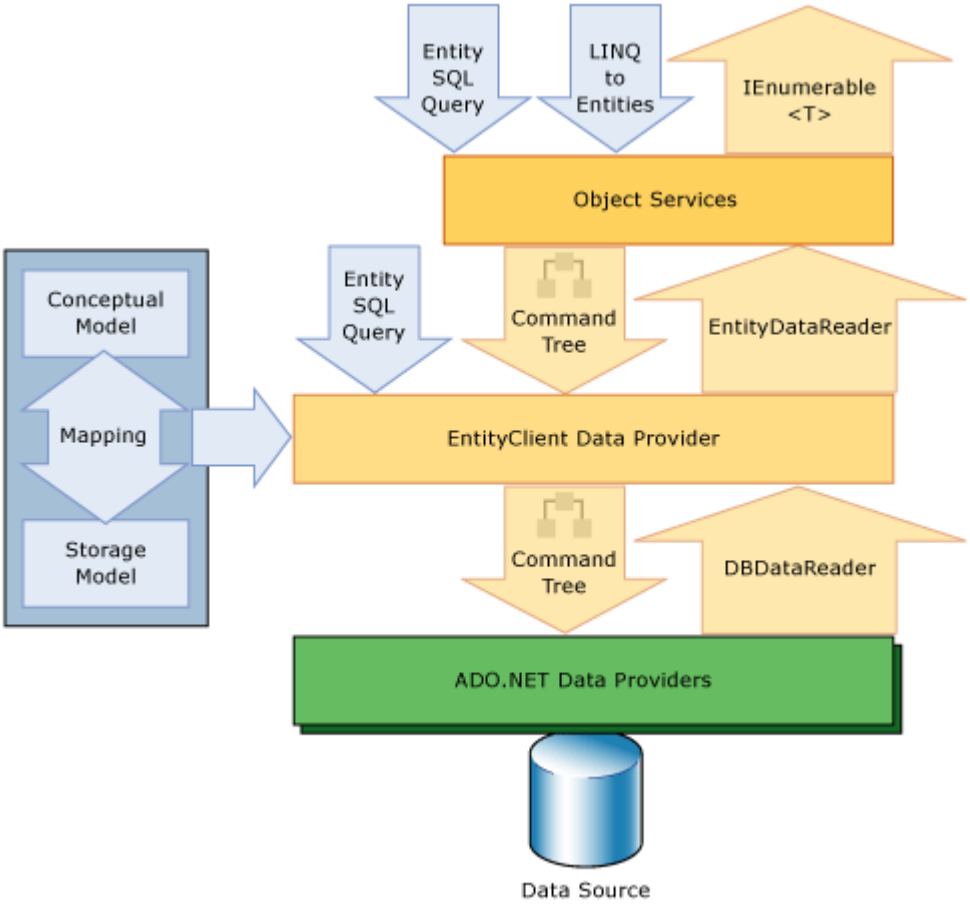


Obrázek 16: Architektura LINQ to SQL (28)

2.4.3 .NET Entity Framework

.NET Entity Framework (EF) je open-source ORM nástroj součástí .NET Frameworku. Posouvá způsob práce s daty databáze na vyšší úroveň abstrakce než doposud. Pokud potřebujeme získat data z databáze, nedotazujeme se přímo, ale datového modelu entit (entity nejsou objekty, ale definice objetků), který vrací objekty. Jakmile potřebujeme uložit provedené změny, uložíme tyto objekty. Entity Framework provede konverzi objektů do tabulek databáze. Využívá Entity Data Model (EDM) – model vyvinutý z ER modelu. (29)

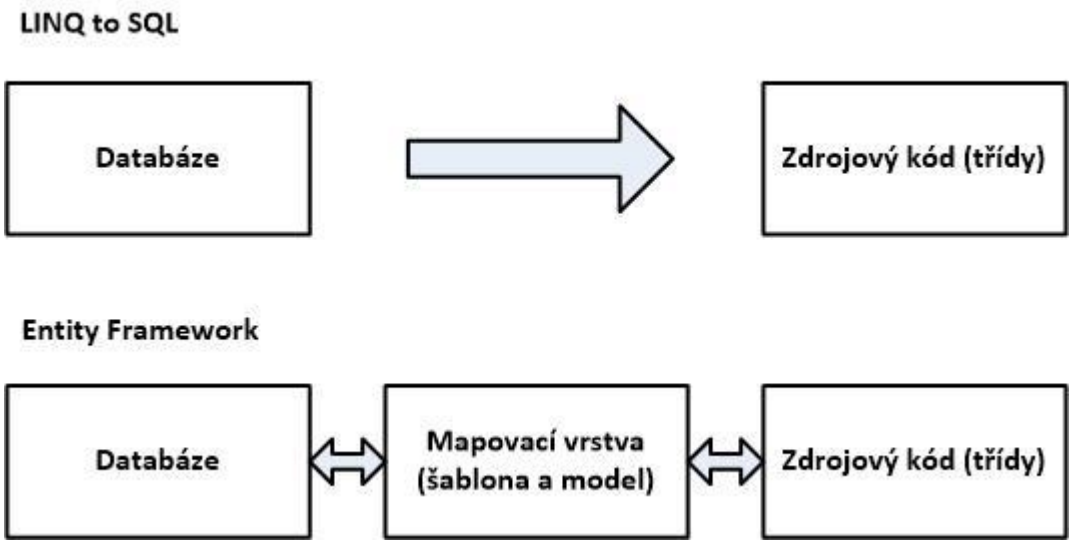
Architektura tohoto systému je zobrazena na obrázku 17.



Obrázek 17: Architektura Entity Framework (15)

EDM je klíčová část Entity Frameworku definující entity a vztahy. Není identická s modelem databáze, ale popisuje strukturu objektů na business vrstvě. EDM je v Entity Frameworku reprezentován EDMX souborem. Visual Studio 2008 SP1 a vyšší poskytuje nástroje pro generování EDM z fyzické databáze. Možný je i opačný postup, na základě EDM vygenerovat databázi. (29)

Oproti LINQ to SQL Entity Framework negeneruje zdrojový kód z databáze, ale zavádí mapovací vrstvu, která je složena z šablony pro generování kódu a modelu databáze, viz obrázek 18.

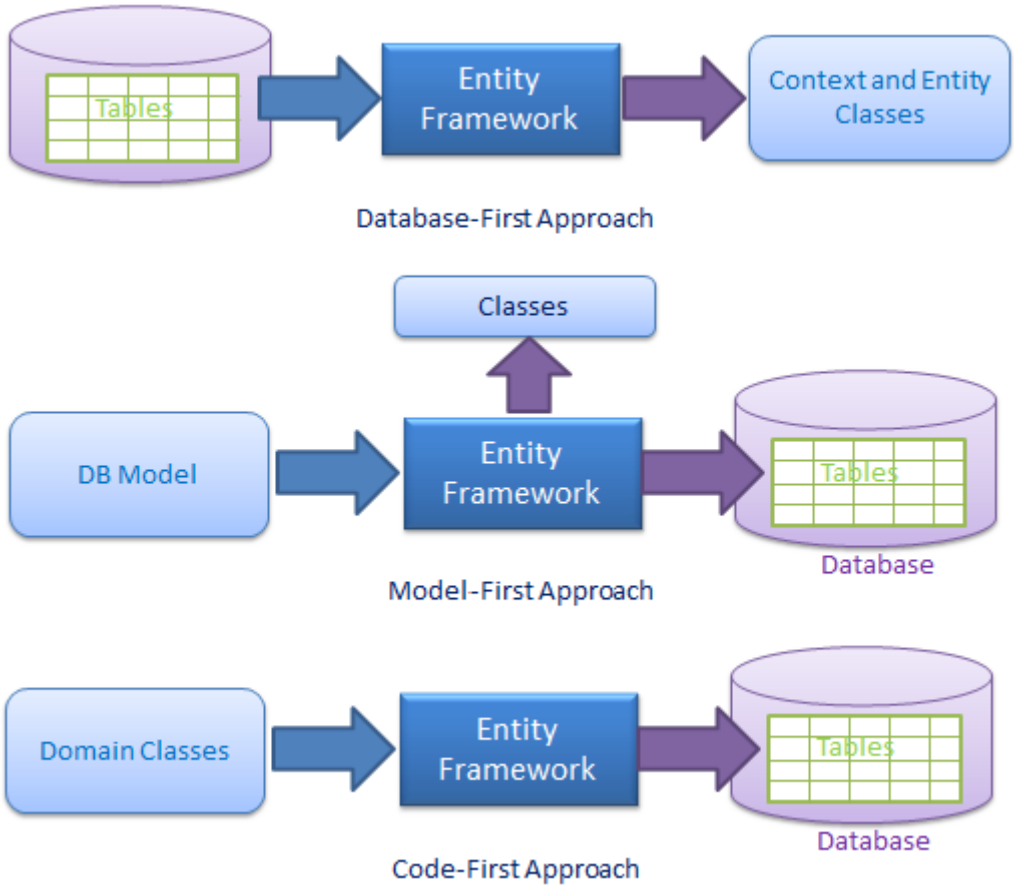


Obrázek 18: LINQ to SQL vs. Entity Framework (vlastní zpracování)

Další výhodou mapovací vrstvy je generování code-first a model-first (popsány v další části).

Entity Framework podporuje tři různé vývojové přístupy k použití Entity Frameworku v aplikacích.

- **Database First** – vygenerování EDM a tříd podle struktury existující databáze
- **Model First** – vytvoření EDM přímo v designeru, z EDM je vygenerována databáze a třídy
- **Code First** – napsání tříd programovacího jazyka dle návrhu doménového modelu a automatické vygenerování databáze



Obrázek 19: Database-First, Model-First a Code-First přístupy (29)

Shrnutí objektově relačních frameworků

Hlavním rizikem NHibernate je jeho podpora do budoucna, implementace funkcí z nových verzí .NET frameworku se výrazně zpomalila a snížil se počet vývojářů. Použití některých standardních vlastností je v NHibernate o něco pracnější než v LINQ to SQL nebo Entity Frameworku, slabinou NHibernate je také dokumentace a absence designeru entit, který je nutné dokoupit.

Co se týče LINQ to SQL, tak jde o nejjednodušší a zároveň o nejrychlejší z vybraných objektově-relačních frameworků. Microsoft původně zamýšlel použít LINQ to SQL pouze jako součást některých interních projektů. Proto LINQ to SQL nedisponuje bohatou funkcionalitou. U LINQ to SQL se oproti Entity Frameworku nebo NHibernate nevyskytuje žádná mapovací vrstva. Na druhou stranu oproti NHibernate obsahuje designer, který je součástí Visual Studia.

Entity Framework se používá velmi podobně jako LINQ to SQL, má také svůj designer, který je integrován ve Visual Studiu. Nabízí však bohatší funkcionalitu – především propracovanější mapování, možnost použití model-first či code-first přístupu. Vývoj LINQ to SQL je v současnosti již uzavřen, Microsoft svoji pozornost soustřeďuje na Entity Framework a pravidelně přidává novou funkcionalitu.

Vzhledem k výše uvedenému a taktéž k požadavku zadavatele na uložení dat na databázovém serveru MS SQL Server je pro potřeby této práce zvolen .NET Entity Framework.

3 VLASTNÍ NÁVRHY ŘEŠENÍ

Na základě analýzy bude v této části vytvořen návrh řešení databázového systému vzhledem k daným požadavkům zadavatele.

3.1 Postup řešení

Nejprve bude zpracován návrh databázové struktury, kde budou identifikovány všechny entity a vazby mezi nimi, poté bude vytvořen ER diagram a realizována samotná databáze na databázovém serveru MS SQL Server.

Dále již půjde o práci ve vývojovém prostředí MS Visual Studio, kde bude nejprve potřeba provázat data mezi relační databází na MS SQL Server a objekty v objektově orientovaném jazyce C# prostřednictvím .NET Entity Framework. Budou provedeny dvě strategie přístupu, a to *Database first* a *Model first*. Výsledky obou přístupů porovnány a vybrán jeden, který bude následně použit při implementaci WCF služby. Funkčnost této služby bude nakonec otestována.

3.2 Návrh databáze

Návrh databáze zahrnuje identifikaci všech entit (relačních tabulek) a vazeb mezi nimi, vytvoření datového modelu a implementace databáze do cílového DBMS. Vzhledem k následnému využití jazyka C# jsou pro pojmenování jednotlivých entit použity anglické výrazy.

3.2.1 Identifikace entit

Následující tabulka identifikuje všechny entit, které je potřeba uchovávat v databázi.

