Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence

Jaromír KEJDA

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jaromír Kejda

Informatika

Název práce

Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence

Název anglicky

Using .NET Framework for data processing and securing data persistence

---

Cíle práce

Práce se zabývá problematikou zpracování dat v prostředí .NET. Prvním cílem práce je poskytnout přehled o možnostech .NET/C# v oblasti získávání, ukládání, zpracování a exportování dat. Dalším cílem je pak na základě těchto informací navrhnout a s využitím jazyka C# implementovat pokladní a skladový systém.

Metodika

Metodika práce je založena na analyticko-syntetickém přístupu. Na základě analýzy odborných informačních zdrojů budou popsány možnosti zpracování dat v prostředí .NET.

Ze získaných teoretických poznatků se bude dále vycházet při zpracování praktické části práce, která spočívá v návrhu a implementaci pokladního a skladového systému. Při návrhu a implementaci budou použity standardní metody softwarového návrhu.

Doporučený rozsah práce

35-40 stran

Klíčová slova

C#, Microsoft, .NET Framework, data, soubor, databáze, Visual Studio, WPF, LocalDB, MSSQL Server, XML

Doporučené zdroje informací

ALBAHARI, J. – ALBAHARI, B. C# 5.0 in a nutshell. Sebastopol: O'Reilly, 2012. ISBN 978-1-449-32010-2.

NAGEL, Christian. C# 2008: programujeme profesionálně. Brno: Computer Press, 2009. ISBN 978-80-251-2401-7.

SHARP, J. Microsoft Visual C# 2008: krok za krokem. Brno: Computer Press, 2008. ISBN 978-80-251-2027-9.

TROELSEN, A. Pro C# 5.0 and the .NET 4.5 framework. Berkeley: Apress, 2012. ISBN 978-1-4302-4234-5.

VIRIUS, M. *C# 2010 : hotová řešení.* Brno: Computer Press, 2010. ISBN 978-80-251-3730-7.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 07. 11. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 22. 2. 2017 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D. za inspirativní návrhy při upřesňování tématu této práce a dále za jeho vstřícnost a ochotu při konzultacích.

Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence

Souhrn

Hlavním cílem této práce je představit prostředky .NET Frameworku využitelné při implementaci desktopové aplikace v jazyce C#, která perzistentně ukládá data jak do souborů na samotném operačním systému Windows, tak do databáze Microsoft SQL Server. Navazujícím vedlejším cílem je na základě teoretické části navrhnout a implementovat pokladní a skladový systém s důrazem na využití souborů, databází a standardních knihoven pro běžné operace s daty. Práce se dále zabývá použitím různých formátů pro uložení dat, v projektu především XML a problematikou jejich zpracování. Kromě sledování výše zmíněných cílů, práce také hodnotí výslednou aplikaci a uvádí další návrhy na její zlepšení a rozšíření.

Klíčová slova: C#, Microsoft, .NET Framework, data, soubor, databáze, Visual Studio, WPF, LocalDB, MS SQL Server, XML

Using .NET Framework for data processing and securing data persistence

Summary

The first aim of this thesis is to introduce useful tools of .NET Framework for the implementation of a desktop application in C# which persistently saves data in a Windows local file system as well as in a database system Microsoft SQL. The second following aim is to design and implement a cash register and storage system with emphasis on using files, database, and standard libraries for common data operations as described in the theoretical part. Additionally, this thesis practically shows how to use different formats of data, in this project mainly XML, and explains the problematics of data processing. Apart from the above-mentioned aims, this thesis also gradually presents the pros and cons of some of the described approaches and produces a critical evaluation of the final application together with some ideas of possible future improvements.

Keywords: C#, Microsoft, .NET Framework, data, file, database, Visual Studio, WPF, LocalDB, MS SQL Server, XML

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

1 Úvod

V dnešním uspěchaném světě 21. století, jak by řekl klasik, není nic častějšího než změna. Ze všech stran se na nás v dnešní informační společnosti valí množství zpráv, kterým chceme nejenom porozumět, ale také je i zaznamenat, mezitím co se nám okolní svět mění pod rukama. V této dynamické době je o to složitější s daty pracovat tak, aby korektně odrážela okolní nestatický svět. Data, která jsou zobrazením informací, často ukládáme proto, aby nám v budoucnu pomohla odpovědět i na poměrně abstraktní otázky. Úspěšné fungování dnešních podniků, ať už soukromých či veřejných, je založeno na datech a interpretaci těchto dat. Na rozdíl od dob minulých se podniky nemohou již spoléhat na tradiční manuální záznam dat, neboť v mnoha situacích je data potřeba zaznamenávat spojitě, a to již není pochopitelně v silách žádného člověka. Proto se v současnosti spoléháme na počítače a počítačové programy, které tuto práci odvádějí za nás a je zřejmé, že kvalitní návrh, implementace, vhodná volba technologie a technických prostředků hraje velmi důležitou roli při ukládání a zpracování dat, přičemž je podstatné, aby data byla co nejméně náchylná k poškození, věrohodná, zabezpečená a především persistentní. Prostřednictvím této práce se snažím ukázat, že platforma .NET spolu s jazykem C# je vhodným nástrojem pro realizaci systémů, které jsou schopny efektivním způsobem nakládat s množstvím dat a naplňovat informační potřeby svých uživatelů.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je nejprve představit knihovny a vývojové nástroje platformy .NET a způsob jejich využití při persistentním ukládání a následném zpracování dat s využitím jazyka C#.

Vedlejším navazujícím cílem práce je využít získané teoretické znalosti knihoven, nástrojů a jazyka C# k návrhu a implementaci Pokladního a skladového systému (PASS), na kterém jsou demonstrované postupy dány do souvislosti. Práce poskytuje přehled tématiky vztahující se především k souborovému a databázově-relačnímu uložení dat a dále se soustředí na přístup k datům z různých datových zdrojů, stejně tak jako na práci s těmito daty.

2.2 Metodika

Metodika teoretické části práce je založena na studiu odborné literatury týkající se platformy .NET, konkrétně především oblasti práce se soubory a databázemi. Na základě analyticko-syntetického přístupu k odborné literatuře je uveden teoretický přehled knihoven, jejichž znalost je nezbytným předpokladem pro úspěšnou implementaci výsledné aplikace.

Metodika praktické části práce je založena na autorově volbě jednotlivých funkcionalit programu, které jsou zaznamenány prostřednictvím Use Case diagramu. Po provedení tohoto návrhu případů užití pomocí standardního modelovacího prostředku UML autor přejde k návrhu databázového schématu a datové normalizaci. Nakonec autor přistoupí k samotné implementaci softwaru s využitím standardních vývojářských nástrojů platformy .NET. Konečné publikaci aplikace předchází proces nasazení a testování.

3 Teoretická východiska

3.1 Úvod do problematiky zpracování dat

Jelikož se tato práce zabývá především problematikou uložení a zpracování dat, je namístě pojem data vysvětlit. Data je možné obecně chápat jako nějaké údaje, které byly získány pozorováním nebo měřením okolního světa. Konkrétně mohou mít data podobu číselnou, textovou, zvukovou nebo obrazovou. Zobrazují tedy nějakou skutečnost formálním způsobem, a díky tomu můžeme data zaznamenat, přenášet, zpracovat a uchovat.

Poté, co datům přisoudíme nějaký význam, hovoříme o tom, že jsme získali informaci. Informace jsou tedy interpretovaná data, které reprezentují jakoukoliv sdělitelnou vlastnost libovolného objektu. Zakladatel kybernetiky, Norbert Wiener (1963, s. 32), definoval informaci takto: „*Informace je název pro obsah toho, co se vymění s vnějším světem, když se mu přizpůsobujeme a působíme na něj svým přizpůsobováním.*“.

V dnešní době uživatelé požadují od počítačů vyřešení značně abstraktních úloh, přičemž ne všechny úlohy jsou řešitelné klasickými algoritmickými postupy. Proto některé programy kromě množiny dat, která je tzv. bázi dat, obsahují ještě bázi znalostí. Tyto systémy jsou schopné si poradit s kvalitativními informacemi a na základě odvozovacího mechanismu poskytovat expertní rady.

Již zmíněná abstrakce úloh klade na celkovou architekturu výpočetních systémů značné nároky. Aby bylo možné řešit složité abstraktní úlohy, tak je nutné nad holým počítačem vrstvovitě vytvořit virtuální počítače, které zastiňují detaily nižších vrstev a umožňují vyšší míru abstrakce. Výsledkem je značná složitost dnešních počítačů a nutnost existence více úrovní architektur, jako je architektura obvodů, stroje, operačního systému, vyšších programovacích jazyků, a nakonec architektura samotného programu, se kterým uživatel pracuje. Rozdíl mezi požadavky na abstraktní práci s počítačem a samotnou nízkoúrovňovostí uložení dat vytváří tzv. sémantickou mezeru neboli mezeru ve vyjadřování (Merunka, 2004, s. 13).

Již zmíněná potřeba abstraktního přístupu k práci s daty vedla postupně ke vzniku vyšších programovacích jazyků, jímž je i jazyk C#, a také ke vzniku složitých aplikačních programů (například databáze). Díky tomu se může uživatel vyjadřovat na vyšší úrovni abstrakce a je odstíněn od komplexity nižších vrstev počítače. Jak píše Merunka (2004, s. 13): *„Vyšší úrovně architektury používají abstraktní datové typy, lokální proměnné, složené datové typy, složité operace nad daty, paralelní procesy a jejich synchronizace a další složité vlastnosti.“,* což značně ulehčuje tvorbu komplexních softwarových produktů.

3.1.1 Zobrazení a způsob uložení dat v počítači

Pokud nyní opomineme starší způsoby organizace dat prostřednictvím papírových kartoték nebo později děrných štítků a přejdeme k dnešním číslicovým počítačům, tak je nutné pochopitelně zajistit, jak budou informace v počítači zobrazeny a organizovány. Klíčové je, že stejná data mohou mít mnoho různých interpretací – vždy záleží na použitém kódování, tzn. jakým způsobem je množina kódovaných objektů zobrazena na množinu kódových slov. Máme tedy nějaký vzor, což je hodnota datového typu, kterému přiřazujeme jeho obraz, což je paměťový úsek nul a jedniček. Datový typ hodnoty, kterou zobrazujeme, je určen množinou svých hodnot a množinou operací nad těmito hodnotami.

Data jsou v počítači uložena na nějakém paměťovém médiu a mohou mít buď povahu dat informačních (čísla, znaky apod.), nebo povelových (instrukce). Některá data jsou uložena pouze dočasně v nestálé, elektricky závislé RAM paměti, či v registrech procesoru. Tato data se označují jako data transientní (Hrach 2013, s. 11). Datům uloženým trvale v elektricky nezávislé paměti, nejčastěji na pevním disku, říkáme naopak data stálá neboli data perzistentní.