Název entity	**Popis**
TUser	Konkrétní zaměstnanec ve společnosti
TRole	Role uživatele dle pravomocí uživatelů
TGroup	Zájmová skupina firemních akcí
TEvent	Informace o konkrétním termínu firemní akce
TPost	Příspěvek uživatele ke konkrétní akci

TReg_current_status	Záznam o současném stavu registrace uživatele
TReg_log	Záznam o každých změnách registrací uživatele (logovací tabulka)
TReg_type	Typ jednotlivých registračních záznamů uživatele
TEvent_type	Typ jednotlivých akcí
TUser_delegated	Informace o uživatelích, na které je delegována pravomoc od jiného uživatele

Tabulka 1: Entity (vlastní zpracování)

3.2.2 Identifikace vztahů

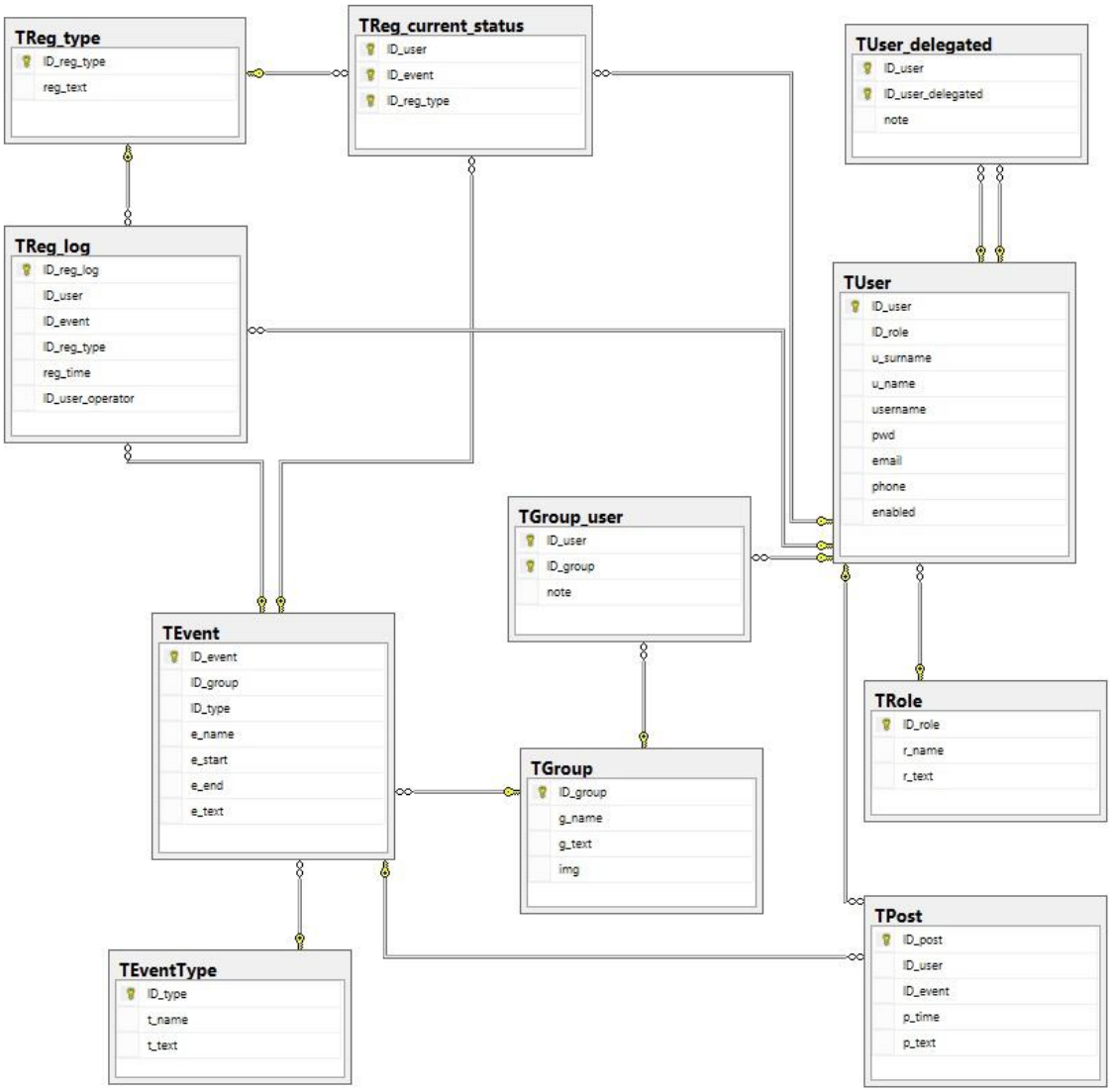
Níže uvedená tabulka identifikuje vztahy mezi entitami.

Entity	Vztah
TUser – TPost	1:N
TUser – TGroup	M:N
TGroup – TEvent	1:N
TEvent – TReg_current_status	1:N
TEvent – TReg_log	1:N
TEvent_type – TEvent	1:N
TRole – TUser	1:N
TUser – TReg_current_status	1:N
TUser – TReg_log	1:N
TUser – TUser_delegated	1:N
TReg_type – TReg_log	1:N
TReg_type - TReg_current_status	1:N

Tabulka 2: Entitní vztahy (vlastní zpracování)

3.2.3 Datový model

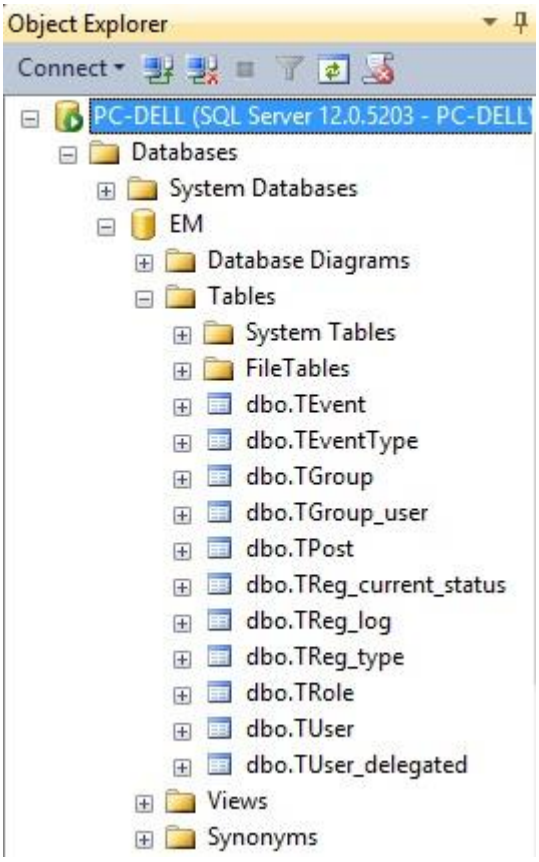
Na základě identifikovaných entit a jejich vztahů byl vypracován entitně-relační diagram (ER diagram), kde jsou již doplněny atributy jednotlivých relačních tabulek. Rovněž byla provedena dekompozice vztahu M:N u tabulek *TUser* a *TGroup*, čímž vznikla další tabulka *TGroup_user*.



Obrázek 20: ER diagram (vlastní zpracování)

3.2.4 Realizace databáze

V této části je realizována samotná databáze na databázovém serveru MS SQL Server 2012. K tomuto byl využit nástroj MS SQL Management Studio. Zdrojový kód (skript) v jazyce SQL pro vytvoření tabulek se vzhledem k jeho obsáhlosti nachází na přiloženém disku, na němž se nachází taktéž záloha databáze.



Obrázek 21: Struktura databáze EM (vlastní zpracování)

3.3 Objektově relační mapování

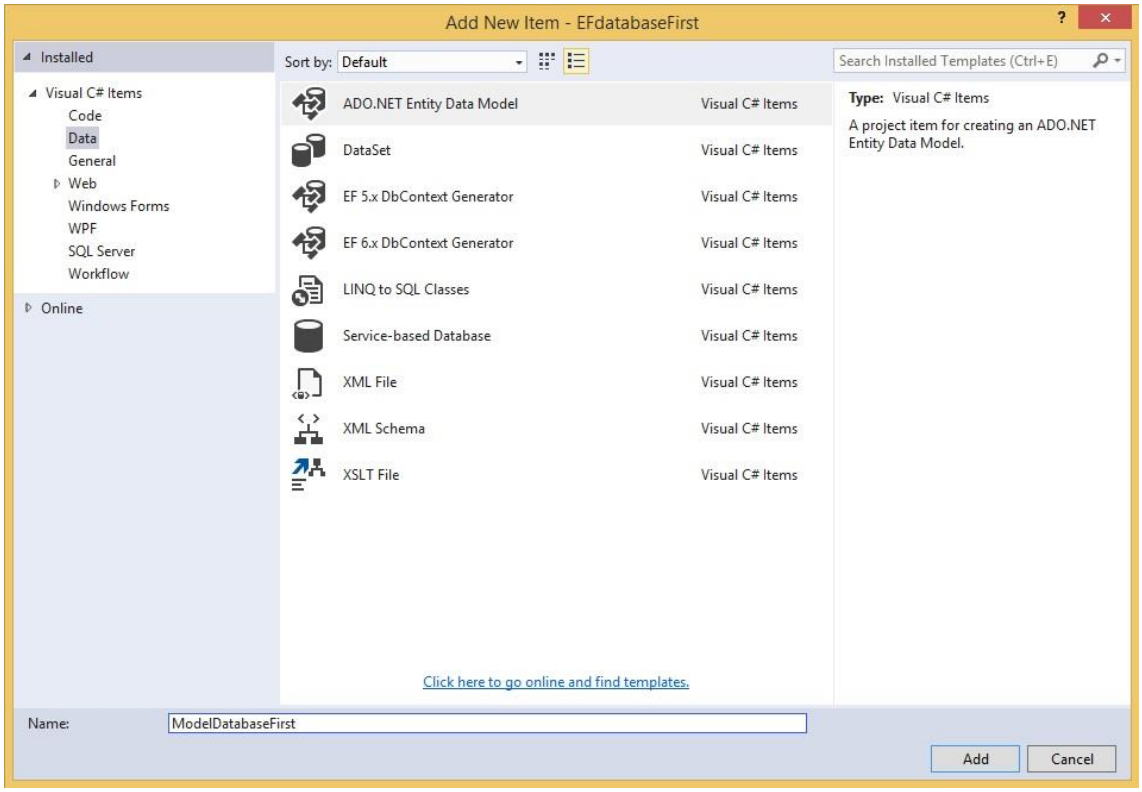
Pro další části databázového systému již je použit objektově orientovaný jazyk C#. Propojení mezi objektově orientovanými jazyky a relační databází zajišťují objektově relační frameworky. V rámci tento práce byl zvolen .NET Entity Framework. V kapitole 3.1.2 jsou představeny tři různé přístupy vývoje k použití .NET Entity Framework - *Database first*, *Model first* a *Code first*. V následující kapitole jsou provedeny dvě strategie, a to *Database first* a *Model first*.

Vzhledem k tomu, že je již vytvořena databáze na SQL Serveru, nabízí se provést strategii Database first. Nicméně je provedena i strategie Model first pro potřeby srovnání. Dále při implementaci serverové části aplikace je použit již jeden přístup, který více vyhovuje dané situaci. K vývoji je využito Visual Studio 2015 ve verzi Professional a .NET Entity Framework ve verzi 6.

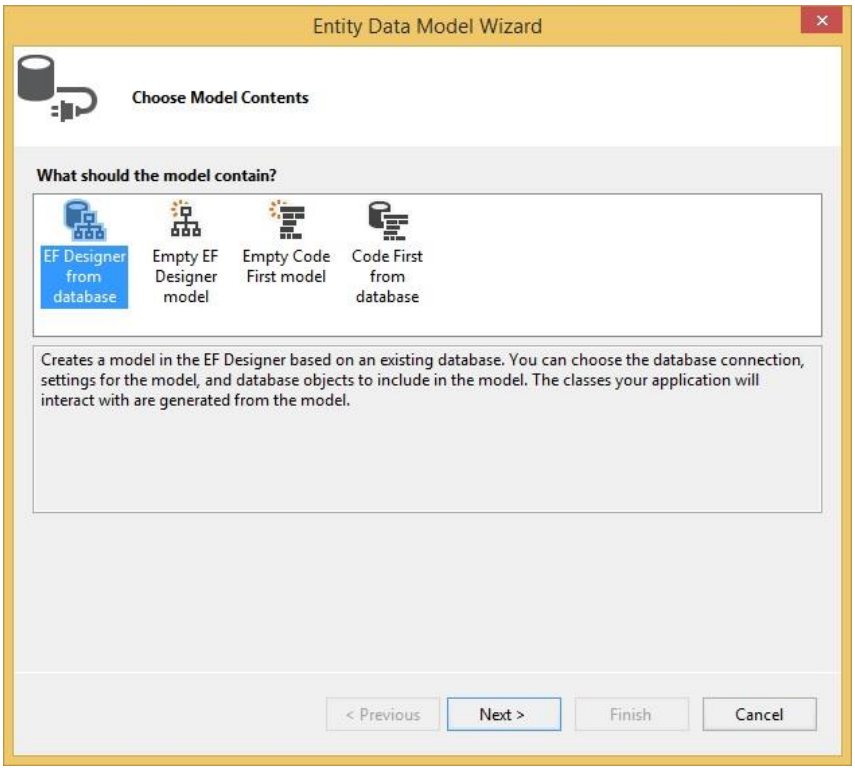
3.3.1 Database first přístup

Strategie „database first“ je vytvoření Entity Data Modelu (EDM) podle struktury existující databáze. Vývojové prostředí Visual Studio 2015 obsahuje průvodce, ve kterém se lze jednoduše připojit k databázi a zvolit databázové objekty k vytvoření EDM. Těmito objekty mohou být tabulky, pohledy i uložené procedury. Entity Framework poté z modelu vygeneruje třídy, včetně jejich vlastností a závislostí, jež jsou patrné z definic tabulek a klíčů mezi nimi.

Pro Database first přístup byl vytvořen projekt typu *Class Library* s názvem *EFdatabaseFirst*, do kterého byl přidán *ADO.NET Entity Data Model* (obrázek 22) a následně zvolena možnost *EF Designer from database* (obrázek 23), kde byly poté specifikovány údaje potřebné k připojení do databáze EM, jež byla vytvořena v kapitole 3.2.4, a také vybrány všechny databázové objekty, které je potřeba do modelu zahrnout.

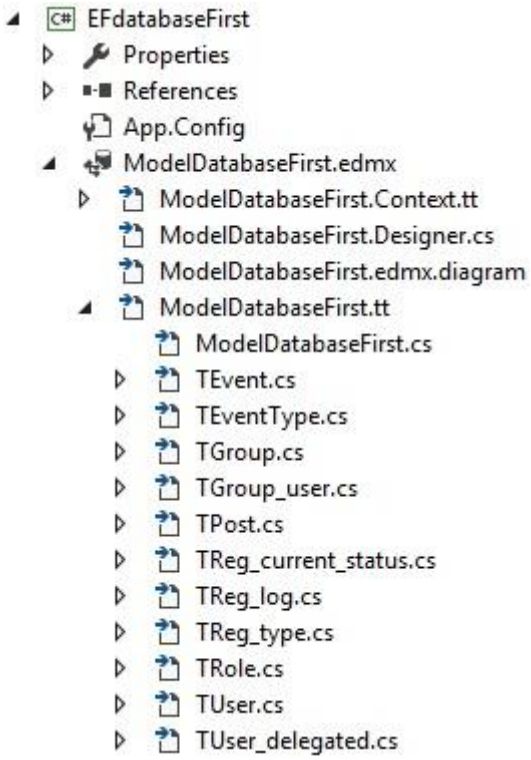


Obrázek 22: Přidání ADO.NET EDM do projektu EFdatabaseFirst (vlastní zpracování)



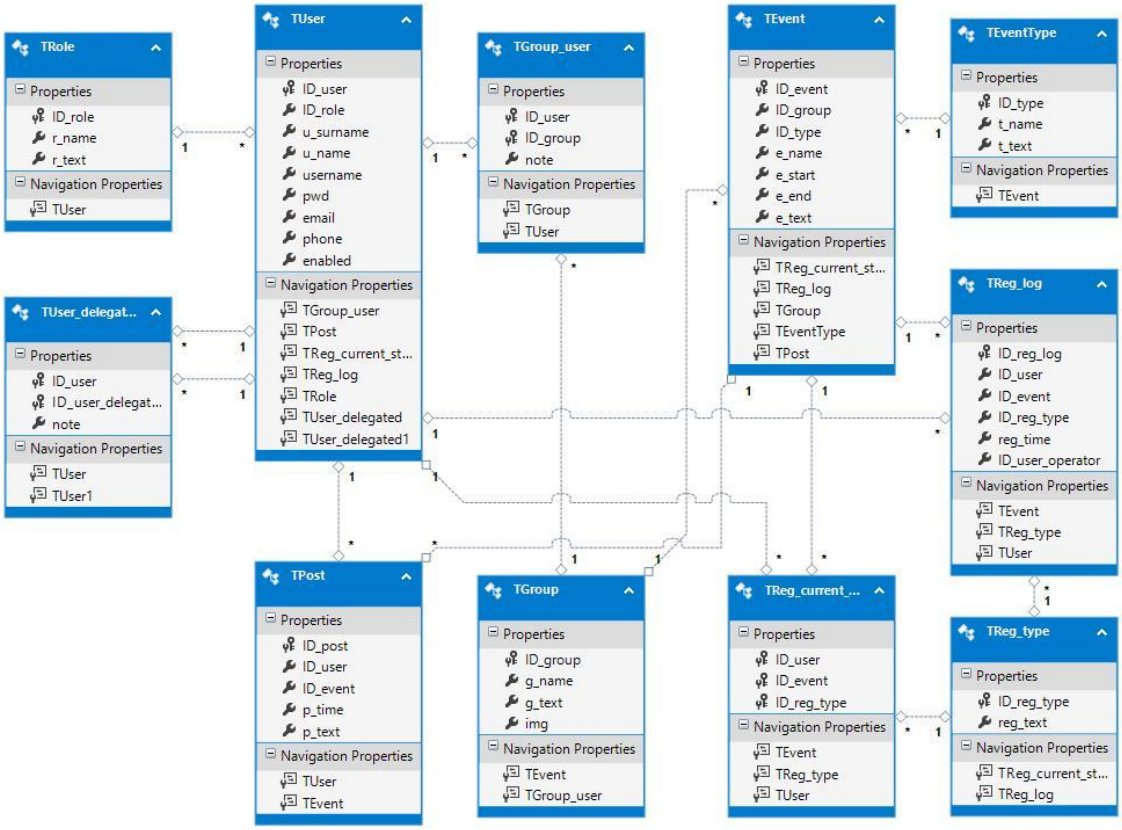
Obrázek 23: EF Designer from database (vlastní zpracování)

Struktura projektu *EFdatabaseFirst* se nachází na obrázku 24.



Obrázek 24: Struktura projektu EFdatabaseFirst (vlastní zpracování)

Obrázek 25 představuje konečný Entity Data Model, který byl vygenerován prostřednictvím strategie Database first.

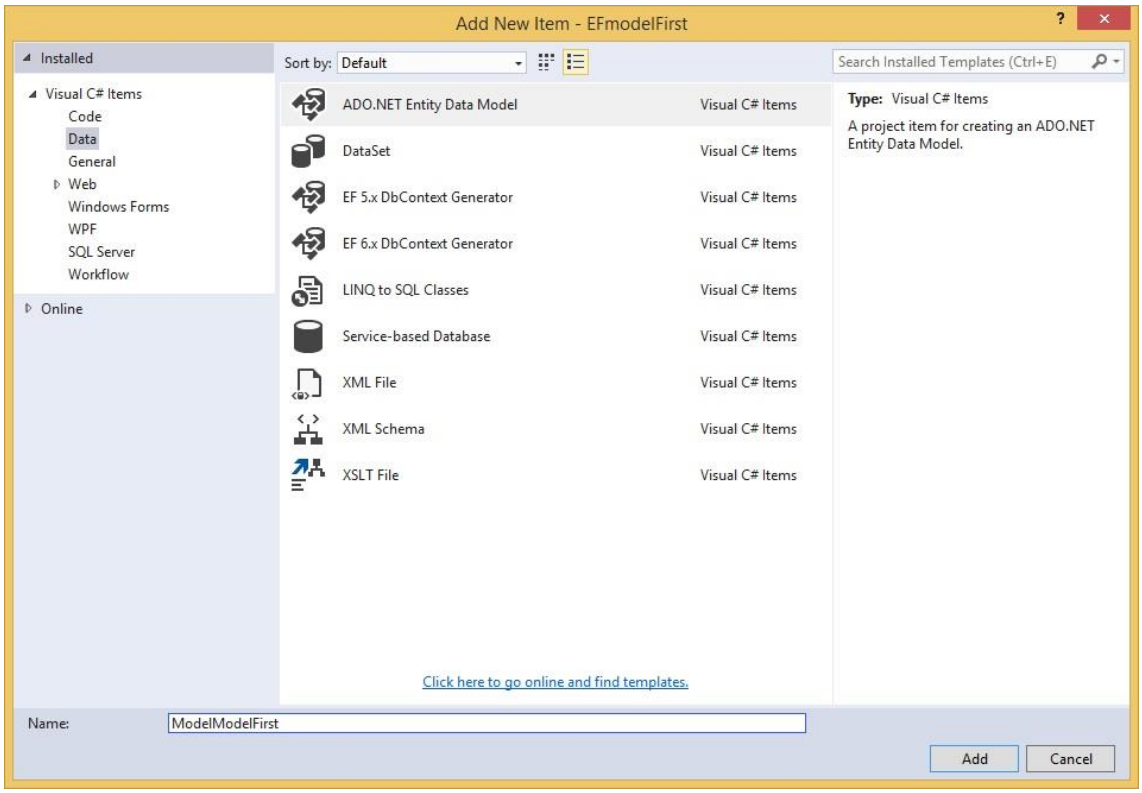


Obrázek 25: Vygenerovaný EDM strategií Database first (vlastní zpracování)

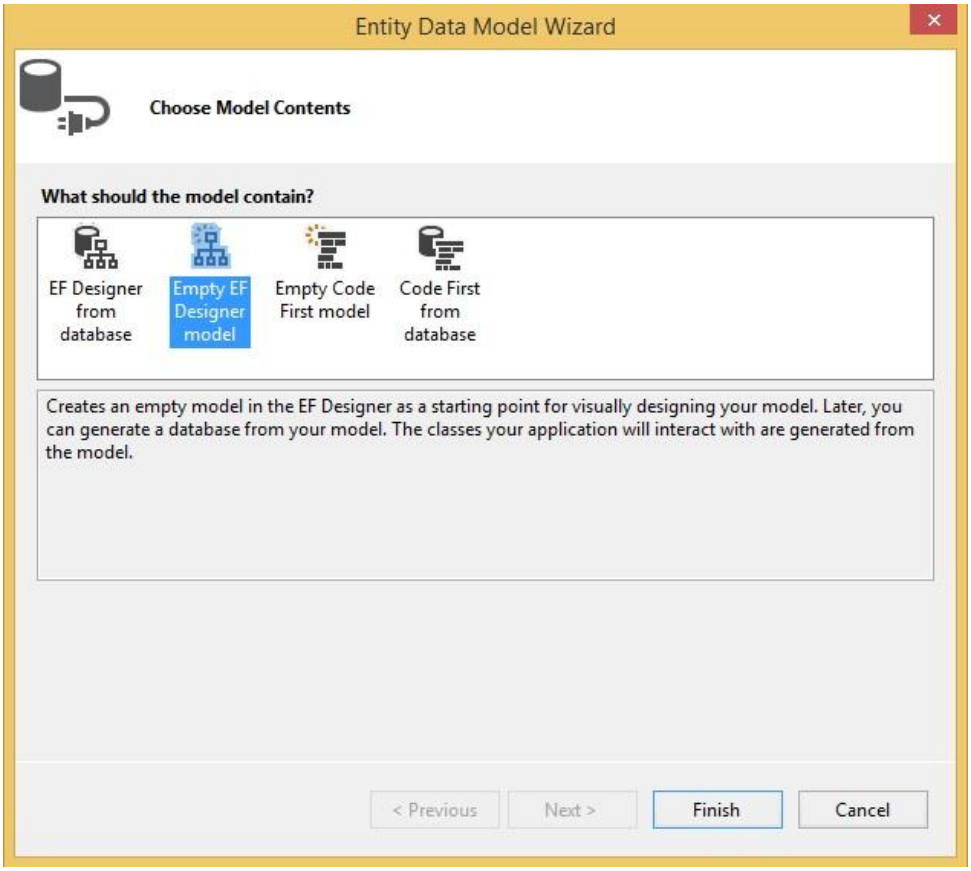
3.3.2 Model first přístup

Strategie „model first“ je vytvoření EDM přímo v designeru, který je součástí Visual Studia. Entity Framework následně vygeneruje a spustí skripty potřebné k vytvoření databáze. V designeru lze navrhovat entity, vztahy, dědičnost, standardní hodnoty či některá integritní omezení.

Pro Model first přístup byl vytvořen projekt typu *Class Library* s názvem *EFmodelFirst*, do kterého byl přidán *ADO.NET Entity Data Model* (obrázek 26) a následně zvolena možnost *EF Empty designer model* (obrázek 27), kde byly vytvořeny požadované entity (tabulky), včetně jejich vazeb a integritních omezení.