Důsledkem použití John von Neumannovy architektury počítače je, že v jedné paměti se nacházejí jak data, tak i programy. Chování počítače se proto nemění změnou jeho struktury, ale obsahem jeho paměti, což je zřejmě jeden z nejdůležitějších principů vN architektury. Samotná paměť je realizovaná jako posloupnost buněk pevné délky, které jsou adresovány svými pořadovými čísly. Tato skutečnost je ve vztahu k již zmíněné sémantické mezeře nejslabší stránkou architektury počítačů, protože toto řešení je příliš

primitivní. *„Pro funkci počítače by bylo mnohem výhodnější používat buňky proměnlivé velikosti podle ukládané hodnoty.“* (Merunka, 2004, s. 12)

3.1.2 Soubory a souborové systémy

Současné operační systémy využívají služeb souborových systémů, které organizují související data do souborů a ty jsou pak hierarchicky umístěny do adresářů. Dále souborové systémy určují fyzické uspořádání dat na paměťovém médiu. Soubory kromě samotných dat obsahují také ještě metadata, která nesou doplňující informace o souboru. Operační systémy poskytují programové rozhraní pro práci se soubory, které zahrnuje služby jako je otevření, zavření, čtení, zápis, mazání a souběžný přístup více uživatelů k souboru. Některé soubory obsahují speciální typ dat, tzv. instrukce, které může vykonat procesor počítače. Hovoříme poté o spustitelných binárních souborech. Některé soubory neobsahují přímo instrukce pro procesor, ale zápis kódu, který je poté interpretován příkazovým interpretem. Hovoříme zde o tzv. skriptech. (Veselý, 2013, s. 9, 10,17,18)

3.1.3 Problematika zpracování většího množství dat

S narůstajícím množstvím dat, která je nutné evidovat, vyvstala potřeba zavést do jejich organizace určitý systém a pravidla, neboť je zřejmé, že postupná neorganizovaná akumulace dat by vedla dříve nebo později k chaosu. Velmi často daná vytvářená datová množina neslouží pouze tomu, kdo ji vytvořil, ale také ostatním lidem a programovým prostředkům. Z již řečeného vyplývá, že je potřeba dodržovat určitý sjednocený postup při ukládání dat.

3.1.3.1 Agendové zpracování dat

Počátečním řešením nastíněné problematiky bylo organizovat související data do jednotlivých souborů. Tomuto způsobu se také někdy říká agendové datové zpracování. Agendou je zde myšlena rozsahově malá úloha, která je reprezentovaná nějakým množstvím souborů (Vostrovský, 2012, s. 6). Každý z těchto souborů má poté svou vnitřní strukturu, která může být libovolná. Tuto strukturu musí poté bezpodmínečně znát každý program, který chce s těmito daty pracovat.

Na soubor v rámci agendy je nahlíženo jako na množinu jednotlivých záznamů. Každý záznam reprezentuje nějakou entitu prostřednictvím vlastností této entity, které jsou

reprezentovány jednotlivými položkami záznamu. Množina záznamů v jednom souboru by se zpravidla měla vztahovat vždy jen k jednomu typu objektu.

Důsledkem použití agendového zpracování dat je nutnost existence příslušného datového zdroje ke každému programu (obr. 3.1). Dalším problémem je nulová integrovanost (propojení) jednotlivých agend, a proto se některá data musejí v různých agendách opakovat. Hovoříme o tzv. redundanci dat, která později vede ke komplikacím při aktualizaci dat.

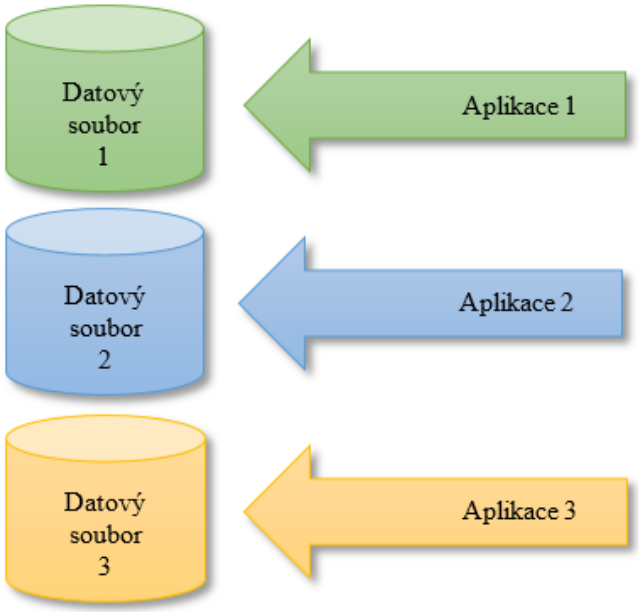
Problémy s nedostatečnou datovou integrovaností se snaží řešit tzv. integrované zpracování dat. Tento přístup posouvá agendové zpracování směrem k vyšší datové integrovanosti tím, že v nových agendách využívá již existující data agend stávajících. Stále jsou ovšem programy nadále závislé na struktuře obsahu souboru (Vostrovský, 2012, s. 10).

3.1.3.2 Databázové zpracování dat

V 70. letech 20. století bylo již zřejmé, že agendové zpracování dat je ve větších podnicích neudržitelné (Vostrovský, 2012, s. 10). Proto přichází databázové zpracování, při kterém programy již nepracují s mnoha různými soubory, ale pouze s jedním zdrojem dat – s tzv. bází dat neboli databází.

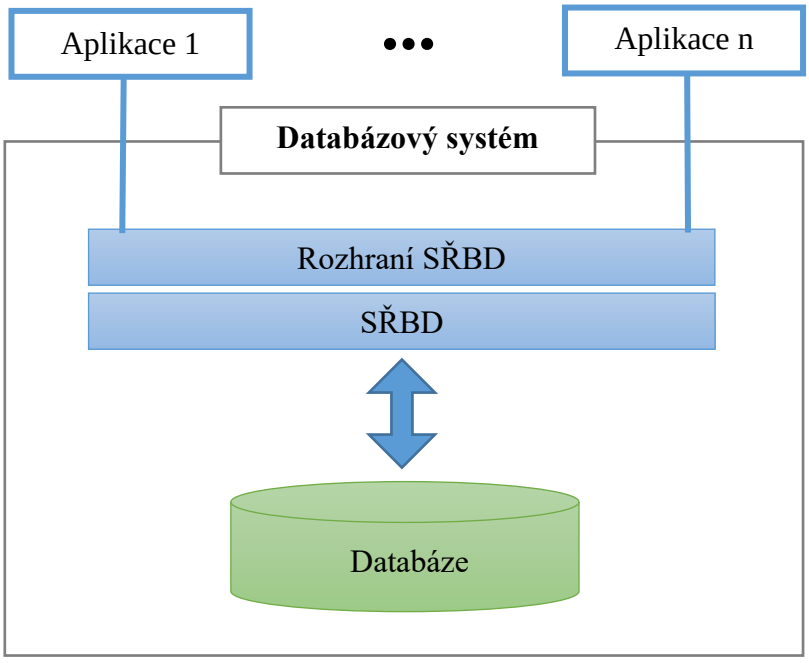
Systém řízení báze dat (anglicky DBMS – Database Management System), někdy také databázový stroj, je softwarové vybavení realizující komunikaci mezi aplikačním programem a databází. SŘBD odděluje uživatele od samotných dat a zajišťuje nezávislost dat na programu. Komunikačním rozhraním SŘBD je nejčastěji nějaký programovací jazyk, který slouží k formulaci operací nad databází. Samotná data jsou uložena v databázi a jsou řízena SŘBD. Tyto dvě části tvoří dohromady tzv. databázový systém, jak ukazuje obrázek 3.2. (Vostrovský, 2012, s. 10)

Obrázek 3.1 - Schéma agendového zpracování dat



Zdroj: (Vostrovský, 2012, s. 10), upraveno

Obrázek 3.2 - Princip databázového zpracování



Zdroj: (Vostrovský, 2012, s. 10), upraveno

3.2 Platforma .NET a jazyk C#

Při tvorbě komplikovaného a rozsáhlého software není dost dobře možné, aby tvůrci programů řešili příliš nízkoúrovňové detaily, neboť je potřeba, aby se soustředili na řešení zadaných úkolů, které jsou již samy o sobě často značně náročné. Rostoucí požadavky na abstraktní práci s počítači vedou k růstu nároků na samotné tvůrce softwaru, kteří se kromě tvorby samotné logiky aplikace a návrhem datové základny musí zabývat i tvorbou uživatelského rozhraní. Rozdělení aplikace na tyto tři vrstvy značně ulehčuje proces tvorby softwaru – hovoříme o vícevrstvé architektuře aplikace (n-tier application architecture). Platforma .NET poskytuje množství prostředků použitelných pro implementaci vícevrstevných aplikací. K tvorbě těchto aplikací platforma .NET poskytuje běhové prostředí, množství knihoven a díky CLI (Common Language Infrastructure) podporuje také mnoho programovacích jazyků, mezi něž patří kromě jazyka C# například jazyky Visual Basic, F#, Boo, IronPython, SmallTalk a mnoho dalších[1]. (MSDN, 2003)

3.2.1 Platforma .NET

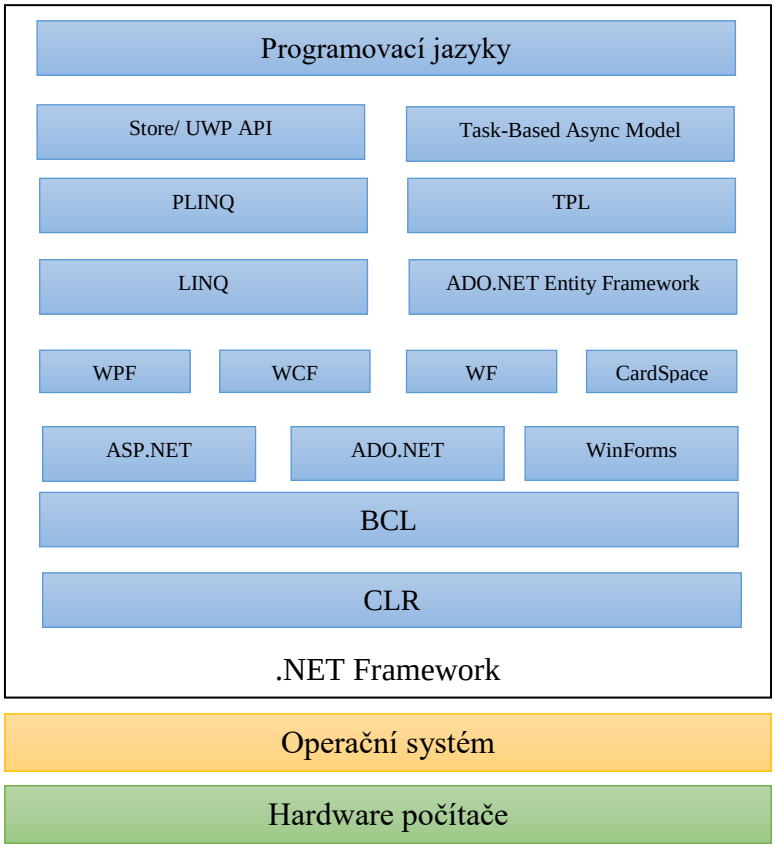
Microsoft .NET Framework je platforma pro provozování aplikací především na operačním systému Microsoft Windows. Jedná se o skupinu technologií, které jsou použitelné jak v desktopových, tak i webových aplikacích. Dle informací dostupných na MSDN byla tato platforma poprvé představena v roce 2001 a o rok později byla vydána první verze s označením 1.0.

Platforma .NET byla nejdříve koncipována jako další vrstva nad operačním systémem, ale od Windows Vista je jeho přímou součástí (Virius, 2010, s. 20). Obrázek 3.3 znázorňuje strukturu prostředí .NET. Na tomto diagramu je platforma .NET znázorněna jako nadstavba operačního systému, ale jak již bylo řečeno, v posledních verzích Windows je již jeho součástí. Základem .NET Frameworku je *společný běhový systém* – CLR (Common Language Runtime), který definuje CTS (Common Type System) a další součásti. O úroveň výše je BCL (Base Class Library), což je knihovna tříd, která obsahuje především základní typy, kolekce a třídy pro realizaci I/O operací. Nad BCL jsou další

[1] Více informací na adrese: https://msdn.microsoft.com/en-us/library/aa292164(v=vs.71).aspx

knihovny, například pro práci s uživatelským rozhraním, nebo webovými službami. Na nejvyšší úrovni jsou překladače programovacích jazyků. Tyto překladače překládají zdrojový kód do tzv. *mezilehlého jazyka* – IL (Intermediate Language), někdy také označovaného jako CIL či MSIL. Tento jazyk je relativně nízkoúrovňový a běží všude tam, kde je k dispozici běhové prostředí .NET. (Virius, 2010, s. 20-23)

Obrázek 3.3 - Základní struktura prostředí .NET



Zdroj: (Virius, 2010, s. 21), aktualizováno

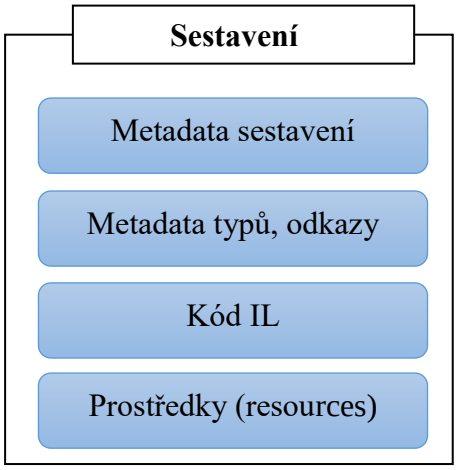
3.2.2 Jazyk C#

Jazyk C# je objektový, vysokoúrovňový, multiparadigmatický, komponentově orientovaný programovací jazyk navržený firmou Microsoft spolu s platformou .NET (MSDN, 2015). Tento programovací jazyk vychází z jazyka C, ze kterého přebírá syntaxi a také se inspiroval dalšími jazyky, například Javou. John Sharp (2008, s. 31) píše, že C#: *„Hraje velmi důležitou roli v architektuře Microsoft .NET Framework a někteří lidé v tomto pohledu poukazují na podobnou roli jazyka C při vývoji systému UNIX.“*

Původ názvu tohoto jazyka pochází z hudební notace, kde symbol křížku znázorňuje zvýšení noty o půl tónu. Z praktických důvodů se místo speciálního znaku „♯“ z hudební notace používá klasický znak mřížky „#“.

Pokud zkompilujeme program v již zmíněném jazyce C, dostaneme neřízený a teoreticky velmi rychlý kód, který je závislý na daném procesoru (Virius, 2010, s. 23). Na platformě .NET je výsledkem kompilace zdrojového kódu (například v jazyce C#) CIL kód, který je také relativně nízkoúrovňový (často se přirovnává k jazyku symbolických adres, JSA) a běží všude tam, kde je běhové prostředí .NET. Výsledkem kompilace je na platformě .NET tzv. *sestavení*, někdy také označováno jako *distribuční jednotka*. Samotné sestavení může mít podobu spustitelného souboru (.exe), nebo se může jednat o knihovnu (.dll) (Virius, 2010, s. 26).

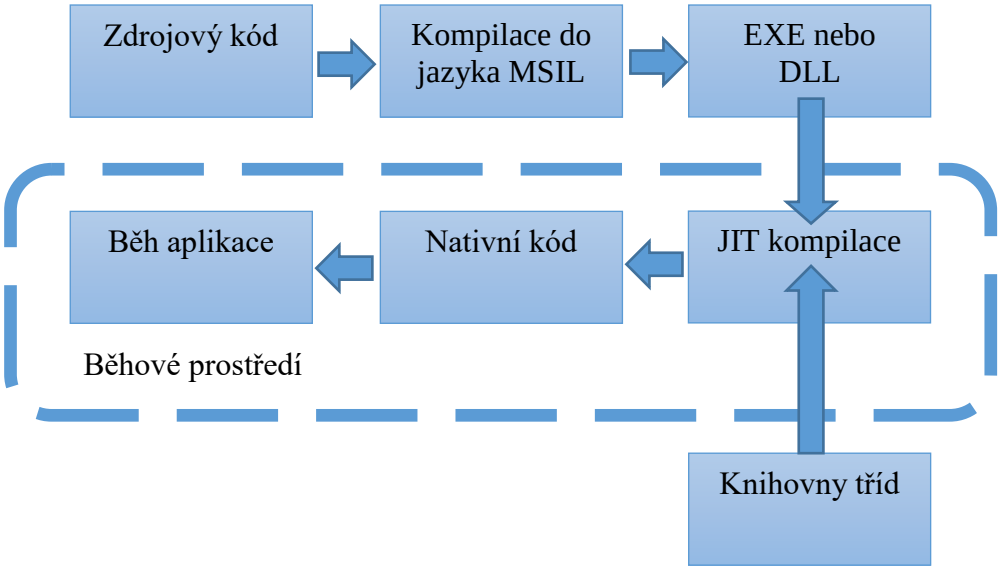
Obrázek 3.4 - Struktura sestavení



Zdroj: (Virius, 2010, s. 26), upraveno

Jak je vidět z obrázku 3.4, sestavení obsahuje samotný kód v mezilehlém (intermediárním) jazyce, metadata poskytující informace o jednotlivých typech, které jsou v tomto sestavení definovány a metadata obsahující obecné informace o sestavení jako celku – pro tyto metadata sestavení se často používá název *manifest*. Schéma na obrázku 3.5 popisuje proces kompilace a spuštění.

Obrázek 3.5 - Kompilace zdrojového kódu a jeho spuštění



Zdroj: (BĚHÁLEK, Common Language Runtime), upraveno

3.3 Data jako přímá součást aplikace

Data jsou v programech uložena v proměnných. Proměnná je jednoznačně identifikovatelné místo v paměti, které obsahuje určitou hodnotu nějakého datového typu. Proměnná je dočasné úložiště pro data, se kterými program pracuje. Překladač jazyka C# člověku srozumitelné názvy proměnných přeloží na paměťové adresy, kterým rozumí procesor počítače. Paměťová místa pro uložení dat přiřazuje programu operační systém, který využívá *systém virtuálního adresování* (Nagel, 2008, s. 400). Program poté nepracuje s fyzickým adresovým prostorem, ale s virtuálním, kde jsou paměťová místa číslována od nuly.

3.3.1 Paměť počítače a zásobník

Pochopení principu zásobníku je předpokladem pro obsáhnutí problematiky hodnotových a referenčních datových typů, proto se mu v této kapitole věnuji podrobněji.

V paměti procesoru se nachází zásobník (stack), což je abstraktní datová struktura sloužící pro dočasné ukládání dat. V zásobníku se s daty manipuluje způsobem LIFO (Last-In First-Out, poslední-uvnitř první-ven), kdy data uložena jako poslední jsou čtena jako první. Klasickým příkladem z učebnic programování je analogie zásobníku s hromadou talířů naskládaných na sobě, kdy talíře z hromady odebíráme od toho nejvrchnějšího. (Sharp, 2008, s. 193)

Do zásobníku se ukládají hodnoty proměnných, které nejsou datovými složkami objektů. Zásobník se plní daty od vrcholu zásobníku směrem k nižším adresovým místům. Operační systém udržuje speciální proměnnou, tzv. *ukazatel zásobníku*, která ukazuje na adresu poslední přidané položky. Z principu fungování zásobníku je zřejmé, že je přidělená paměť uvolňována v opačném pořadí, než byla přidělena. K uvolnění paměti dojde v okamžiku, kdy proměnná opustí obor své platnosti, tj. skoční metoda, kde byla proměnná deklarována, nebo překladač dorazí na konec bloku kódu obsahující deklaraci proměnné. (Nagel, 2009, s. 400)

3.3.2 Úvod do datových typů

Datový typ je množina dat a množina operací s těmito daty. Základním datovým typům programovacích jazyků se říká *primitivní typy* (Sharp, 2008, s. 56) a kromě ukládání samotných hodnot slouží ke skládání abstraktních datových typů. Jazyk C# má několik vestavěných primitivních datových typů.

V jazyce C# je většina primitivních typů takzvanými **hodnotovými datovými typy**. Hodnotové datové typy jsou poté reprezentovány prostřednictvím struktur (int, float, decimal, bool, char, double, uživatelsky definované struktury) a výčtů. Tyto typy mají společné to, že jejich hodnoty jsou uloženy v zásobníku (Sharp, 2008, s. 159). Při deklaraci hodnotového typu je na zásobníku alokován dostatečný paměťový prostor pro uložení hodnoty daného datového typu. Sharp (2008, s. 150) uvádí jako příklad deklaraci a inicializaci typu *int*. V tomto případě je proměnné kompilátorem přidělena paměť o velikosti 4 bajtů. Pokud této proměnné následně přiřadíme nějakou korektní hodnotu, zde celé číslo v rozmezí -2^{31} až $2^{31}-1$, bude toto číslo uloženo do vyčleněného paměťového místa.

Před použitím proměnné je v jazyce C# nutné jí nejdříve přiřadit počáteční hodnotu (v dokumentaci vedeno jako pravidlo *Definite Assignment Rule*), což není u jazyků C/C++ nutné a je to častým zdrojem špatně dohledatelných chyb, neboť neinicializovaná proměnná, jak uvádí standard jazyka C: *ISO/IEC 9899:TC3*, obsahuje nespecifikovanou hodnotu (pravděpodobně tu hodnotu, která byla na daném místě v paměti uložena předtím).

Protipólem hodnotových datových typů jsou **referenční datové typy**. Tyto typy jsou reprezentovány třídami, rozhraními a delegáty. Namístě je zde také upozornit na skutečnost, že typ *string* patří mezi tyto referenční typy, nikoliv mezi typy hodnotové či primitivní, protože je to alias třídy *System.String*. Referenční typy se liší od hodnotových tím, že místo samotné hodnoty obsahují odkaz (referenci) na nějaké paměťové místo. Samotný odkaz je uložen v zásobníku. Odkazované paměťové místo již ale není na zásobníku, ale na *řízené haldě* (managed heap). Surová paměť na haldě je převedena na objekt až v okamžiku vytvoření instance onoho objektu pomocí klíčového slova *new*. O uvolnění paměti rozhoduje *garbage collector*, který ji uvolní poté, co na ní zanikne poslední reference v kódu. Mechanismus referenčních typů je náhradou za používání klasických ukazatelů, které jsou známé například z jazyka C. Při práci s referenčními typy je třeba brát v úvahu, že při jejich kopírování dochází pouze ke kopírování odkazu na jeden a ten samý objekt na haldě. (Sharp, 2008, s. 149–155)

Je potřeba také chápat odlišné chování hodnotových a referenčních typů při jejich předávání jako parametrů metod. Při standardním předávání proměnné (bez modifikátoru *ref*) jako parametr metody se proměnná předává svojí hodnotou (passing by value). Pokud tedy uvnitř metody nějak tuto hodnotu modifikujeme, změny se mimo tuto metodu neprojeví. (Sharp, 2008, s. 156)

3.3.3 Jednoduché datové typy

Jednoduché (elementární) datové typy jsou typy, které jsou vestavěné v jazyku C# a slouží k uložení číselných, znakových, logických a výčtových hodnot a ke skládání složitějších datových struktur. Jednoduché datové typy obecně dělíme na ordinální: celé číslo (short, int, long, byte …), logická hodnota (bool), znak (char), výčtový typ (enum) a neordinální:

reálné číslo (float, double), prázdný datový typ (void). Elementární typy jsou v jazyce C# implementovány jako struktury. (Virius, 2010, s. 24)

3.3.4 Složené datové typy

Složené (agregované) datové typy jsou složené z hodnotových, nebo referenčních typů. Složené datové typy tedy mohou být tvořeny jak primitivními, tak dalšími složenými typy. Někdy hovoříme o tzv. *abstraktních datových typech*. Abstraktní typ je navenek reprezentován svým rozhraním a zapouzdřuje svoji vnitřní implementaci. Příklady složených datových typů dle MSDN jsou:

- pole – jazyk C# umožňuje vytvářet pole různých typů a dimenzí
- datový typ s*tring* a *object*
- kolekce – *Dictionary, List, Queue, SortedList, Stack …*
- uživatelsky definované třídy a struktury
- třídy vestavěných typů .NET Frameworku – *Random, Console, DateTime …*

Například datový typ *DateTime* je typ hodnotový, vestavěný a složený (má množství metod, vlastností a implementuje několik rozhraní).