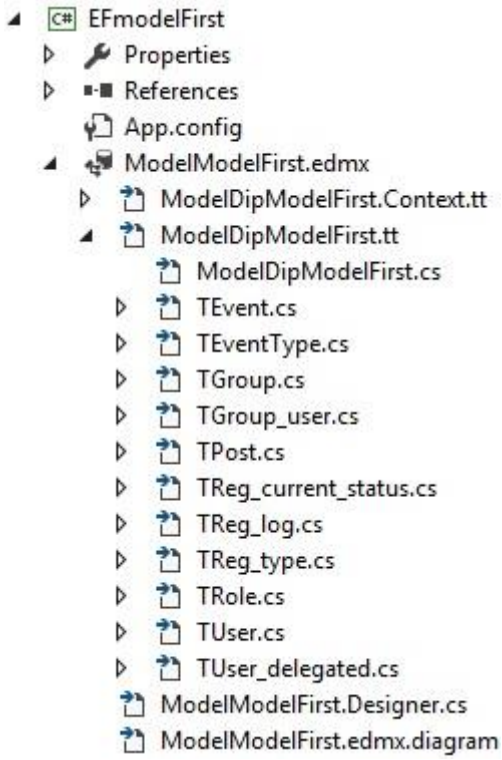


Obrázek 26: Přidání ADO.NET EDM do projektu EFmodelFirst (vlastní zpracování)



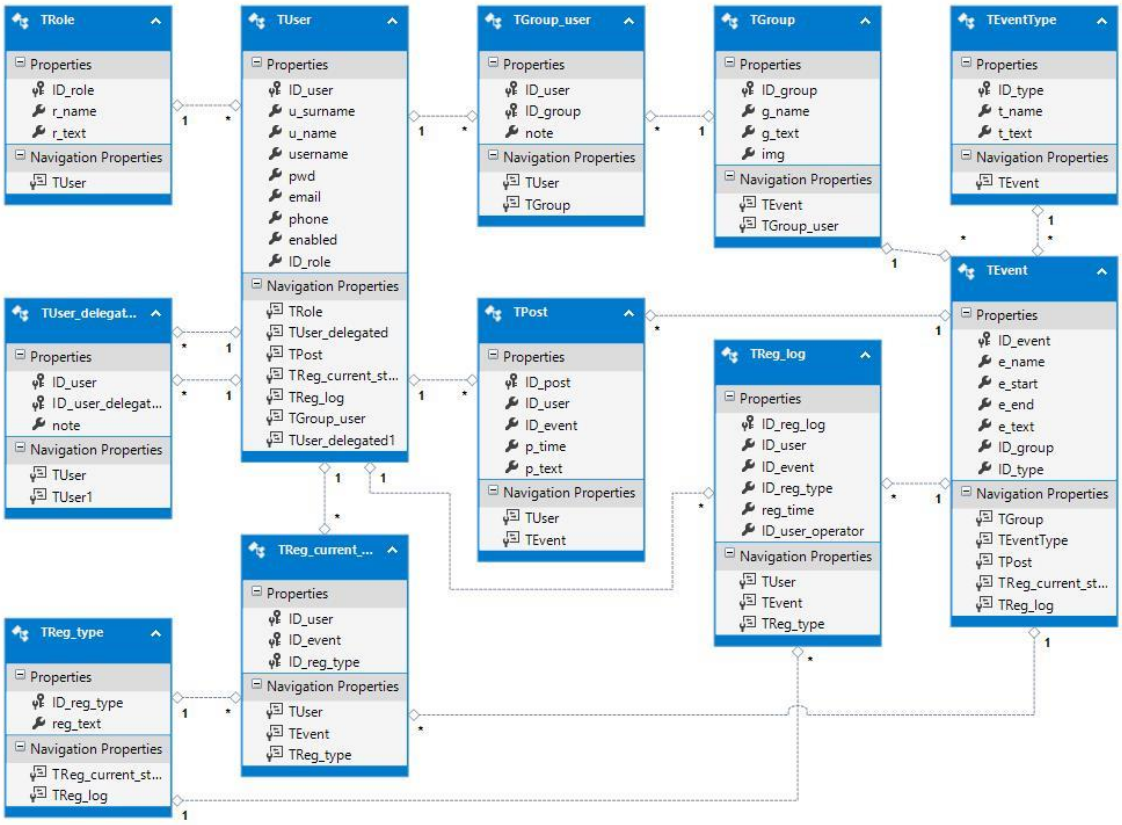
Obrázek 27: Empty EF Designer model (vlastní zpracování)

Konečná struktura projektu *EFmodelFirst* se nachází na obrázku 28.



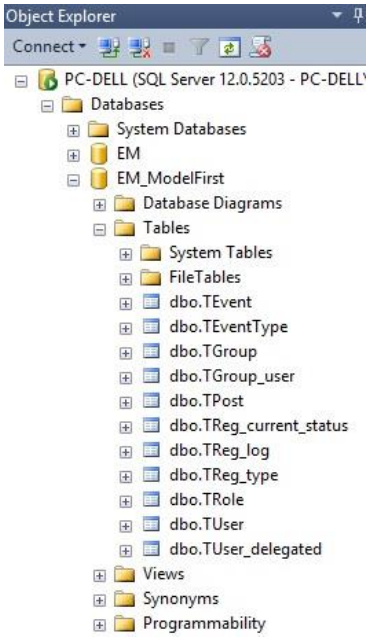
Obrázek 28: Struktura projektu EFmodelFirst (vlastní zpracování)

Na obrázku 29 se nachází výsledný Entity Data Model, který byl vytvořen prostřednictvím strategie Model first.



Obrázek 29: Vytvořený EDM strategií Model first (vlastní zpracování)

Z modelu byl vygenerován SQL skript pro založení databáze, který byl spuštěn. Vytvořená databáze má název *EM_ModelFirst* a její struktura se nachází na obrázku 30.



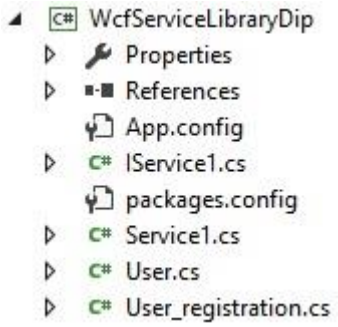
Obrázek 30: Struktura databáze EM_ModelFirst (vlastní zpracování)

3.3.3 Srovnání přístupu Database first a Model first

Třídy obou projektů *EFdatabaseFirst* a *EFmodelFirst* jsou naprosto totožné. Tudíž jde o individuální potřebu, jaký přístup daný programátor preferuje. V tomto případě je pro další vývoj zvolena možnost Database first a v další části práce jsou třídy projektu *EFdatabaseFirst* využívány vytvořenou WCF službou.

3.4 WCF služba

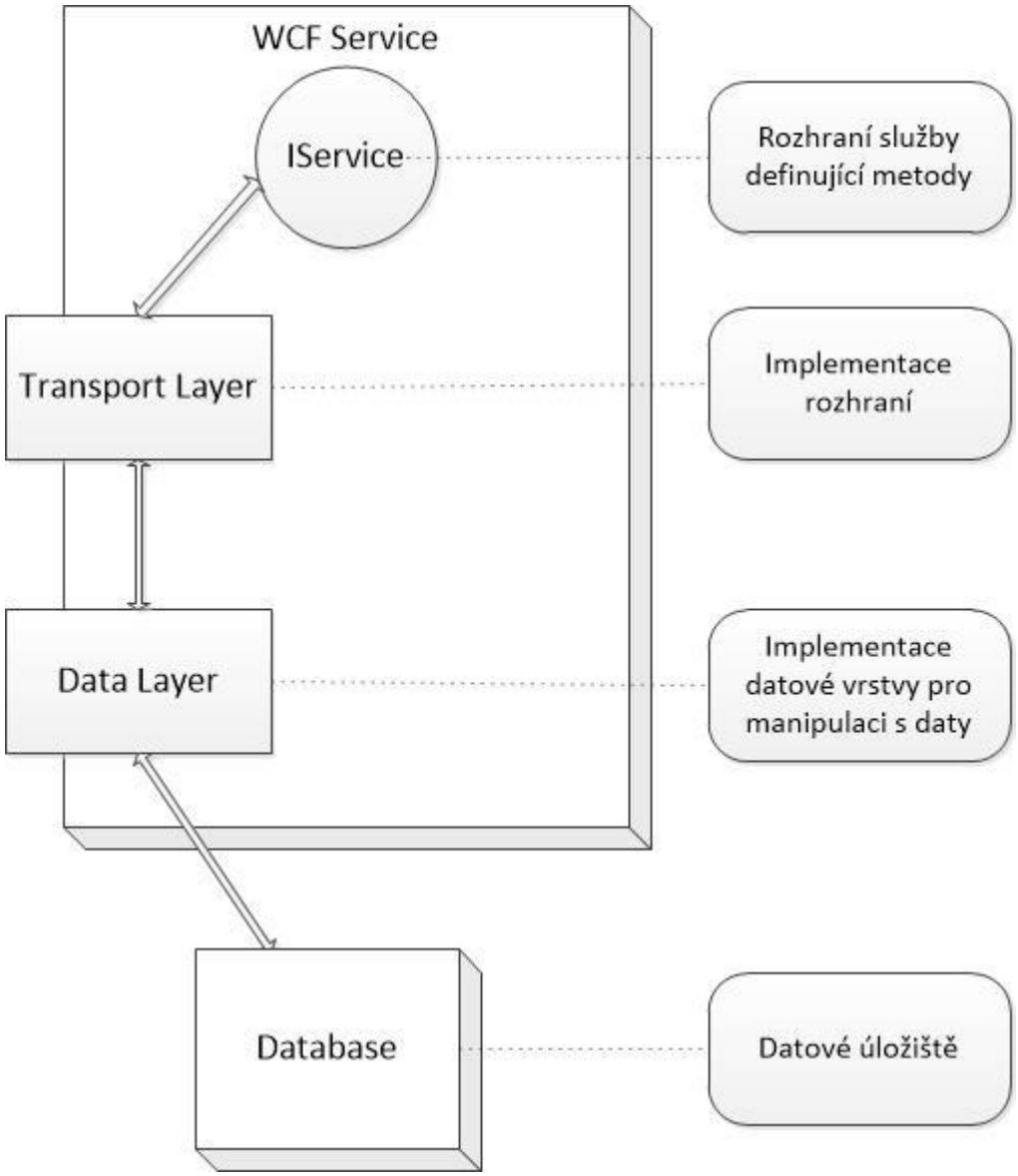
V této části je vytvořena WCF služba v prostředí .NET implementována v jazyce C#. Tato WCF služba je otestována a zabezpečena z hlediska přístupu k jednotlivým metodám služby. Pro WC službu byl vytvořen projekt typu *WCF Service Library* s názvem *WcfServiceLibraryDip*.



Obrázek 31: Struktura projektu WcfServiceLibraryDip (vlastní zpracování)

3.4.1 Architektura

WCF služba se skládá ze dvou samostatných vrstev. První vrstva je tzv. transportní vrstva a tvoří ji samotná WCF služba svým rozhraním a implementací metod, které jsou definovány tímto rozhraním. Druhou vrstvou je datová vrstva, která zajišťuje veškerou manipulaci s daty nad databází. K její implementaci je použita technologie .NET Entity Framework fungující na principu mapování databázových tabulek – viz. kapitola 3.3. Každá databázová tabulka je ve formě entity implementována v datové vrstvě a dotazy do databáze jsou realizovány prostřednictvím jazyka LINQ přímo v programovém kódu jazyka C#. Tímto řešením je docíleno rozdělení výpočetní zátěže databázového serveru a přenesení zátěže z části na datovou vrstvu. Datové úložiště je řešeno databází, která je provozována na MS SQL Serveru 2012. Strukturu služby a její vrstvy znázorňuje obrázek 32.



Obrázek 32: Architektura WCF služby (vlastní zpracování)

3.4.2 Technologické řešení implementace WCF služby

Je implementováno rozhraní *IService* WCF služby, které zpřístupňuje metody a umožňuje jejich volání externě. Toto rozhraní je zároveň vzdáleným koncovým bodem, ke kterému se budou připojovat klientské aplikace. V rozhraní je přiřazen atribut *[ServiceContract]* třídě reprezentující službu a jednotlivým metodám atribut *[OperationContract]*. Následující zdrojový kód je částečnou ukázkou implementace služby, která byla vytvořena.

```
[ServiceContract]
    public interface IService1
    {
        #region user
        [OperationContract]
        bool insertUser(TUser user);

        [OperationContract]
        bool updateUser(TUser user);

        [OperationContract]
        bool deleteUser(int id);

        [OperationContract]
        TUser getUserInfo(int id);

        [OperationContract]
        List<TGroup> getUserGroups(int id);

        [OperationContract]
        List<TEvent> getUserEvents(int id);

        [OperationContract]
        List<TUser> getUsersInGroup(int id);

        [OperationContract]
        List<User> getAllDelegatedUsers(int id_user);

        [OperationContract]
        List<User> getAllUsers(int id_user);
        #endregion

        #region group
        [OperationContract]
        bool insertGroup(TGroup group);

        [OperationContract]
        bool updateGroup(TGroup group);

        [OperationContract]
        bool deleteGroup(int id);
...
#endregion
}
```

Jednotlivé metody rozhraní *IService* jsou následně implementovány ve třídě *Service*. Níže je uveden zdrojový kód části zmíněné třídy, konkrétně jde o smazání uživatele.

```
public bool deleteUser(int id)  //smazání uživatele + odstranění všech vazeb
{
       try
       {
       TUser userdb = dip.TUser.Where(u => u.ID_user == id).FirstOrDefault();
       if (userdb != null)
       {
```

```
        List<TGroup_user> grpusers = dip.TGroup_user.Where(r => r.ID_user
        == id).ToList();
if (grpusers != null)
{
        foreach (TGroup_user g in grpusers)
        {
                dip.TGroup_user.Remove(g);
        }
        dip.SaveChanges();  //vymazání tabulek registrací mazaného
        uživatele
}
List<TReg_current_status> regs = dip.TReg_current_status.Where(r =>
r.ID_user == id).ToList();
if (regs != null)
{
        foreach (TReg_current_status r in regs)
        {
                dip.TReg_current_status.Remove(r);
        }
        dip.SaveChanges();  //vymazání tabulek registrací mazaného
        uživatele
        }
        List<TReg_log> reg_logs = dip.TReg_log.Where(l => l.ID_user ==
        id).ToList();
        if (reg_logs != null)
        {
                foreach (TReg_log l in reg_logs)
                {
                        dip.TReg_log.Remove(l);
                }
                dip.SaveChanges();  //vymazání tabulek logů mazaného
                uživatele
                }
                List<TPost> posts = dip.TPost.Where(p => p.ID_user ==
                id).ToList();
                if (posts != null)
                {
                        foreach (TPost p in posts)
                        {
                                dip.TPost.Remove(p);
                        }
                        dip.SaveChanges();  //vymazání tabulek postů mazaného
                        uživatele
                }
                dip.TUser.Remove(userdb);
                dip.SaveChanges();
                return true;
           }
           else
                return false;
     }
     catch
     {
          return false;
     }
 }
```

Všechny metody, funkce a objekty tvoří samostatnou (transportní) vrstvu WCF služby. Tato vrstva je nadřazená datové vrstvě v hierarchii architektury služby.