3.4 Realizace perzistence dat

Perzistencí dat je myšleno přetrvání dat i po skončení daného procesu, který s těmito daty pracoval. Pro programy pracující s daty, která nejsou dočasného charakteru, je schopnost realizovat perzistenci dat klíčovou záležitostí. Platforma .NET poskytuje bohatou sadu prostředků pro trvalé uložení dat, ať už se jedná o ukládání pouhého textového řetězce či rovnou celého objektu. Pro pokročilejší správu velkého množství dat tato platforma také pochopitelně nabízí možnost využití služeb databázových systémů. Pro ukládání dat můžeme využít:

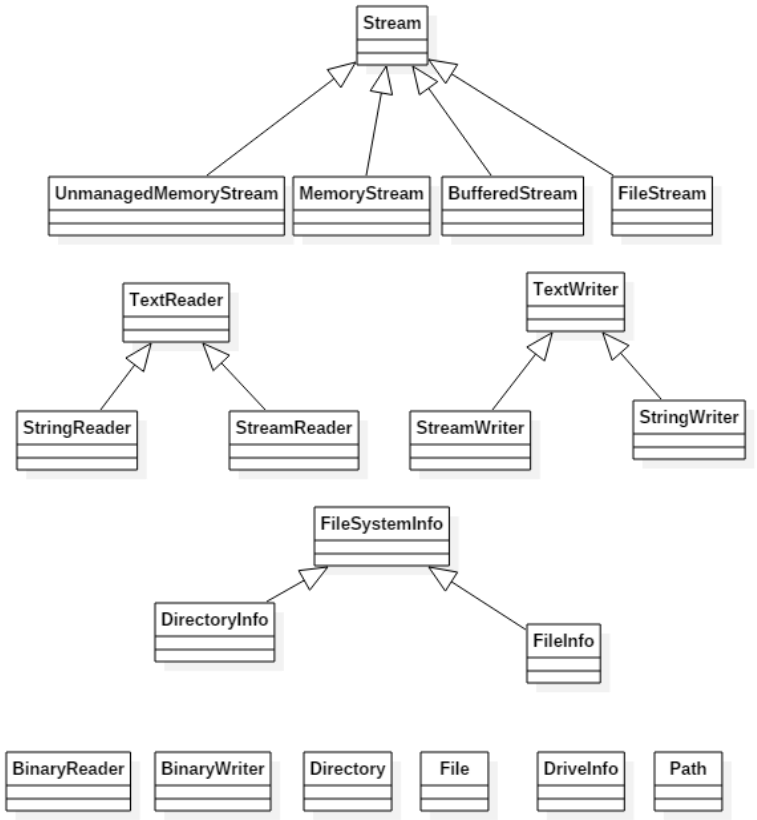
- Jednoduché textové soubory.
- Textové soubory se složitější vnitřní hierarchií – např. CSV, XML.
- Binární soubory.
- Komplexní databázové systémy (MS SQL Server, Oracle Database, MySQL...).

3.4.1 Soubory, vstupy, výstupy

Uložení dat do samostatných souborů je jedním z nejméně sofistikovaných způsobů trvalého uložení dat, které .NET Framework nabízí. Pro tento účel .NET Framework poskytuje množství tříd, jak pro práci se soubory, tak i s adresáři. Všechny tyto třídy sídlí ve jmenném prostoru *System.IO* (IO je zkratka input – output, tzn. vstup – výstup). Jmenný prostor *System.IO* definuje množství tříd, rozhraní, výčtových typů, struktur a delegátů.
V .NET Frameworku jsou všechny vstupně – výstupní operace realizovány pomocí datových proudů – streamů. Stream je objekt, který slouží k přenosu dat. (Troelsen, 2012, s. 754)

V uvedeném diagramu (obr. 3.6) jsou znázorněny pouze vztahy mezi třídami ze jmenného prostoru *System.IO*. Závislosti na třídách z jiných jmenných prostorů jsou zde pro přehlednost vynechány.

Obrázek 3.6 - Zjednodušený diagram tříd jmenného prostoru System.IO



Zdroj: vlastní

3.4.1.1 Výjimky

Při práci se soubory můžeme zranitelná místa našeho programu ošetřit buď aktivně pomocí podmínek, nebo zvolit pasivní přístup – tedy zachytávání výjimek. Druhý zmíněný postup se považuje za vhodnější, protože ne vždy víme, jaký druh chyby můžeme očekávat (je obtížné předpovídat chybu, kdy nám dojde paměť, uživatel zadá neexistující soubor…).

Při ošetřování kódu pomocí výjimek máme na výběr buď metodu založenou na posloupnosti bloků *try*, *finally*, kterou můžeme doplnit popřípadě o blok *catch,* nebo můžeme rovnou použít konstrukci *using*, která se za nás postará o vyřazení proměnné z oboru platnosti a nakonec uzavře i daný soubor. (Sharp, 2008, s. 113–117)

3.4.1.2 Vyřazení zdroje

Pokud pracujeme s omezenými zdroji, je více než nevhodné, aby se o jejich vyřazení staral destruktor, který se spustí, až když zanikne poslední reference na objekt a až běhové prostředí uzná destrukci za vhodnou. Omezené zdroje musíme proto uvolňovat ručně pomocí tzv. *vyřazovací metody* (disposal method). (Sharp, 2008, s. 251)

Pokud nebudeme uzavírat soubory, může k dojít k situaci, že nám dojdou popisovače souborů a nebudeme moci již další soubory otevřít. Dalším scénářem je, že při nečekaném ukončení programu by některé informace mohly zůstat v bufferu a nedošlo by k jejich zapsání do souboru, protože některé datové proudy kvůli zvýšení výkonu ukládají data postupně do vyrovnávací paměti a poté tyto data zapisují se zpožděním naráz. Pro vynucení zápisu dat z vyrovnávací paměti do úložiště se používá metoda *Flush()* daného zdroje. Tato funkce se volá také automaticky při uzavření datového proudu pomocí metody *Close()*. (Sharp, 2008, s. 252)

3.4.1.3 Příkaz Using

Příkaz using nabízí způsob, jak řídit dobu existence zdroje. Po skončení tohoto bloku je na daném zdroji volána metoda *Dispose()*, proto musí proměnná deklarovaná v tomto bloku implementovat rozhraní *IDisposable*.

Toto řešení se snáze rozšiřuje o vyřazení dalších zdrojů, nezasahuje do logiky programového kódu, zabraňuje nutnosti opakovat kód a vyřazuje proměnnou deklarovanou uvnitř bloku *using* z oboru platnosti. (Sharp, 2008, s. 253)

3.4.1.4 Třída FileStream

Třída *FileStream* pracuje s daty v binární podobě a zpřístupňuje základní funkcionalitu pro práci se soubory, jako je otevření, zavření, změna pozice v proudu dat a čtení/zápis bajtů nebo pole bajtů. Práce se zdrojem dat, jako s posloupností bajtů, může být poněkud neobratná, a proto je možné použít objekt typu *FileStream* jako podkladový proud pro jinou třídu, typicky pro *StreamWriter* a *StreamReader*. Díky jejich přetíženým metodám, které vstup převedou na bajty a předají je podkladovému proudu, poté můžeme zapisovat rovnou například textové, nebo číselné údaje. (Troelsen, 2012, s. 755)

3.4.1.5 Třídy StreamWriter a StreamReader

Tyto třídy jsou vhodné pro zápis textových dat, defaultně v kódování Unicode. Mezi podstatné metody třídy *StreamWriter* patří *Close()*, *Flush()*, *Write()* a *WriteLine()*. Jednou z vlastností třídy *StreamWriter* je vlastnost *autoflush*, která při nastavení na *true* zajistí, že po každém zápisu do souboru bude zavolána metoda *Flush()* automaticky.

Třída *StreamReader* slouží pro čtení ze souboru a obsahuje mimo jiné metodu *ReadLine()*, která vrací přečtený celý řádek, pokud se dostane na konec souboru, tak vrací *null*. Metoda *ReadToEnd()* přečte celý soubor a vrátí ho jako jeden řetězec. (Troelsen, 2012, s. 756)

3.4.1.6 Binární serializace

Pokud bychom chtěli uložit do souboru nejenom hodnotu nějakého primitivního typu, ale rovnou celý objekt, museli bychom ukládat jednotlivé datové složky objektu jeden po druhém. Pokud by náš objekt obsahoval další složené typy, byli bychom nuceni ukládat i datové složky všech těchto typů, což by bylo velice zdlouhavé a komplikované. Tento problém je za nás již na platformě .NET vyřešen pomocí tzv. *serializace.*

Serializace je tedy proces, kdy je stav objektu převeden do binární nebo XML podoby a v této podobě je uložen. Virius (2010, s. 183) píše, že pochopení samotného principu

serializace vyžaduje znalost poměrně složitých algoritmů z teorie grafů, a tudíž je tato problematika již mimo rozsah této práce.

Stejně, jako u třídy *BinaryWriter*, je výsledný soubor pro člověka nečitelný. Serializace se dá použít například pro uložení stavu programu, uložení nastavení programu, nebo pro výměnu dat s jiným programem. Procesu opětovnému načtení serializovaných dat se říká deserializace. Třída, kterou chceme serializovat musí být označena aribute*m [Serializable]*. O samotnou serializaci a deserializaci se stará třída *System.Runtime.Serialization. Formatters.Binary.BinaryFormatter.* Virius (2010, s. 183)

3.4.1.7 XML serializace

Alternativou k binární serializaci je XML serializace prostřednictvím třídy *XmlSerializer,* která se nachází ve jmenném prostoru *System.Xml.Serialization*. Tento způsob uložení dat má však několik omezení, jak píše Virius (2010, s. 188):

- Není možné ukládat sítě zanořených objektů různých typů.
- Ukládají se pouze veřejně přístupné datové složky (pro čtení i zápis).
- Třída, kterou chceme uložit, musí mít veřejně přístupný bezparametrický konstruktor (struktura ho má implicitně).

3.4.1.8 Třídy XmlReader a XmlWriter

Jak už napovídá jejich název, slouží tyto třídy pro čtení a zápis dat ve formátu XML. Třídy *XmlReader* a *XmlWriter* se nacházejí ve jmenném prostoru *System.Xml*. Princip těchto tříd je založen na sekvenčím čtení jednotlivých uzlů. Důsledkem tohoto přístupu je, že není nutné v paměti uchovávat všechny uzly najednou, ale stačí si pamatovat pouze uzel na aktuální pozici v souboru. (Albahari, 2012, s. 458, 467)

3.4.1.9 Třída XmlDocument

Třída *XmlDocument* slouží stejně jako *XmlReader* a *XmlWriter* ke čtení a zápisu dat ve formátu XML. Na rozdíl od nich ale nabízí odlišný přístup, který je založen na načtení celého zdroje dat do paměti a následně jeho procházení. (Albahari, 2012, s. 473)

3.4.2 Práce s daty na databázovém serveru

Platforma .NET poskytuje množství tříd pro práci s databázemi. Největší podporu má pochopitelně databázový server od Microsoftu – **MS SQL Server**. Pro potřeby malých a nenáročných databází postačuje verze **MS SQL Server Express**, která je dostupná zdarma, ale s určitými omezeními[2], která se týkají velikosti databáze, použitelné operační paměti a počtu procesorů. Tato práce se bude zabývat verzí tohoto serveru z roku 2014. Kromě MS SQL Serveru je možné v prostředí .NET využívat služeb i jiných databází, například Access, Oracle Database a MySQL.

Zdrojem informací týkajících se jednotlivých popisovaných SW produktů jsou příslušné stránky těchto programů na webu MSDN a také dostupné související návody pro vývojáře od společnosti Microsoft.

S ohledem ke zvolené databázi musíme používat příslušného *datového poskytovatele* (data provider) – jedná se o skupinu tříd pro práci s danou databází. Názvy těchto tříd pro odlišné databázové systémy se zpravidla liší pouze svou předponou. (Sharp, 2008, s. 455)

Platforma .NET poskytuje pro práci s databázemi knihovnu ADO.NET (ActiveX Data Object for .NET). Fungování knihovny ADO.NET je založeno na již zmíněných poskytovatelích. Každý databázový systém má poté svého vlastního poskytovatele, pomocí kterého se navazuje spojení s databází a provádějí se příkazy dotazovacího jazyka SQL (Structured Query Language). Jak píše Sharp (2008, s. 455), práce s knihovnami jednotlivých poskytovatelů se zásadně neliší a díky velké míře abstrakce je snadné přenést dané řešení i na jiný databázový systém.

Již zmíněná databáze **MS SQL Server Express** je dle MSDN omezena pouze podporou jednoho procesoru (může být i vícejádrový) a jedna instance serveru může používat nanejvýš 1 GB operační paměti. Dále je maximální velkost jedné databáze omezena na

[2] Podrobnější informace například zde: https://msdn.microsoft.com/cs-cz/sqlserver2014express.aspx

velikost 10 GB a nepodporuje službu SQL Server Agent[3], nebo omezení výkonu databáze při nadměrné zátěži.

Další možnou alternativou od společnosti Microsoft je databáze **SQL Express LocalDB**. Jedná se o odlehčenou verzi databáze MS SQL Server Express a je cílena především na vývojáře. Mezi její výhody patří menší velikost na disku a nulová nutnost její konfigurace. Odpadá tedy instalace poměrně těžkopádného SQL serveru a jeho nastavování.

Prostředek SQL Server CE, nebo také jinak označován jako **Microsoft SQL Server Compact**, je nejminimalističtější databází od společnosti Microsoft, jejíž vývoj je již ukončen. Svým charakterem by se dala přirovnat k Access databázi. Je to databáze, která je oproti ostatním zmíněným databázím značně omezená. Nepodporuje například spouště (triggers), pohledy a uložené procedury.

Nejprimitivnější přístup k databázovému serveru poskytuje konzolový nástroj **SQLCMD.exe**, který je volitelnou součástí například instalace SQL Server Express. Pomocí tohoto nástroje je možné se připojit k exitující instanci databáze a provádět nad ní databázové operace.

Pokročilejším nástrojem, který je také volitelnou součástí instalace SQL Server Express, je **SQL Server Management Studio**. Jedná se o grafickou aplikaci pro správu databáze. Umožňuje například grafický návrh struktury databáze, provádění dotazů, tvorbu grafické reprezentace databázového schématu a zálohování databáze.

Za zmínku stojí také program **Sql Server Configuraton Manager**, který je součástí instalace SQL Serveru a slouží pro spuštění a vypínání databázových služeb (samotná služba databáze, SQL Reporting Services, SQL Server Agent, SQL Server Browser).

3.4.2.1 Připojení k databázi a dotazy na data

Při instalaci SQL Serveru Express je nutné zvolit název instance databáze a nastavit přístupová práva. Instance databáze je kromě svého jména ještě určena názvem počítače.

[3] Komponenta SQL Serveru sloužící pro plánování a vykonávání automatizovaných úkolů (jobs).

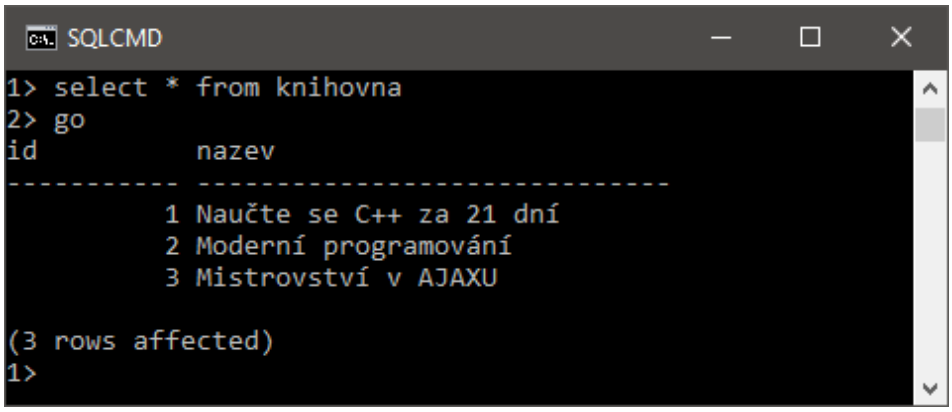
Při přihlašování k databázi si vystačíme pro potřeby tohto projektu s Windows Authentification, což je autentifikace uživatelským účtem systému Windows.

V případě LocalDB je možné instanci databáze vytvořit prostřednictvím nástroje *SqlLocalDB.exe* nebo jí vytvořit prostřednictvím Visual Studia.

Jedním ze způsobů připojení k databázi je využití nástroje SQL Server Management Studio, které při práci s databází nabízí značný komfort v podobě grafického uživatelského rozhraní.

Další možností, jak se připojit k databázi, je využít nástroj *SQLCMD.exe*. Poté již můžeme pomocí příkazů pro SQL Server zvolit danou databázi a dále s ní pracovat. Následující obrázek ukazuje spuštění jednoduchého dotazu nad tabulkou *knihovna*:

Obrázek 3.7 - Prostředí SQLCMD



Zdroj: vlastní

K databázi se můžeme připojit také pomocí Visual Studia. V okně *Server Explorer* je možné se připojit k databázi pomocí volby *Connect to Database* a nastavení připojení.

K databázi se často připojujeme přímo z aplikace, kterou vyvíjíme. K připojení k databázi potřebujeme znát *connection string*, což je připojovací řetězec. Připojovací řetězec různých databázových systémů se může lišit. Jelikož při nasazení aplikace může mít databázový server jiné umístění než server testovací, je dobrým zvykem tento připojovací řetězec ukládat do konfiguračního souboru aplikace (nikoliv přímo do kódu) a odtud ho v aplikaci načítat.

3.5 Prostředky pro zpracování dat

Abychom mohli z dat získávat informace, efektivně a efektně je prezentovat, ukládat, přenášet a v podstatě poskytovat správná data správným lidem ve správný čas, je potřeba tato data vhodným způsobem nejenom organizovat do souborů či databázových tabulek, ale také v nich umět vyhledávat, třídit je a vhodným způsobem je transformovat.

Platforma .NET obsahuje nepřeberné množství prostředků pro tyto úkoly. Často se ale stává, že potřebujeme provést stejnou operaci nad určitými daty, přičemž se liší jen zdroj těchto dat. Můžeme například chtít vyhledávat jak v samotné databázi pomocí jazyka SQL, tak i v souboru či nějaké kolekci. Řešením může být přístup, kdy ručně vytvoříme kód pro každý zdroj dat zvlášť. Vedlejším efektem tohoto přístupu je, že je tento kód příliš pevně svázaný se strukturou dat a také je často velmi zdlouhavé ho napsat.

3.5.1 Jazyk LINQ

Výsledkem snah zabránit komplikacím při dotazování se na data z různých zdrojů a usnadnit práci vývojářům je deklarativní dotazovací jazyk LINQ (Language Integrated Query), dostupný v .NET Frameworku od verze 3.5. Tento jazyk se inspiroval již zmíněným funkcionálním jazykem SQL, který je abstrahován od samotné implementace dat. Vývojář se poté nemusí starat, jak jsou data v databázi uložena a jak se dají získat, ale stačí mu napsat dotaz v tomto jazyce a o zbytek se již postará běhové prostředí a SŘBD. Jazyk LINQ je syntakticky velice podobný jazyku SQL. Na rozdíl od něj nabízí mnohem větší flexibilitu, protože umí pracovat s mnoha logickými datovými strukturami a není limitován pouze na databázový systém. (Sharp, 2008, s. 343)

Jazyk LINQ je použitelný nad datovými strukturami, které implementují rozhraní *System.Collections.IEnumerable*. Prakticky se může jednat o pole, kolekci, objekt typu *DataSet* a o mnoho dalších entit.

Jazyk LINQ byl ve verzi .NET Framework 3.5 představen spolu s mnoha novými jazykovými konstrukty, které umožnily tento jazyk implementovat a používat. Troelsen (2012, s. 440) jako příklady těchto konstruktů uvádí:

- implicitně typované proměnné (klíčové slovo *var*)
- objektový inicializátor (object initialization syntax)
- lambda výrazy
- rozšiřující metody
- anonymní typy

Jazyk LINQ používá k tvorbě dotazů několik operátorů v uvedeném pořadí: *from, in, where, orderby* a *select.* Dotazy napsané pomocí těchto operátorů jsou poté přeloženy na volání odpovídajících rozšiřujících metod beroucí jako parametry lambda výrazy.

From slouží pro označení proměnné, která reprezentuje jeden záznam v kolekci. *In* slouží pro zvolení zdroje dat. *Where* označuje podmínku pro výběr dat. *Orderby* umožňuje třídění a *Select* nakonec určí, která data chceme z výsledku vybrat. Pořadí těchto operátorů se liší od pořadí odpovídajících operátorů v SQL. (Troelsen, 2012, s. 449)

Výsledek dotazu můžeme uložit do objektu generického typu *IEnumerable<T>* nebo do implicitně typované proměnné označené klíčovým slovem *var*.

3.5.1.1 LINQ To Objects

LINQ to Objects je provider poskytující metody pro práci s poli a kolekcemi. Následující příklad demonstruje použití dotazovacího jazyka LINQ pro získání jmen, která mají délku přesně sedm znaků.

Výpis 3.1 - LINQ To Objects

```
            string[] jmena = { "Jaromír","Karel","Jana","Markéta"};
            IEnumerable<string> vysledekDotazu = from j in jmena
                                                 where (j.Length == 7)
                                                 select j;

            foreach (string s in vysledekDotazu)
            {
                Console.WriteLine(s);
            }
```

Výpis 3.2 - Výstup programu

```
Jaromír
Markéta
```

Předchozí příklad je možné zapsat ekvivalentně také přímo pomocí rozšiřujících metod a lambda výrazů:

Výpis 3.3 - Přímý zápis LINQ dotazu

```
IEnumerable<string> vysledekDotazu =
jmena.Where(j => (j.Length == 7)).Select(j => j);

foreach(string s in vysledekDotazu)
{
    Console.WriteLine(s);
}
```

Jazyk LINQ můžeme použít také pro práci s kolekcemi, například generickou třídou List<T>, nebo můžeme také jako zdroj dat použít anonymní typ, jak ukazuje následující kód:

Výpis 3.4 - LINQ a anonymní objekty

```
var anonymníObjekt = new[]
{
    new {skupina=1,nazev="Jablko" },
    new {skupina=2,nazev="Mrkev"},
    new {skupina=2,nazev="Kapusta"},
};

var vysledekDotazu = from o in anonymníObjekt
                     where (o.skupina == 2)
                     select o;

foreach(var s in vysledekDotazu)
{
    Console.WriteLine(s.nazev);
}
```

Výpis 3.5 - Výstup programu

```
Mrkev
Kapusta
```

V předchozím příkladu jsme použili anonymní typ pro uložení anonymního objektu, který reprezentuje ovoce a zeleninu, které jsou roztříděny do skupin. Pomocí jazyka LINQ poté vybíráme pouze názvy produktů z kategorie č. 2 (zelenina). Některé dotazy jazyka LINQ mohou vracet objekty ne snadno určitelných typů a někdy také i anonymní objekty. Proto může být vhodné pro uložení výsledku dotazu použít spíše implicitně typovanou proměnnou než konkrétní datový typ. Další výhodou tohoto přístupu je, že při změně

dotazu (což může mít za následek změnu návratového datového typu) není nutné měnit typ proměnné do které se ukládá výsledek dotazu.

3.5.1.2 LINQ To XML (XLINQ)

LINQ to XML je provider poskytující metody pro práci s XML dokumenty, nacházející se ve jmenném prostoru *System.Xml.Linq*. Jak píše Albahari (2012, s. 423), jedná se o nadstavbu relativně nízkoúrovňových tříd *XmlReader* a *XmlWriter*.