3.4.3 Datová vrstva

V datové vrstvě odpovídá každá třída objektu ekvivalentní entitě datového modelu, která je vygenerována pomocí vývojových prostředků již zmiňované technologie .NET Entity Framework postavené na principu mapování jednotlivých databázových tabulek. Pro příklad je uvedena třída *TPost* (příspěvek).

```
public partial class TPost
{
	public int ID_post { get; set; }
	public int ID_user { get; set; }
	public int ID_event { get; set; }
	public System.DateTime p_time { get; set; }
	public string p_text { get; set; }

	public virtual TUser TUser { get; set; }
	public virtual TEvent TEvent { get; set; }
}
```

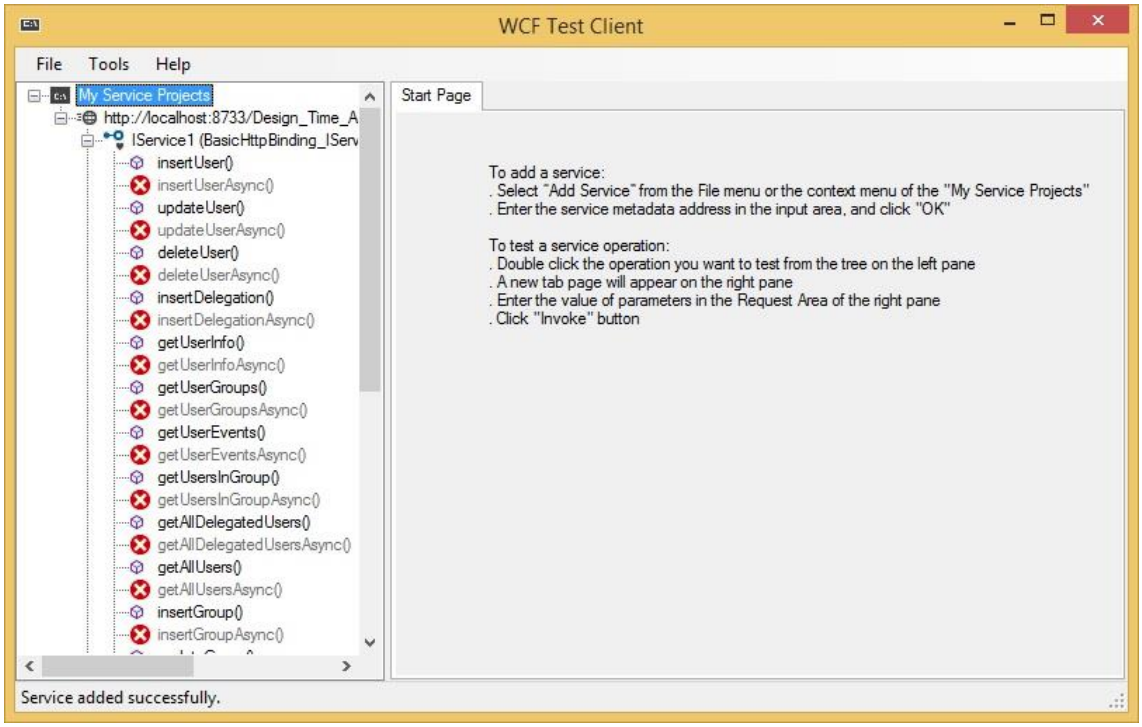
3.4.4 Zabezpečení přístupu k WCF službě

Je nutné zajistit, aby k veřejným metodám WCF služby měli přístup pouze oprávnění uživatelé. Před každou metodu je přidán atribut s parametrem role, které mohou tyto metody volat. Například metodu pro vložení nové skupiny může volat pouze *SuperAdmin*.

```
[PrincipalPermission(SecurityAction.Demand, Authenticated = true, Role =
"SuperAdmin")]
public bool insertGroup(TGroup group)
{
	TGroup groupdb = db.TGroup.Where(g => g.g_name ==
	group.g_name).FirstOrDefault();
	if (groupdb == null)
	{
		try
		{
			db.TGroup.Add(group);
			db.SaveChanges();
			return true;
		}
		catch
		{
			return false;
		}
	}
	else
		return false;
}
```

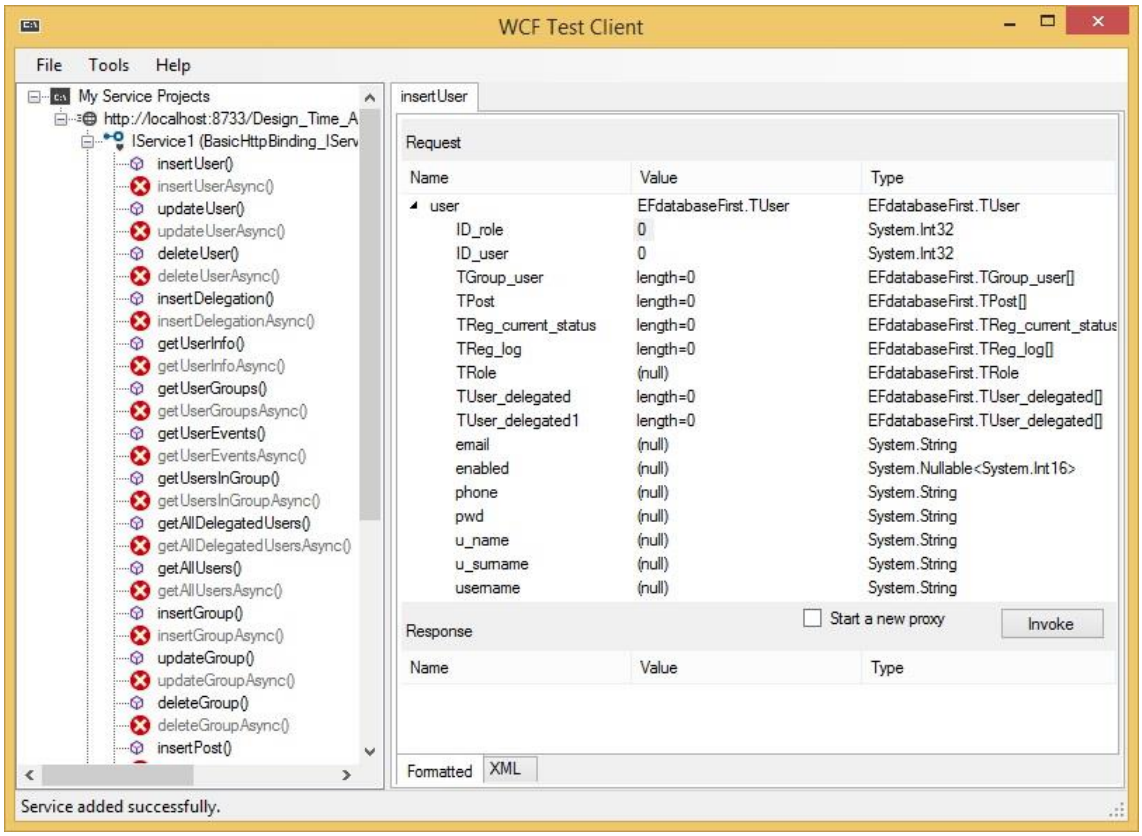
3.4.5 WCF Test Client

Jednou z možností otestování vytvořené WCF služby je WCF Test Client, kdy je projekt *WcfServiceLibraryDip* nastaven jako spouštěcí (pravým tlačítkem myši na daný projekt - Set as Start Up Project) a po spuštění tlačítkem Start se otevře nové okno WCF Test Client.



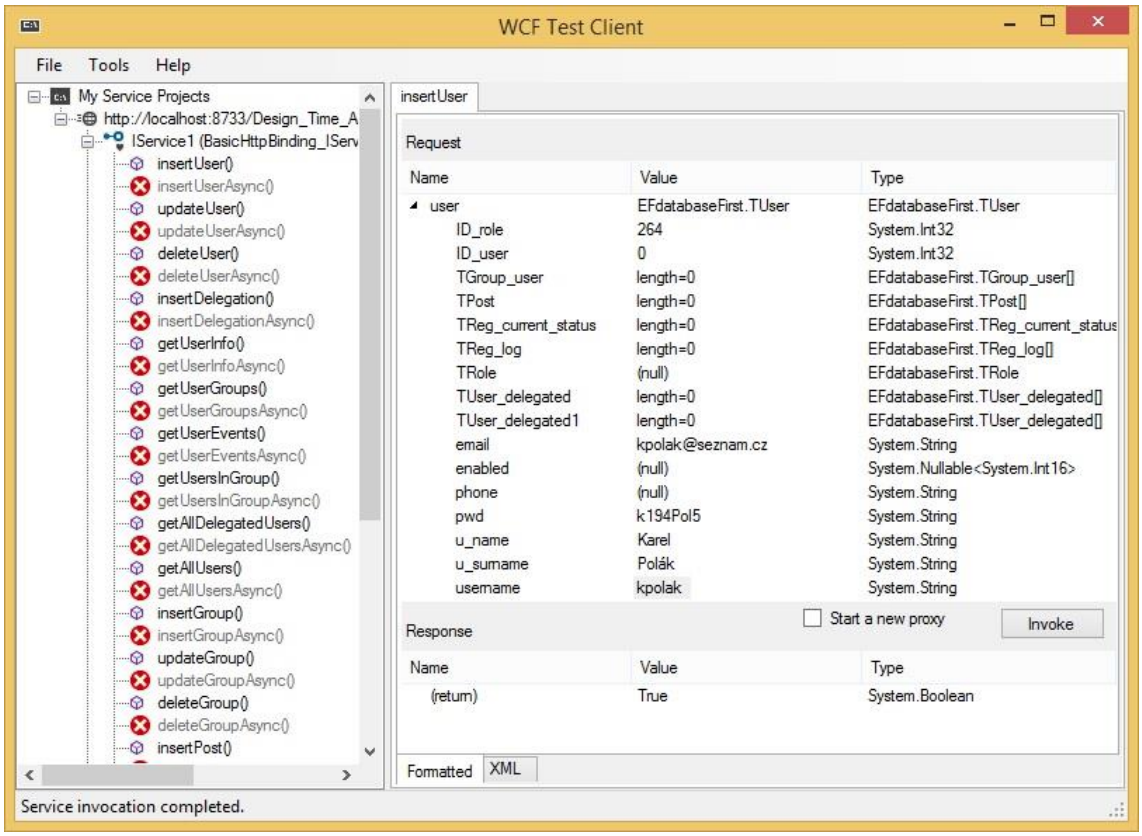
Obrázek 33: WCF Test Client – Start Page (vlastní zpracování)

Toto okno obsahuje všechny funkce implementované ve WCF službě a po kliknutí na kteroukoli z nich, se zobrazí vstupní množina parametrů, které je potřeba zadat, aby mohla být otestována správnost dané funkce.



Obrázek 34: WCF Test Client - insertUser (vlastní zpracování)

Například u funkce pro přidání uživatele *insertUser* je nutné vyplnit pole *Value* u příslušných atributů třídy *TUser*. Po vyplnění povinných atributů jako jsou *ID_role*, *email*, *pwd*, *u_name*, *u_surname* je po zmáčknutí tlačítka *Invoke* v tomto případě vrácena hodnota *true* nebo *false* (jelikož se jedná o metodu s návratovým typem *bool*).



Obrázek 35: WCF Test Client – insertUser „Invoke“ (vlastní zpracování)

Obrázek 36: Ověření přidání uživatele na SQL Serveru (vlastní zpracování)

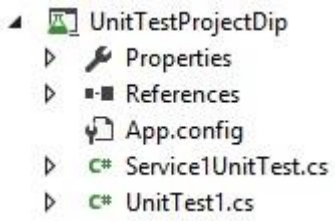
True znamená, že byl uživatel úspěšně přidán, což je možné ověřit přímo na SQL Serveru – viz. obrázky 35 a 36. Vrácení false by znamenalo, že uživatel přidán nebyl, ovšem není známo, v čem konkrétně je chyba a je potřeba dalšího zkoumání a debugování (testování a ladění) dané funkce. Navíc se o tomto testování neukládají

žádné informace a není možné se k danému problému pohodlně vrátit později. Proto je pro tyto potřeby lepší zvolit unit testování, které je rozebráno v následující části.

3.5 Unit testování

Pro ověření funkčnosti implementace WCF služby byly vytvořeny unit testy pomocí Visual Studio Unit Testing Frameworku, který je integrovaný přímo ve Visual Studiu. Toto testování probíhalo postupným voláním všech funkcí s různými argumenty, které byly zvoleny takovým způsobem, aby byly otestovány veškeré možné reakce funkcí na vstup.

Pro unit testování byl vytvořen projekt typu Unit Test Project s názvem *UnitTestProjectDip*. Jeho struktura se nachází na obrázku 37.