Následující kód ukazuje použití LINQ to XML pro čtení a výpis dat ze souboru *vstup.xml*:

Výpis 3.6 - LINQ To XML

```
XElement xDoc = XElement.Load(File.Open("vstup.xml",FileMode.Open));
IEnumerable<XElement> obchod = from el in xDoc.Elements("polozka")
                               select el;
foreach(var item in obchod)
{
     Console.WriteLine(item.Value);
}
```

Výpis 3.7 - Výstup programu

```
Jablko
Pomeranč
```

Výpis 3.8 - Obsah souboru vstup.xml

```
<?xml version="1.0" encoding="utf-8"?>
<obchod>
<polozka>Jablko</polozka>
<polozka>Pomeranč</polozka>
</obchod>
```

Třída *XElement* slouží jako reprezentace kořenového uzlu XML dokumentu. Prostřednictvím její vlastnosti *Elements* je možné získat kolekci jednotlivých elementů.

3.5.1.3 LINQ To SQL (DLINQ)

LINQ to SQL je provider poskytující metody pro práci s daty na databázovém serveru prostřednictvím jazyka LINQ. Tento provider překládá LINQ dotazy do databázového dotazovacího jazyka T-SQL (Transact Structured Query Language) a posílá tento převedený dotaz ADO.NET poskytovateli k vykonání.

Alternativou k LINQ to SQL je *Entity Framework*, který poskytuje robustnější řešení, především při práci s databázemi od různých společností (Troelsen, 2012, s. 912).
Pro použití LINQ to SQL je nutné nejprve do projektu přidat speciální třídy prostřednictvím volby v solution exploreru: *Project/Add new Item/LINQ to SQL Classes*. Z připojené databáze poté pomocí okna server explorer zvolíme požadované databázové tabulky jejich přetažením do oblasti designeru. Tímto jsou do projektu přidány všechny potřebné třídy.

Prostřednictvím automaticky vygenerovaných tříd je možné po inicializaci kontextového objektu pracovat s databází s využitím jazyka LINQ, jak demonstruje následující příklad, který pracuje s databází *Obchod2016* a tabulkou *sklad (**id**, nazev)*.

Výpis 3.9 - LINQ To SQL

```
        string connectionString = @"Data Source=DESKTOP\SQLEXPRESS;Initial
        Catalog=Obchod2016;Integrated Security=True";
        LinqToSqlDataContext db = new LinqToSqlDataContext(connectionString);
        var produkty = from p in db.sklads
                       select p.nazev;

        foreach (string nazev in produkty)
        {
            Console.WriteLine(nazev);
        }
```

Tabulka 3.1 - Obsah tabulky sklad

id	nazev
1	Chléb konzumní
2	Jablko
3	Pomeranč
4	Jahody

Výpis 3.10 - Výstup programu

```
Chléb konzumní
Jablko
Pomeranč
Jahody
```

Pomocí LINQ To SQL je možné data i měnit, jak ukazuje následující příklad, který mění hodnotu atributu *nazev* u prvního záznamu v databázi z hodnoty „Chléb konzumní“ na „Bageta šunková“:

Výpis 3.11 - LINQ to XML, změna dat v databázi

```
string connectionString = @"Data Source=DESKTOP\SQLEXPRESS;Initial
Catalog=Obchod2016;Integrated Security=True";
LinqToSqlDataContext db = new LinqToSqlDataContext(connectionString);
sklad polozka = (from p in db.sklads
                 where (p.id == 1)
                 select p).Single();
polozka.nazev = "Bageta šunková";
db.SubmitChanges();
```

Metoda *Single()* zde slouží k návratu pouze jednoho elementu (nikoliv jejich kolekce). Provedené změny je nutné nakonec potvrdit metodou *SubmitChanges()*. Do databáze často také potřebujeme přidávat nové záznamy, což se dá prostřednictvím LINQ to SQL realizovat následovně:

Výpis 3.12 - LINQ To XML, přidání záznamu

```
string connectionString = @"Data Source=DESKTOP\SQLEXPRESS;Initial
Catalog=Obchod2016;Integrated Security=True";
LinqToSqlDataContext db = new LinqToSqlDataContext(connectionString);

sklad novyZaznam = new sklad();
novyZaznam.id = 5;
novyZaznam.nazev = "Mouka hladká";
db.sklads.InsertOnSubmit(novyZaznam);

try
{
    db.SubmitChanges();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

4 Vlastní práce

Pokladní a skladový systém, zkráceně PASS, je aplikace navržená a implementovaná za cílem demonstrovat možnosti platformy .NET a jazyka C# tak, jak byly popsány v teoretické části této práce. Dalším cílem tohoto projektu bylo využít standardní postupy softwarového návrhu, které byly podpořeny primárně vývojovými nástroji, které jsou svázány s platformou .NET.

Kapitola 4.1 podává základní charakteristiku navrženého projektu, kapitola 4.2 se následně zabývá návrhem a implementačními podrobnostmi aplikace. Kapitola 4.3 vysvětluje způsob instalace aplikace, která je dostupná na přiloženém optickém nosiči a nakonec kapitola 4.4 popisuje jednotlivé funkcionality hotové aplikace.

4.1 Charakteristika projektu a jeho významné části

PASS je softwarovým produktem, který kvůli omezenému rozsahu a zaměření této práce není integrován s elektronickou evidencí tržeb (EET) a tudíž se nejedná o software použitelný v praxi v aktuální podobě, ačkoliv je možné program PASS o tuto funkcionalitu rozšířit. Celý projekt je kvůli přehlednosti rozdělen z pohledu uživatele do několika logických celků:

- SKLAD – Sklad slouží k evidenci jednotlivých skladových položek.
- POKLADNA – Pokladna umožňuje prodávat jednotlivé položky ze skladu.
- MANAGEMENT – Management slouží pro zobrazení elementárních statistik programu, nastavení uživatelských účtů a správě účtenek.
- PROFIL – Tato část aplikace se vztahuje k aktuálně přihlášenému uživateli a umožňuje mu změnit si své heslo, nebo se odhlásit.
- NASTAVENÍ – Administrátor aplikace prostřednictvím této sekce volí typ databáze a nastavuje aplikaci jako celek.

4.2 Návrh a implementace projektu

Projekt byl implementován prostřednictvím programovacího jazyka C# 6.0 a .NET Frameworku 4.5.2 s využitím technologie WPF (Windows Presentation Foundation). K implementaci grafického rozhraní byl použit značkovací jazyk XAML. Pro vývoj aplikace bylo použito Microsoft Visual Studio 2015 Community Edition a pro vývoj databáze SQL Server Management Studio. Pro podporu vývoje jsem dále využil program Tortoissse SVN, pro práci s grafikou program Paint.NET a pro tvorbu instalačního programu byl použit doplněk Microsoft Visual Studio 2015 Installer Project Extension. Program využívá standardních knihoven platformy .NET, přičemž žádné knihovny třetích stran nebyly použity, kromě bootstrapperu pro LocalDB 2014, který je k dispozici volně ke stažení na portálu Github[4].

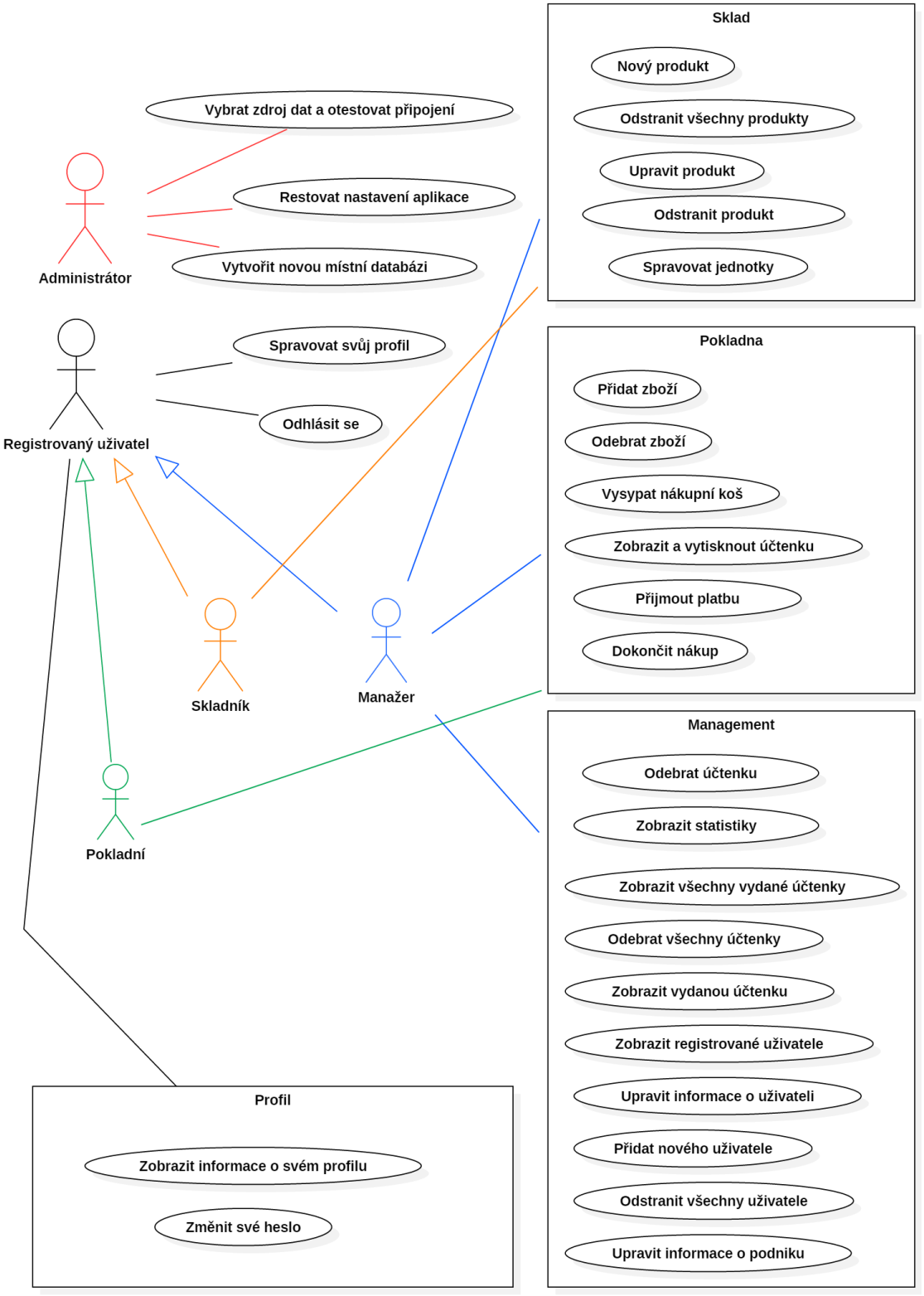
Klíčovou vlastností projektu je možnost provozovat PASS bez nutnosti instalace plnohodnotného SQL Serveru, který může vyžadovat řádově několik GB volného místa a jeho konfigurace je příliš pokročilá pro laického uživatele. Tohoto bylo možné dosáhnout pomocí rozšířeného nastavení aplikace, kdy si uživatel volí mezi použitím Microsoft SQL LocalDB 2014, nebo plnohodnotnou verzí MS SQL Serveru. Společnost Microsoft v době psaní této práce nenabízí vlastní bootstraper pro instalaci produktu LocalDB 2014 na klientský počítač v rámci Installer Project a proto jsem použil bootstrapper od třetí strany. Ve výsledku se při absenci programu LocalDB 2014 při instalaci PASS tento produkt nainstaluje automaticky.

4.2.1 Diagram případů užití

Návrh programu vychází z diagramu případů užití, jak ukazuje obrázek 4.1. Pro větší přehlednost schématu jsem jednotlivé uživatelské cíle rozdělil do několika skupin: Sklad, Pokladna, Management a Profil.

[4] https://github.com/kjbartel/SqlLocalDB2014-Bootstrapper

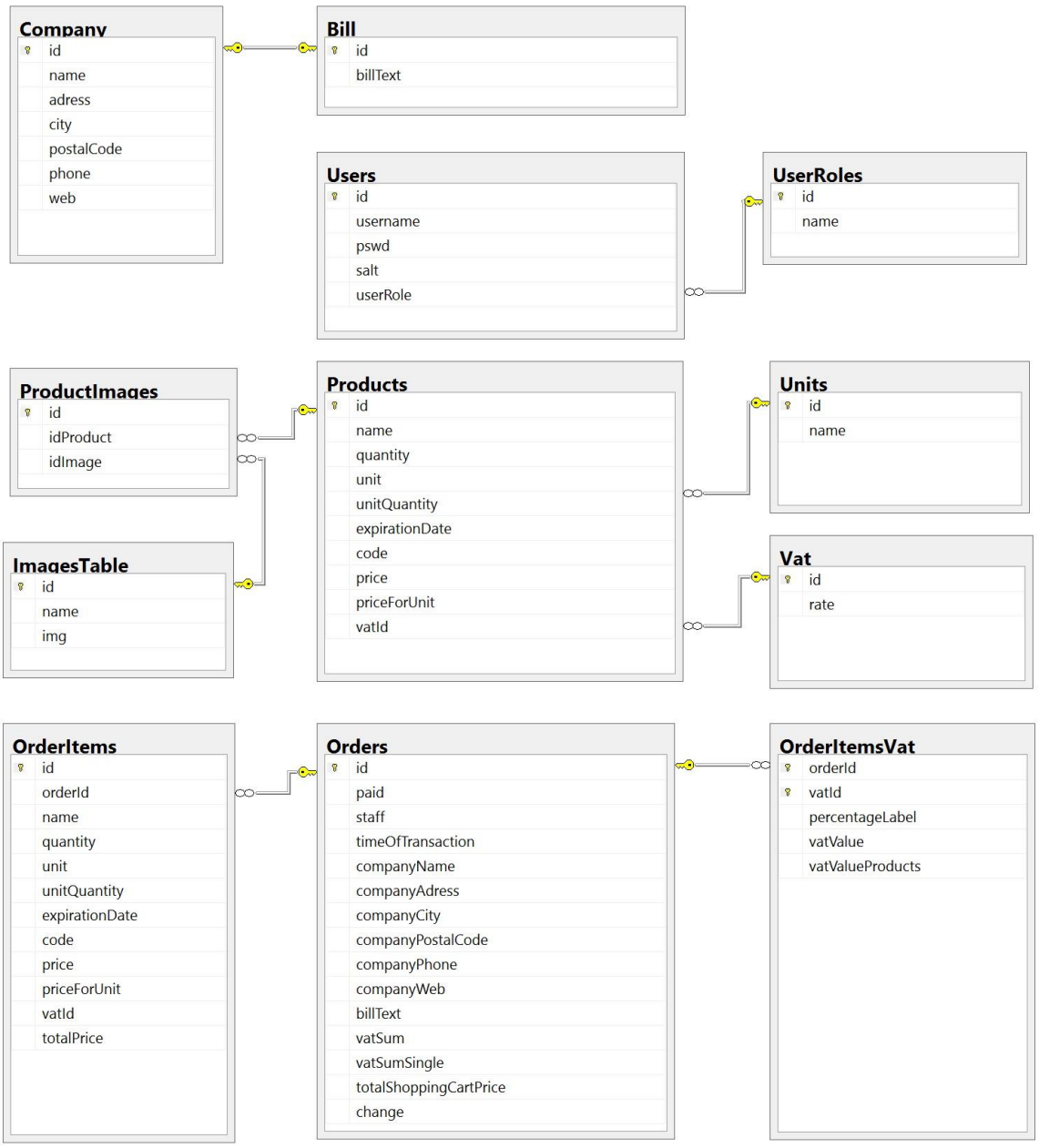
Obrázek 4.1 - Use Case diagram



4.2.2 Databázový návrh a normalizace

Ukládání dat je realizováno buď prostřednictvím databáze LocalDB, nebo MS SQL Serveru. Následující obrázek představuje použité databázové schéma, které bylo vygenerováno pomocí Microsoft SQL Server Management Studia. Většina atributů tabulek je samopopisných a proto se následující vysvětlení věnuje pouze vztahům a atributům, které by mohly být dle mého názoru pro čtenáře nejasné.

Obrázek 4.2 - Databázové schéma



Tabulka *Company* představuje informace o podniku. PASS podporuje pouze přiřazení jedné společnosti k jedné instalaci programu, a tudíž obsahuje tabulka *Company* přesně jeden záznam. Z tohoto důvodu je již zbytečná komplikace provádět převod této tabulky do 3NF a ponechal jsem jí pouze ve 2NF (tabulka *Company* obsahuje totiž tranzitivní závislost *city* na *postalCode*).

Tabulka *Bill* představuje dodatečné informace, které se tisknou na účtenku a které souvisí s danou společností. Tato tabulka je ve vztahu 1:1 s tabulku *Company* a pouze zpřehledňuje návrh. Tato tabulka by se dala dále hypoteticky použít například pro uložení firemního loga, které by bylo upravené pro tisk na účtenku.

Tabulka *Users* představuje jednotlivé uživatele systému. Každý uživatel má *username* – uživatelské jméno, *pswd* a *salt* – zakódované heslo a náhodně vygenerovaný řetězec, který je za dané heslo připojen pro zvýšení bezpečnosti. Každý uživatel musí mít také přiřazenou přesně jednu roli – *userRole*, která je cizím klíčem a musí nabývat hodnot ze sloupce *id* tabulky *UserRoles*.

Tabulka *UserRoles* slouží pro uložení jednotlivých rolí. Jméno každé role musí být unikátní a nenulové, a proto je atribut *name* kandidátním klíčem této tabulky.

Zřejmě nejdůležitější tabulkou je relace *Products*, jejíž jednotlivé záznamy reprezentují jednotlivé položky na skladě. Z jejích atributů si zaslouží vysvětlení položka *quantity*, která udává počet exemplářů daného výrobku na skladě, tedy například počet minerálek, zmrzlin apod. Položka *unitQuantity* představuje množství jednoho konkrétního exempláře, tedy například kolik litrů má jedna minerálka, nebo například kolik kilogramů váží jedno balení hranolek. Atribut *price* udává buď cenu za jeden exemplář výrobku nebo cenu za jeden kilogram, gram, litr apod. jednoho výrobku v závislosti na booleanovském atributu *priceForUnit*. Pokud tento atribut nabývá hodnoty true, cena je násobena danou váhou, tzn. atributem *unitQuantity*, nikoliv množstvím (tzn. nikoliv *quantity*, které je chápáno v této situaci jako rovné hodnotě jedna, což je omezení kladené programem na výrobek účtovaný dle váhy). Pro shrnutí, atribut *priceForUnit* udává způsob, jakým se bude vypočítávat cena – buď dle počtu kusů, nebo podle toho, kolik váží jedna objednávka mrkví, jablek apod.

Položka *unit* je cizím klíčem a hodnoty tohoto atributu jsou závislé na hodnotách položky *id* v tabulce *Units*, která je číselníkem jednotlivých jednotek, ve kterých můžeme výrobky vážit (kg, g, l, ml …). Dalším cizím klíčem je atribut *vatId*, který je vázán na atribut *id* v tabulce *Vat* (anglicky VAT = value added tax, česky DPH = daň z přidané hodnoty). Tabulka *Vat* je číselníkem jednotlivých daňových sazeb a má atribut *rate*, jehož hodnoty jsou unikátní a přestavují výši jednotlivých daňových sazeb (0%, 10%, 15%, 21%).

Tabulka *ImagesTable* slouží pro uložení jednotlivých obrázků ke skladovým položkám. Atribut *name* slouží jako neunikátní název obrázku, v této aplikaci je to název souboru, ze kterého byl obrázek načten. Položka *img* je samotný obrázek uložený v binární podobě.

Tabulka *ProductImages* slouží jako vztahová (propojovací) relace, která jednomu produktu přiřazuje jeden nebo více obrázků. Položky *idProdukt* a *idImage* jsou cizí klíče a jsou závislé na primárních klíčích tabulek *Products* a *ImagesTable*. V tabulce *ProductImages* tvoří atributy *idProdukt* a *idImage* kandidátní složený klíč a sloupec *id* může působit redundantním dojmem. Tato domněnka by byla pravdivá, kdyby aplikace nevyužívala sloupec *id* jako informaci o tom, jaký obrázek byl přidán jako poslední. Proto je existence tohoto sloupce odůvodnitelná.

Tabulky *Orders, OrderItems* a *OrderItemsVat* slouží k uložení již vydaných účtenek do databáze. Tabulka *Orders* ukládá kromě informací týkajících se vydané účtenky také data vztahující se ke společnosti, což může působit jako ukládání redundantních dat, která jsou dostupná již v tabulce *Company*. Je potřeba ale brát v úvahu skutečnost, že zpětné zobrazení účtenky vyžaduje data, která byla dostupná v době provedení nákupu. Jelikož program umožňuje měnit název společnosti a informace s ní související, nemusela by být potřebná historická data dostupná a tudíž je potřeba je ukládat zvláště. To samé se vztahuje na sazby DPH. Tabulka *Orders* také ukládá redundantně cenu celého nákupu, která by šla dopočítat dynamicky, ale kvůli nevhodnému návrhu funkcí pracujících s daty a příslibu výrazného zjednodušení aplikace jsem se rozhodl tuto tabulku nenormalizovat.

Výpis 4.1 představuje příkazy jazyka SQL pro vytvoření databázové struktury. Přidání cizích klíčů bylo pro přehlednost vyčleněno do samostatných příkazů.

Výpis 4.1 - SQL pro vytvoření tabulek a integritních omezení

```
CREATE TABLE Company (id INT PRIMARY KEY IDENTITY(1,1),
                      name NCHAR(100) not null UNIQUE,
                      adress NCHAR(100),
                      city NCHAR(100),
                      postalCode INT,
                      phone NCHAR(20),
                      web NCHAR(100))

CREATE TABLE Bill (id INT PRIMARY KEY,
                   billText NVARCHAR(200))

ALTER TABLE Bill ADD CONSTRAINT fk_bill FOREIGN KEY (id) REFERENCES Company(id)

CREATE TABLE Users (id INT PRIMARY KEY IDENTITY(1,1),
                    username NCHAR(40) not null UNIQUE,
                    pswd NVARCHAR(MAX) not null,
                    salt NVARCHAR(MAX) not null,
                    userRole INT not null)

CREATE TABLE UserRoles (id INT PRIMARY KEY IDENTITY(1,1),
                        name NCHAR(40) not null UNIQUE)

ALTER TABLE Users ADD CONSTRAINT fk_role FOREIGN KEY (userRole) REFERENCES
UserRoles(id)



CREATE TABLE Products (id INT PRIMARY KEY IDENTITY(1,1),
                       name NCHAR(100) not null,
                       quantity INT not null,
                       unit INT,unitQuantity DECIMAL(18,2),
                       expirationDate DATE,
                       code INT not null UNIQUE,
                       price DECIMAL(18,2),
                       priceForUnit BIT not null,vatId NCHAR(1) not null)

CREATE TABLE Units (id INT PRIMARY KEY IDENTITY(1,1),
                    name NCHAR(40) not null UNIQUE)

CREATE TABLE Vat (id NCHAR(1) PRIMARY KEY,
                  rate INT UNIQUE not null)

ALTER TABLE Products ADD CONSTRAINT fk_units FOREIGN KEY (unit) REFERENCES Units(id)

ALTER TABLE Products ADD CONSTRAINT fk_vat FOREIGN KEY (vatId) REFERENCES Vat(id)