Obrázek 37: Struktura projektu UnitTestProjectDip (vlastní zpracování)

Nejprve bylo nutné do konfiguračního souboru *App.config*. přidat nastavení *connectionStrings*, které představuje připojovací řetězec k datovému modelu projektu *EFdatabaseFirst*.

```
<connectionStrings>
    <add name="EMEntitiesDatabaseFirst"
    connectionString="metadata=res://*/ModelDatabaseFirst.csdl|res://*/ModelDatab
    aseFirst.ssdl|res://*/ModelDatabaseFirst.msl;provider=System.Data.SqlClient;p
    rovider connection string=&quot;data source=.;initial catalog=EM;integrated
    security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
    providerName="System.Data.EntityClient" />
</connectionStrings>
```

Testovací třída *UnitTest1.cs* obsahuje testovací metodu *ClearDb()*, která smaže všechny záznamy ve všech tabulkách databáze, a metodu *SetupDb()*, ve které je volána zmiňovaná metoda ClearDb() a následně jsou vložena testovací data do prázdných tabulek.

```
[TestClass]
public class UnitTest1
{
	[TestMethod]
	public void ClearDb()
	{
		using (var db = new DipEntities())
		{
		db.TPost.RemoveRange(db.TPost);
		db.TReg_log.RemoveRange(db.TReg_log);
		db.TReg_current_status.RemoveRange(db.TReg_current_status);
		db.TEvent.RemoveRange(db.TEvent);
		db.TEventType.RemoveRange(db.TEventType);
		db.TReg_type.RemoveRange(db.TReg_type);
		db.TGroup_user.RemoveRange(db.TGroup_user);
		db.TUser.RemoveRange(db.TUser);
		db.TGroup.RemoveRange(db.TGroup);
		db.TRole.RemoveRange(db.TRole);
		db.SaveChanges();
		}
	}

	public void SetupDb()
	{
		Console.WriteLine("SetupDb " + DateTime.Now.ToShortTimeString());
		ClearDb();

		using (var db = new DipEntities())
		{
			//vložení testovacích dat do prázdných tabulek
			db.TRole.Add(new TRole { ID_role = 1, r_name = "SuperAdmin",
			r_text = "Hlavní administrátor systému" });
			db.TRole.Add(new TRole { ID_role = 2, r_name = "GroupAdmin",
			r_text = "Administrátor skupin" });
			db.TRole.Add(new TRole { ID_role = 3, r_name = "User", r_text
			= "Běžný uživatel" });

			db.TGroup.Add(new TGroup { ID_group = 1, g_name = "Tenis
			Brno", g_text = "Tenisová škola Brno" });

			db.TUser.Add(new TUser { ID_user = 1, ID_role = 1, username =
			"pet", u_name = "Petr", u_surname = "Novák", email =
			"petrnovak@seznam.cz", pwd = "xxx" });
			db.TUser.Add(new TUser { ID_user = 2, ID_role = 2, username =
			"jan", u_name = "Jan", u_surname = "Svoboda", email =
			"jansvoboda@seznam.cz", pwd = "xxx" });

			db.TGroup_user.Add(new TGroup_user { ID_group = 1, ID_user =
			1 });
			db.TGroup_user.Add(new TGroup_user { ID_group = 1, ID_user =
			2 });

			db.TReg_type.Add(new TReg_type { ID_reg_type = 1, reg_text =
			"Přihlášení" });
			db.TReg_type.Add(new TReg_type { ID_reg_type = 2, reg_text =
			"Odhlášení" });
			db.TReg_type.Add(new TReg_type { ID_reg_type = 3, reg_text =
			"Náhradník" });
```

```
            db.TEventType.Add(new TEventType { ID_type = 1, t_name =
            "Fotbal", t_text = "Pravidelný fotbalový trénink" });
            db.TEventType.Add(new TEventType { ID_type = 2, t_name =
            "Tenis 2", t_text = "Tenis dvouhra" });
            db.TEventType.Add(new TEventType { ID_type = 3, t_name =
            "Tenis 4", t_text = "Tenis 4 hra" });

            db.TEvent.Add(new TEvent
            {
                  ID_event = 1,
                  ID_group = 1,
                  ID_type = 2,
                  e_name = "Tenis pátek 10.3.2017",
                  e_start = Convert.ToDateTime("2017-03-10
                  18:00:00.000"),
                  e_end = Convert.ToDateTime("2017-03-10 20:00:00.000"),
                  e_text = "Hala VUT - dvouhra"
            });

            db.TPost.Add(new TPost { ID_event = 1, ID_user = 2, p_text =
            "Nemůžu přijít.", p_time = DateTime.Now });

            db.SaveChanges();
            }
      }
```

V třídě Service1UnitTest.cs se nachází metoda *ClassInitialite(TestContext tc)* označená atributem *[ClassInitialize()]*, což označuje, že se spustí jen jednou pro všechny testy v dané třídě, a to před spuštěním samotných testovacích metod. Totéž platí u metody *ClassCleanUp()* označené atributem *[ClassCleanup]*, ovšem ta se spustí, až proběhnou všechny testy v dané třídě. Metoda *TestInitialize()* označená atributem *[TestInitialize()]* běží před a po každém testu.

```
[TestClass()]
public class Service1UnitTest
{
      int idTestUser = 0;
      int idTestGroup = 0;

      [ClassInitialize()]
      public static void ClassInitialize(TestContext tc)
      {
            (new UnitTest1()).SetupDb();
            tc.WriteLine("Začátek testů " + DateTime.Now.ToString());
      }
      [TestInitialize()]
      public void TestInitialize()
      {
            using (var db = new DipEntities())
            {
            idTestUser = db.TUser.Where(p => p.username == "pet").Select(p =>
            p.ID_user).FirstOrDefault();
            idTestGroup = db.TGroup.Where(g => g.g_name == "Tenis
            Brno").Select(g => g.ID_group).FirstOrDefault();
```

```
            db.SaveChanges();
            }
            Console.WriteLine("Vybraný uživatel má ID_user = " +
            idTestUser.ToString() + " " + DateTime.Now.ToString());
            Console.WriteLine("Vybraná skupina má ID_group = " +
            idTestGroup.ToString() + " " + DateTime.Now.ToString());
        }

        [ClassCleanup]
        public static void ClassCleanUp()
        {
            Console.WriteLine("Konec testů, mazání databáze " +
            DateTime.Now.ToString());
            (new UnitTest1()).ClearDb();
        }
```

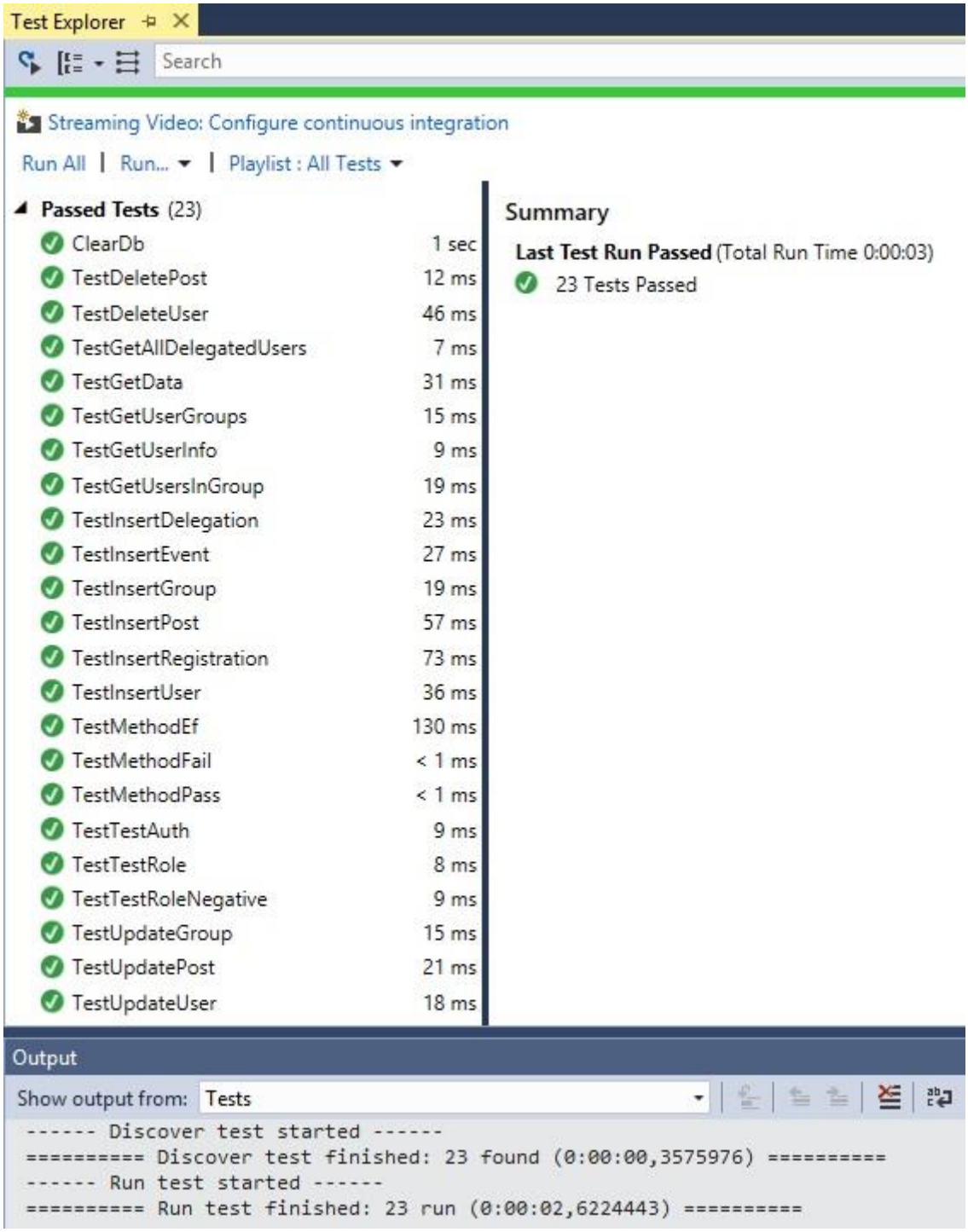
Dále se v dané testovací třídě nacházejí testovací metody pro jednotlivé funkce WCF služby, označené atributem *[TestMethod()]*. Pro ukázku je uvedena testovací metoda *TestDeleteUser()*, která otestuje funkci pro smazání uživatele a všech jeho vazeb (registrace, logy, příspěvky) z databáze.

```
[TestMethod()]
public void TestDeleteUser()
{
    Console.WriteLine("Začátek testu TestDeleteUser " +
    DateTime.Now.ToShortTimeString());
    using (var db = new DipEntities())
    {
        int ID_user = db.TUser.Where(p => p.username == "jan").Select(p =>
        p.ID_user).FirstOrDefault();
        Service1 s = new Service1();
        Console.WriteLine("Mazaný uživatel má ID_user = " +
        ID_user.ToString() + " " + DateTime.Now.ToString());
        s.deleteUser(ID_user);

        TUser user = db.TUser.Where(u => u.username ==
        "rad").FirstOrDefault();
        TReg_current_status user_reg = db.TReg_current_status.Where(r =>
        r.ID_user == ID_user).FirstOrDefault();
        TReg_log user_log = db.TReg_log.Where(l => l.ID_user ==
        ID_user).FirstOrDefault();
        TPost user_post = db.TPost.Where(p => p.ID_user ==
        ID_user).FirstOrDefault();

        Assert.IsNull(user, "Uživatel nebyl smazán");
        Assert.IsNull(user_reg, "Registrace uživatele nebyly smazány");
        Assert.IsNull(user_log, "Logy uživatele nebyly smazány");
        Assert.IsNull(user_post, "Příspěvky uživatele nebyly smazány");
    }
}
```

Unit testy byly spuštěny v *Test Exploreru*, který je součástí Visual Studia. Na obrázku 38 se nachází výsledek unit testování. Všechny testy proběhly v pořádku, to znamená, že metody WCF služby jsou funkční.



Obrázek 38: Výsledek unit testování (vlastní zpracování)

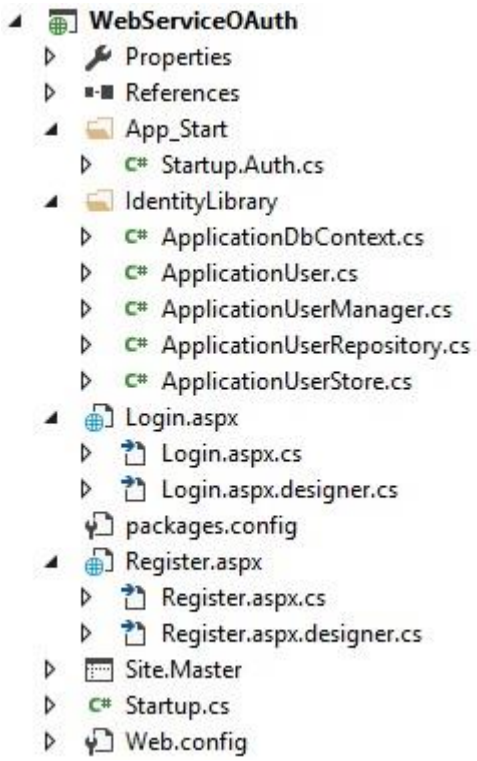
3.6 Autentizace uživatele

Pro autentizaci je využito ASP.NET Identity 2.0, což je technologie od Microsoftu, která umožnuje jednoduchou správu uživatelů, přiřazování uživatelů do rolí a práci s nimi, a je možné si tuto správu nastavit dle vlastních potřeb. (30)

Byl vytvořen projekt typu *ASP.NET Web Application* s názvem *WebServiceOAuth*, ve kterém bylo nakonfigurováno *ASP.NET Identity*, které pro přístup k datům používá .NET Entity Framework. Tudíž bylo mimo jiné potřeba do konfiguračního souboru *Web.config*. přidat nastavení *connectionStrings*, které představuje připojovací řetězec k datovému modelu projektu *EFdatabaseFirst*.

```
<connectionStrings>
    <add name="EMEntitiesDatabaseFirst"
    connectionString="metadata=res://*/ModelDatabaseFirst.csdl|res://*/ModelDatab
    aseFirst.ssdl|res://*/ModelDatabaseFirst.msl;provider=System.Data.SqlClient;p
    rovider connection string=&quot;data source=PC-DELL;initial
    catalog=EM;integrated
    security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
    providerName="System.Data.EntityClient" />
</connectionStrings>
```

Výsledná struktura projektu *WebServiceOAuth* se nachází na obrázku 39.



Obrázek 39: Struktura projektu WebServiceOAuth (vlastní zpracování)

Jádro tohoto projektu tvoří pět tříd, jež se nacházejí v adresáři *IdentityLibrary*:

- *ApplicationDbContext.cs* – třída pro propojení na databázi
- *ApplicationUser.cs* – třída pro uživatele
- *ApplicationUserManager.cs* – třída pro správu uživatelů, která mimo jiné umožňuje nastavit např. validační pravidla pro sílu uživatelských hesel
- *ApplicationUserRepository.cs* – třída, která obsahuje metody, jež jsou aplikovány na uživatele (jinými slovy úložiště, které obsahuje logiku pro získávání a ukládání dat)
- *ApplicationUserStore.cs* – třída, která údaje o uživatelích promítá do databáze

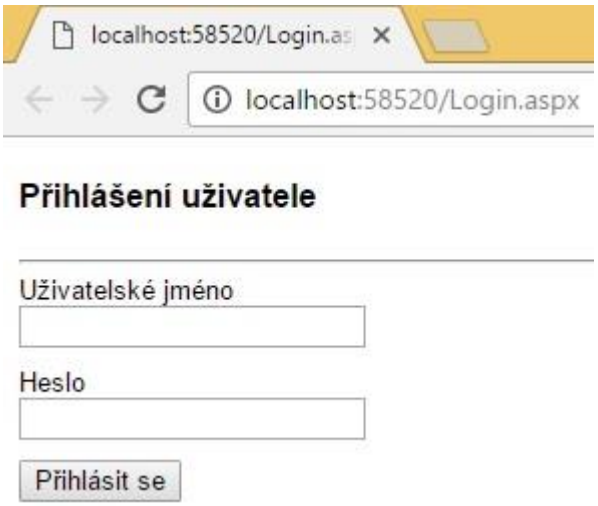
Pro registrace nového uživatele je vytvořen formulář viz. obrázek 40.

Obrázek 40: Formulář pro registraci (vlastní zpracování)

Pro zvýšenou bezpečnost systému obsahuje ASP.NET Identity také funkcionalitu hashování uživatelského hesla a díky tomu je do databáze ukládán pouze hash textového řetězce a samotná hodnota hesla zůstává utajena i pro administrátora databáze viz. obrázek 41. Hashování zajišťuje vlastnost *PasswordHash* třídy *IdentityUser*, ze které dědí již zmiňovaná třída *ApplicationUser*.

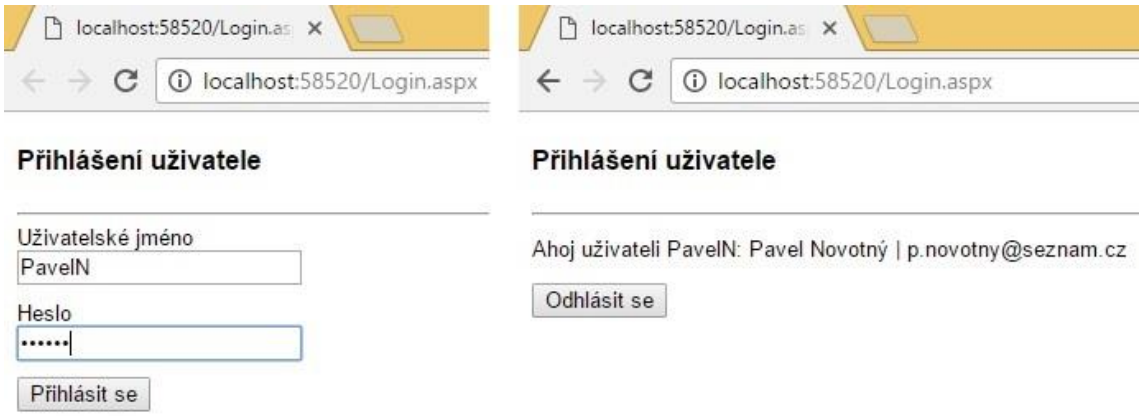
Obrázek 41: Heslo v databázi v podobě hashovaného řetězce (vlastní zpracování)

K autentizaci uživatele je vytvořen formulář pro přihlášení viz. obrázek 42.



Obrázek 42: Formulář pro přihlášení (vlastní zpracování)

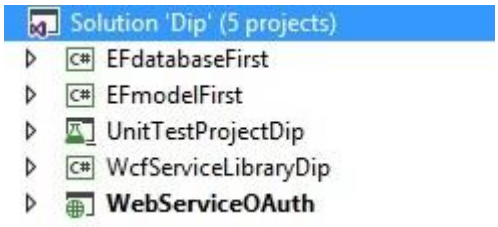
Při ověřování hesla uživatele, který se do systému přihlašuje, dochází taktéž k hashování zadaného hesla a porovnání s hashovaným textovým řetězcem uloženým v databázi. Pokud se oba řetězce shodují, tak je uživatel úspěšně autentizován.



Obrázek 43: Autentizace uživatele PavelN (vlastní zpracování)

3.7 Shrnutí návrhu

V prvním kroku byl zpracován návrh databázové struktury, jež byla realizována na databázovém serveru *MS SQL Server*. K tomu bylo využito *SQL Management Studio 2014*. Jde o databázi s názvem *EM*, jejíž záloha se nachází na přiloženém disku. Příloha práce také obsahuje SQL skript pro vytvoření této databáze.



Obrázek 44: Struktura solution Dip (vlastní zpracování)

V dalších krocích již šlo o práci ve vývojovém prostředí *Visual Studio 2015*. Nejprve byla provázána data mezi databází EM a objekty v objektově orientovaném jazyce C# s využitím objektově relačního frameworku .NET Entity Framework. Byly provedeny dvě strategie přístupu – Database first a Model first (projekty *EFdatabaseFirst* a *EFmodelFirst*). V dalším projektu *WcfServiceLibryDip* je vytvořena samotná WCF služba, která jako datovou vrstvu využívá právě model projektu EFdatabaseFirst. Projekt *UnitTestProjectDip* otestoval vytvořenou WCF službu. V projektu *WebServiceOAuth* je nakonfigurováno ASP.NET Identity, které umožňuje správu uživatelů a jejich autentizace do systému. Řešení (*solution Dip*), které obsahuje zmíněných pět projektů, se nachází na přiloženém disku.

3.8 Zhodnocení přínosů

Hlavním přínosem návrhu řešení je vytvořený databázový systém, který umožňuje efektivně spravovat požadované informace o firemních akcích. Tento systém přináší několik vylepšení oproti původnímu stavu.

Uživatelé systému (zaměstnanci společnosti) jsou rozděleni do tří rolí (SuperAdmin, GroupAdmin, User), na základě kterých mají rozdílná práva na provádění činností a k přístupu k určitým informacím, což eliminuje hlavní problém současného nevyhovujícího stavu.

Zaměstnanci patří do různých zájmových skupin, ve kterých mají možnost si přehledně zobrazit akce příslušné skupiny a na tyto akce se následně přihlašovat či se z nich odhlašovat. Je možné si také například zobrazit akce, na něž je daný zaměstnanec již registrován, nebo také všechny ostatní uživatele v konkrétní skupině. Ke každé akci má navíc zaměstnanec možnost přidávat příspěvky (posty), díky čemuž se výrazně zlepší vzájemná komunikace a následně také zpětná vazba.

Data o firemních akcích jsou uložena v jednotné formě, což umožní přehlednější a rychlejší vyhledávání. Tato data je možné taktéž analyzovat a vyhodnocovat. Vedení společnosti tak může například jednoduše zjistit, kteří zaměstnanci jsou v dané zájmové skupině nejaktivnější a poté je v rámci toho odměňovat a také motivovat. Dále systém umožňuje přehledně zobrazit, o jaké firemní akce je největší zájem, a podle toho plánovat akce následující.

Protože je databázový systém vyvinut přímo ve společnosti, která disponuje vlastním databázovým serverem MS SQL Server a technologií .NET, na kterých je systém založen, není problém ho libovolně rozšiřovat bez dalších nákladů, které by v případě koupeného softwaru nepochybně vznikly. Vytvořený databázový systém bude ve společnosti využit pro vývoj webové a mobilní aplikace, která bude uplatněna v praxi.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit návrh databázového systému pro správu akcí ve společnosti ANeT-Advanced Network Technology, s. r. o.

Teoretická část práce představuje problematiku datového modelování, databázových systémů, objektově orientovaného programování a platformy .NET, a to s větším zaměřením na jazyk C#, LINQ a technologii WCF.

Druhá část práce je zaměřena na popis společnosti a analýzy její současné správy firemních akcí, na základě které jsou stanoveny požadavky na databázový systém. Taktéž je provedena analýza a výběr vhodného softwaru pro objektově relační mapování.

V návrhové části práce je stanoven postup řešení, kdy jsou nejprve identifikovány entity, o kterých je v systému potřeba uchovávat informace, a zpracován návrh databáze. Dále jsou provedeny dvě strategie přístupu v .NET Entity Framework. Na základě toho je vytvořena WCF služba, která je poté otestována. Na závěr je navíc implementována autentizace uživatele prostřednictvím technologie ASP.NET Identity.

Zpracováním návrhu databázového systému pro správu akcí byla realizována serverová část aplikace na základě požadavků zadavatele. Konečné řešení bylo ve společnosti předvedeno a společnost s ním souhlasí. Databázový systém bude dále využit pro vývoj webové a mobilní aplikace, které budou používat zaměstnanci společnosti.

SEZNAM POUŽITÝCH ZDROJŮ

(1) AGARWAL, V. V. a J. HUDDLESTON. *Databáze v C# 2008: průvodce programátora.* Brno: Computer Press, 2009. 424 s. ISBN 978-80-251-2309-6.

(2) CONOLLY, T., BEGG, C., HOLOWCZAK, R. *Mistrovství - databáze: Profesionální průvodce tvorbou efektivních databází.* 1. vyd. Brno: Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.

(3) EVJEN, B., S. HANSELMAN a D. RADER. *ASP.NET 3.5 v jazycích C# a Visual Basic*. Brno: Computer Press, 2009. 1598 s. ISBN 978-80-251-2069-9.

(4) KROENKE, D., D. J. AUER a J. GONER. *Databáze*. 1. vyd. Brno: Computer Press, 2015. 496 s. ISBN 978-80-251-4352-0.

(5) SHARP, J. *Microsoft Visual C# 2010: krok za krokem*. Brno: Computer Press, 2010. 696 s. ISBN 978-80-251-3147-3.

(6) WATSON, B. *C# 4.0: řešení praktických programátorských úloh*. Brno: Zoner Press, 2010. 656 s. ISBN 978-80-7413-094-6.

(7) LACKO, L. *Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]*. Brno: Computer Press, 2013. 640 s. ISBN 978-80-251-3773-4.

(8) PIALORSI, P. a M. RUSSO. *Microsoft LINQ: kompletní průvodce programátora*. Brno: Computer Press, 2009. 615 s. ISBN 978-80-251-2735-3.

(9) GILL, P. S. *Database management systems*. New Delhi: I.K. International, 2008. 280 s. ISBN 978-818-9866-839.

(10) ČADA, O. *Objektové programování: naučte se pravidla objektového myšlení.* Praha: Grada, 2009. 200 s. ISBN 978-80-247-2745-5.

(11) HUNT, A. a D. THOMAS. *Programátor pragmatik: jak se stát lepším programátorem a vytvářet kvalitní software.* Brno: Computer Press, 2007. 266 s. ISBN 978-80-251-1660-9.

(12) OPPEL, A. *SQL bez předchozích znalostí*. Brno: Computer Press, 2008. 240 s. ISBN 978-80-251-1707-1.

(13) PONNIAH, P. *Data Modeling Fundamentals a Practical Guide for IT Professionals*. Hoboken: John Wiley, 2007. 464 s. ISBN 978-047-0141-014.

(14) KOCH, M. *Datové a funkční modelování*. 2. vyd. Brno: Akademické nakladatelství CERM, 2006. 108 s. ISBN 80-214-2724-8.

(15) Entity Framework Overview. *MSDN Library – Microsoft* [online]. © 2017 [cit. 2017-05-12] Dostupné z: http://msdn.microsoft.com/en-us/library/bb399567.aspx

(16) What is Entity Framework? *Entity Framework Tutorial.net* [online]. © 2016 [cit. 2017-03-21]. Dostupné z: http://www.entityframeworktutorial.net/what-is-entityframework.aspx

(17) .NET Development. *MSDN Library – Microsoft* [online]. © 2017 [cit. 2017-03-30]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff361664(v=vs.110).aspx

(18) KHASAWNEH A. Understanding LINQ (C#). *Code Project* [online]. © 2017 [cit. 2017-04-03]. Dostupné z: https://www.codeproject.com/Articles/19154/Understanding-LINQ-C

(19) LINQ Query Syntax. *TutorialsTeacher.com* [online]. © 2017 [cit. 2017-04-07]. Dostupné z: http://www.tutorialsteacher.com/linq/linq-query-syntax

(20) LINQ Method Syntax. *TutorialsTeacher.com* [online]. © 2017 [cit. 2017-04-07]. Dostupné z: http://www.tutorialsteacher.com/linq/linq-method-syntax

(21) Windows Communication Foundation. *MSDN Library – Microsoft* [online]. © 2017 [cit. 2017-04-10]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/dd456779(v=vs.110).aspx

(22) SALVATORI P. Customizing and Extending the BizTalk WCF Adapters. *Paolo Savatori's Blog – Microsoft* [online]. 2009 [cit. 2017-04-14]. Dostupné z: https://blogs.msdn.microsoft.com/paolos/2009/11/17/customizing-and-extending-the-biztalk-wcf-adapters/

(23) Lambda Expression. *TutorialsTeacher.com* [online]. © 2017 [cit. 2017-04-15]. Dostupné z: http://www.tutorialsteacher.com/linq/linq-lambda-expression

(24) ANeT-Advanced Network Technology – O nás. *ANeT-Advanced Network Technology* [online]. © 2017 [cit. 2017-04-21]. Dostupné z: http://anet.eu/cz/

(25) PERKINS, B. *Working with NHibernate 3.0*. Indianapolis: John Wiley. 230 s. ISBN 978-111-8104-606.

(26) Chapter 2. Architecture. *Documentation – Nhibernate* [online]. © 2017 [cit. 2017-04-23]. Dostupné z: http://nhibernate.info/doc/nhibernate-reference/architecture.html

(27) LINQ to SQL. *MSDN Library – Microsoft* [online]. © 2017 [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/bb386976(v=vs.110).aspx

(28) GHOSH W. LINQ to SQL : Execution Architecture. *Wriju's BLOG – Microsoft* [online]. 2007 [cit. 2017-04-16]. Dostupné z: https://blogs.msdn.microsoft.com/wriju/2007/12/18/linq-to-sql-execution-architecture/

(29) Entity Framefork Tutorial. *EntityFrameworkTutorial.net* [online] © 2016 [cit. 2017-05-01]. Dostupné z: http://www.entityframeworktutorial.net/

(30) Microsoft ASP.NET Identity 2.0. *MSDN Library – Microsoft* [online]. © 2017 [cit. 2017-05-12]. Dostupné z: https://msdn.microsoft.com/en-us/library/mt173608(v=vs.108).aspx

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM POJMŮ A ZKRATEK

Zkratka	Pojem	Význam
	.NET Framework	Velké množství obecných knihoven řešících běžné úkony a virtuální stroj sloužící ke spouštění programů psaných pro tento framework.
	ADO.NET	Část .NET Frameworku určená pro přístup a datům a datovým službám.
	ASP.NET	Technologie určená pro vývoj webových aplikací nad .NET Frameworkem
	.NET Remoting	Infrastruktura pro distribuované objekty nad .NET Frameworkem
	Hashování	Transformace, která jako vstup přijímá řetězec znaků o libovolné délce a výsledkem je pak řetězec znaků s pevnou délkou, tzv. hash.
	Hash	Jednosměrný otisk vstupu, který se používá např. k zabezpečení hesel.
API	Application Programming Interface	Soubor definic podprogramů, protokolů a nástrojů pro vývoj aplikačního softwaru
ASMX	ASP.NET Webservices Source	Komponenta webové služby pro ASP.NET.
BCL	Base Class Libary	Sada základních knihoven .NET Frameworku.
C#		Objektově orientovaný jazyk pro platformu .NET.
C++		Objektově orientovaný programovací jazyk s přímou podporou v .NET Frameworku.
CLR	Common Language Runtime	Běhové prostředí .NET Frameworku, které zajišťuje běh a kompilaci aplikací

CLS	Common Language Specification	Sada základních jazykových funkcí jazyka potřebných pro vývoj aplikací.
DBMS (SŘBD)	Database Management System (Systém řízení báze dat)	Softwarový systém, který umožňuje definovat, vytvářet a udržovat databázi a poskytuje řízený přístup k této databázi
EF	.NET Entity Framework	Framework pro objektově relační mapování – součást .NET Frameworku.
ERD	Entity Relationship Diagram	Nástroj pro abstraktní a konceptuální znázornění dat. Zobrazuje množiny entit a množiny vztahů.
LINQ	Language Integrated Query	Jazyk určený pro dotazování nad jakýmikoli daty – součást .NET Frameworku
MSDN	Microsoft Developer Network	Síť vývojářů společnosti Microsoft s kompletní dokumentací, fóry či blogy.
NBÚ	Národní bezpečnostní úřad	Ústřední orgán státní správy ČR v oblasti ochrany utajovaných informací
SOA	Service Oriented Architecture	Architektonický styl a způsob analýzy, návrhu, integrace a údržby informačních systémů založených na službách.
SOAP	Simple Object Access Protocol	Protokol pro výměnu zpráv založených na XML a základ webových služeb.
SQL	Structured Query Language	Dotazovací jazyk používaný pro práci s daty v relačních databázích.
URI	Uniform Resource Identifier	Textový řetězec k jednotné identifikaci zdroje.
VS 2015	Visual Studio 2015	Vývojové prostředí od společnosti Microsoft.
WCF	Windows Communication Foundation	Framework pro vytváření servisně orientovaných aplikací
WPF	Windows Presentation Foundation	Rozhraní .NET framewroku pro návrh a zobrazování uživatelského prostředí
XML	Extensible Markup Language	Obecný značkovací jazyk.

SEZNAM PŘÍLOH

Příloha č. 1 – Datový slovník databáze

TEvent

Položka	Typ	Nulový	Klíč	Výchozí	Popis
ID_event	int	Ne	PK		ID akce
ID_group	int	Ne	FK		ID skupiny
ID_type	int	Ne	FK		ID typu akce
e_name	nvarchar(50)	Ne			Název akce
e_start	datatime	Ne			Začátek akce
e_end	datatime	Ne			Konec akce
e_text	text	Ano		NULL	Popis akce

TEvent_type

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_type	int	Ne	PK		ID typu akce
t_name	nvarchar(20)	Ne			Název typu akce
t_text	text	Ano		NULL	Popis typu akce

TGroup

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_group	int	Ne	PK		ID skupiny
g_name	nvarchar(30)	Ne			Název skupiny
g_text	text	Ano		NULL	Popis skupiny
img	image	Ano		NULL	Obrázek skupiny

TGroup_user

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_user	int	Ne	PK		ID uživatele
ID_group	int	Ne	PK		ID skupiny
note	varchar(50)	Ano		NULL	Poznámka

TPost

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_post	int	Ne	PK		ID příspěvku
ID_user	int	Ne	FK		ID uživatele
ID_event	int	Ne	FK		ID akce
p_time	datetime	Ne			Čas vytvoření příspěvku
p_text	nvarchar(100)	Ne			Text příspěvku

TReg_current_status

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_user	int	Ne	PK		ID uživatele
ID_event	int	Ne	PK		ID akce
ID_reg_type	int	Ne	PK		ID typu registrace

TReg_log

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_reg_log	int	Ne	PK		ID logu
ID_user	int	Ne	FK		ID uživatele
ID_event	int	Ne	FK		ID akce
ID_reg_type	int	Ne	FK		ID typu registrace
reg_time	datetime	Ne			čas vytvoření registrace
ID_user_operator	int	Ano		NULL	ID jiného uživatele, který provedl registraci za registrujícího uživatele

TReg_type

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_reg_type	int	Ne	PK		ID logu
reg_text	nvarchar(50)	Ne			ID uživatele

TRole

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_role	int	Ne	PK		ID uživatelské role
r_name	nvarchar(20)	Ne			Název role
r_text	text	Ano		NULL	Popis role

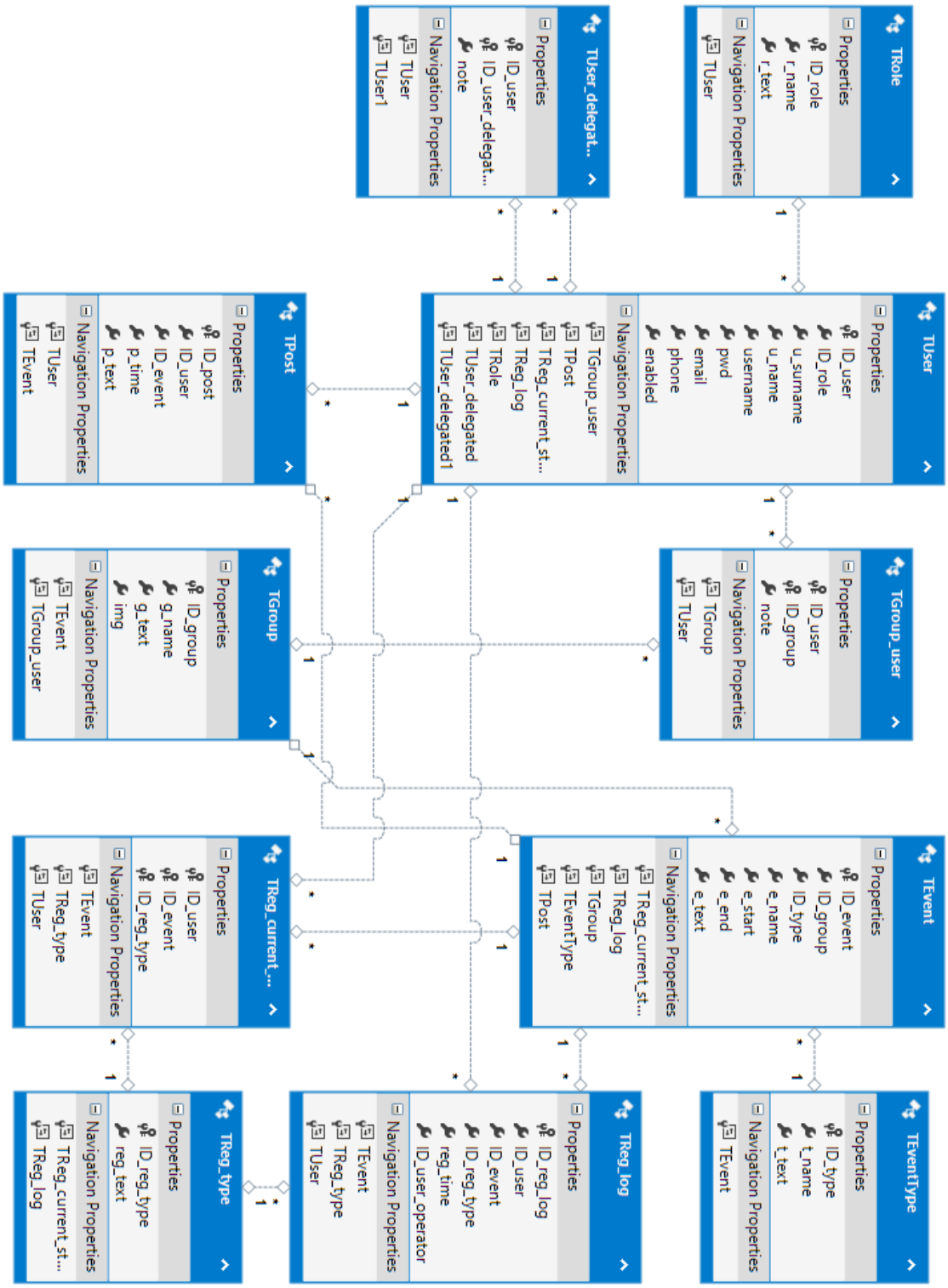
TUser

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_user	int	Ne	PK		ID uživatele
ID_role	int	Ne	FK		ID role
u_surname	nvarchar(50)	Ne			Příjemení
u_name	nvarchar(30)	Ne			Jméno
username	nvarchar(15)	Ne			Uživatelské jméno
pwd	nvarchar(MAX)	Ne			Heslo
email	nvarchar(50)	Ano		NULL	Emailová adresa
phone	nvarchar(13)	Ano		NULL	Mobilní telefon
enabled	smallint	Ano		NULL	Zda je uživatel povolen

TUser_delegated

Atribut	Typ	Nulový	Klíč	Výchozí	Popis
ID_user	int	Ne	PK		ID uživatele
ID_user_delegated	int	Ne	PK		ID delegovaného uživatele
note	nvarchar(50)	Ano		NULL	Poznámka

Příloha č. 2 – ER diagram databáze



Příloha č. 3 – Obsah přiloženého disku

Přiložený disk obsahuje tři hlavní adresáře, jež obsahují další podadresáře či soubory:

- **Text**
  - *Sekula_DP.pdf* – výsledný dokument diplomové práce
- **Db**
  - *EM.bak* – záloha databáze
  - *EM_create.sql* – SQL skript pro vytvoření databáze
- **Dip**
  - *EFdatabaseFirst* – projekt strategie Database first
  - *EFmodelFirst* – projekt strategie Model first
  - *packages* – balíčky pro knihovny využívané projekty
  - *UnitTestProjectDip* – projekt pro testování WCF služby
  - *WcfSeviceLibraryDip* – projekt s WCF službou
  - *WebServiceOAuth* – projekt pro správu uživatelů
  - *Dip.sln* – solution (řešení) sjednocující projekty

Pozn.: Pro spuštění souborů z adresáře *Db* je nutné mít nainstalované SQL Management Studio 2014. Ke spuštění souborů z adresáře *Dip* je potřeba vývojového nástroje Visual Studio 2015.