CREATE TABLE Orders   (id INT PRIMARY KEY IDENTITY(1,1),
                       paid INT,
                       staff NCHAR(100),
                       timeOfTransaction DATETIME,
                       companyName NCHAR(100),
                       companyAdress NCHAR(100),
                       companyCity NCHAR(100),
                       companyPostalCode NCHAR(100),
                       companyPhone NCHAR(100),
                       companyWeb NCHAR(100),
```

```
                        billText NCHAR(100),
                        vatSum DECIMAL(18,2),
                        vatSumSingle DECIMAL(18,2),
                        totalShoppingCartPrice DECIMAL(18,2),
                        change DECIMAL(18,2) )

CREATE TABLE OrderItems(id INT PRIMARY KEY IDENTITY(1,1),
                         orderId INT not null,
                         name NCHAR(100) not null,
                         quantity INT not null,
                         unit NCHAR(40),
                         unitQuantity DECIMAL(18,2),
                         expirationDate DATE,
                         code INT not null,
                         price DECIMAL(18,2),
                         priceForUnit BIT not null,
                         vatId NCHAR(1) not null,
                         totalPrice DECIMAL(18,2))

ALTER TABLE OrderItems ADD CONSTRAINT fk_orderId FOREIGN KEY (orderId) REFERENCES
Orders(id) ON DELETE CASCADE

CREATE TABLE OrderItemsVat (orderId INT,
                            vatId NCHAR(1),
                            percentageLabel NCHAR(100),
                            vatValue DECIMAL(18,2),
                            vatValueProducts DECIMAL(18,2),
                            primary key (orderId, vatId))

ALTER TABLE OrderItemsVat ADD CONSTRAINT fk_compositeKey1 FOREIGN KEY (orderId)
REFERENCES Orders(id) ON DELETE CASCADE

CREATE TABLE ImagesTable (id INT PRIMARY KEY IDENTITY(1,1),
                          name NCHAR(100) not null,
                          img IMAGE not null)

CREATE TABLE ProductImages (id INT PRIMARY KEY IDENTITY(1,1),
                            idProduct INT,
                            idImage INT)

ALTER TABLE ProductImages ADD CONSTRAINT fk_idProduct FOREIGN KEY (idProduct)
REFERENCES Products(id) ON DELETE CASCADE

ALTER TABLE ProductImages ADD CONSTRAINT fk_idImage FOREIGN KEY (idImage) REFERENCES
ImagesTable(id) ON DELETE CASCADE
```

4.2.3 Objektově relační mapování

Synchronizaci mezi objekty v paměti počítače a jejich reprezentací v databázovém systému zajišťuje v tomto projektu LINQ To SQL, které mapuje jednotlivé databázové relace na objekty, jak znázorňuje schéma v příloze A.

4.2.4 Diagram tříd a struktur

Schéma na obrázku v příloze B představuje základní třídy a struktury projektu, pro jednoduchost bez jejich vzájemných vazeb. Třídy s příponou *Setup* se starají o jednotlivé hlavní moduly programu, třídy s předponou *Validate* obsahují metody pro validaci vstupních formulářových dat. Struktury slouží pro uložení dat, která jsou poté předána grafickým objektům k zobrazení.

4.2.5 Implementace účtenky

Na základě obsahu nákupního košíku zákazníka je vytvořena účtenka obsahující informace o jednotlivých položkách, ceně, DPH, obsluze, času nákupu a o podniku.

Účtenka je v režimu náhledu uložena jako dočasný XML soubor na disku počítače, po dokončení nákupu dojde k překlopení těchto dat do databáze. Účtenka je poté nastylována a zobrazena jako HTML stránka pomocí XSLT transformace. Účtenku je možné zobrazit přímo pomocí programu PASS nebo pomocí prohlížeče Internet Explorer. Pro upozornění dodávám, že v době psaní této práce většina konkurenčních prohlížečů neumožňuje z bezpečnostních důvodů zobrazovat transformované XML soubory na lokálním počítači v nastylované podobě. Samotný XML soubor jednoho konkrétního nákupu o dvou položkách vypadá například jako na výpisu 4.2.

Výpis 4.2 - XML soubor účtenky

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="../CashRegister/Bill.xslt"?>
<bill>

  <product>
    <name>Předměřická mouka pšeničná</name>
    <quantity>5</quantity>
    <unit>kg</unit>
    <unitQuantity>1,00</unitQuantity>
    <expirationDate>28.08.2018</expirationDate>
    <code>12</code>
    <totalPrice>49,50</totalPrice>
    <priceForUnit>False</priceForUnit>
    <priceForSingleUnit>9,90</priceForSingleUnit>
    <vatType>B</vatType>
  </product>

  <product>
    <name>Jablka červená</name>
    <quantity>2</quantity>
```

```
    <unit>kg</unit>
    <expirationDate>27.09.2016</expirationDate>
    <code>2</code>
    <totalPrice>35,80</totalPrice>
    <priceForUnit>True</priceForUnit>
    <priceForSingleUnit>17,90</priceForSingleUnit>
    <vatType>B</vatType>
  </product>

  <totalShoppingCartPrice>85,30</totalShoppingCartPrice>
  <paid>100,00</paid>
  <change>15</change>
  <staff>manažer</staff>
  <time>28.08.2016 21:41</time>
  <vatSum>12,80</vatSum>
  <vatSumSingle>85,30</vatSumSingle>
  <D percentage="0 %" totalPrice="0,00">0,00</D>
  <C percentage="10 %" totalPrice="0,00">0,00</C>
  <B percentage="15 %" totalPrice="85,30">12,80</B>
  <A percentage="21 %" totalPrice="0,00">0,00</A>

  <companyName>Večerka Na Rohu</companyName>
  <companyAdress>Hvězdova 1500</companyAdress>
  <companyCity>Pankrác</companyCity>
  <companyPostalCode>14000</companyPostalCode>
  <companyPhone>+420 732 251 420</companyPhone>
  <companyWeb>www.vecerkanarohu.cz</companyWeb>
  <billText>Děkujeme Vám za nákup. Veškeré dotazy zasílejte na adresu
            info@vecerkanarohu.cz.
  </billText>

</bill>
```

Samotná XSLT transformace je prováděna prostřednictvím souboru *Bill.xslt*, jehož obsah prezentuje výpis 4.3.

Výpis 4.3 - XSLT soubor pro transformaci účtenky

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <xsl:text disable-output-escaping='yes'>&lt;!DOCTYPE html&gt;</xsl:text>
    <html>
      <body>
        <head>
          <title>Účtenka</title>
        </head>

        <style>
          .
          .
          . ZKRÁCENO
          .
          .
        </style>

        <div id="container" style="">
          <header>
```

```
          <table id="headerTable">
            <tr>
              <td>
                <b>
                  <xsl:value-of select="bill/companyName"/>
                </b>
              </td>
            </tr>
            <xsl:if test="bill/companyAdress">
              <tr>
                <td>
                  <xsl:value-of select="bill/companyAdress"/>
                </td>
              </tr>
            </xsl:if>
            <xsl:if test="bill/companyCity or bill/companyPostalCode">
              <tr>
                <td>
                  <xsl:value-of select="bill/companyCity"/>
                  <xsl:if test="bill/companyPostalCode and bill/companyCity">,
                  </xsl:if>
                  <xsl:value-of select="bill/companyPostalCode"/>
                </td>
              </tr>
            </xsl:if>
            <xsl:if test="bill/companyPhone">
              <tr>
                <td>
                  Telefon: <xsl:value-of select="bill/companyPhone"/>
                </td>
              </tr>
            </xsl:if>
            <xsl:if test="bill/companyWeb">
              <tr >
                <td>
                  Internet: <xsl:value-of select="bill/companyWeb"/>
                </td>
              </tr>
            </xsl:if>
          </table>
        </header>
           .
           .
           . ZKRÁCENO
           .
           .
        <!-- Patička -->
        <table id="footerTable"  style="margin-top:15px;border-top:1px
        dashed black">
          <tr >
            <td style="padding-top:5px">
              <xsl:value-of select="bill/time"/>
            </td>
          </tr>
          <xsl:if test="bill/billText">
            <tr>
              <td>
                <xsl:value-of select="bill/billText"/>
              </td>
            </tr>
```

```
            </xsl:if>
          </table>
        </div>
      </body>
    </html>
  </xsl:template>
  </xsl:stylesheet>
```

V předcházejícím výpisu byl obsah z režijních důvodů značně zkrácen, v plné podobě je k dispozici v rámci projektu na přiloženém optickém nosiči.

4.3 Systémové požadavky, instalace, odinstalace

Aplikace využívá .NET Framework 4.5.2 a tudíž minimální systémové požadavky[5] jsou:

Podporovaný operační systém:
Windows 7 Service Pack 1; Windows 8; Windows 8.1; Windows 10; Windows Server 2008 R2 SP1; Windows Server 2008 Service Pack 2; Windows Server 2012; Windows Server 2012 R2; Windows Vista Service Pack 2.

Požadavky na hardware:
Procesor s frekvencí 1 GHz nebo vyšší
512 MB paměti RAM, 4,5 GB volného místa na pevném disku (x86, x64)
Instalace programu se spustí prostřednictvím instalačního souboru *setup.exe*, který vyžaduje administrátorská práva. Po provedení standardní instalace je na plochu a do nabídky start přidán zástupce pro spuštění programu.

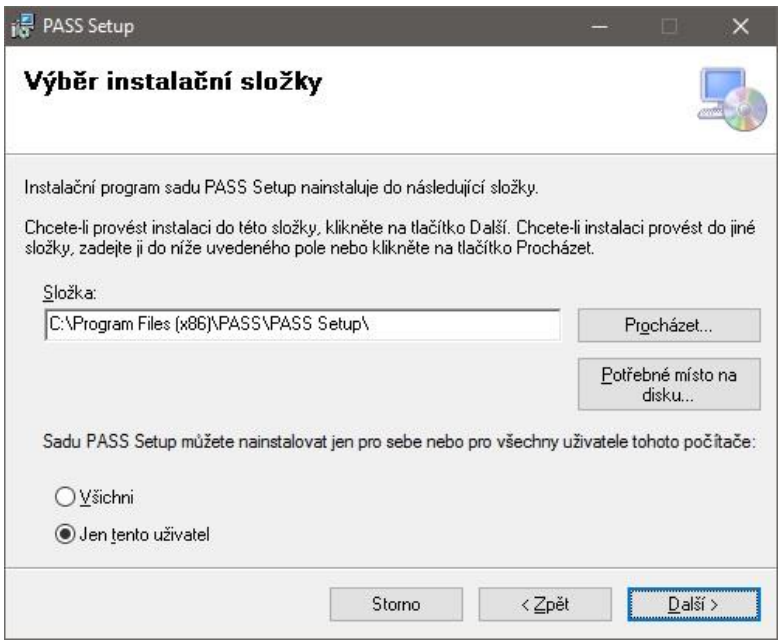
Instalační soubor aplikace *setup.exe* se v případě absence pokusí nainstalovat přiloženou požadovanou verzi .NET Frameworku automaticky (není vyžadováno připojení k internetu), stejně tak jako SQL Server 2014 Express LocalDB. Dále je potřeba místo na pevném disku pro samotnou aplikaci (cca. 14 MB). Podporované jsou jak 32 tak 64 bitové verze Windows a minimální vhodné rozlišení je 800x600 pixelů.

Odinstalace programu se provádí standardně pomocí ovládacích panelů. Po odinstalování programu mohou na lokálním počítači zůstat konfigurační nastavení, místní databáze a

[5] Více informací o .NET Frameworku: https://www.microsoft.com/cs-cz/download/details.aspx?id=42642

uložené nesmazané účtenky. Při opětovné instalaci programu je uživatel upozorněn, že tato data budou ztracena. Po odinstalaci programu je možné reziduální soubory odstranit ručně smazáním složky PASS v adresářích:
C:\Users\user_name\AppData\Local a *C:\Users\user_name\AppData\Roaming*,
kde *user_name* je jméno vašeho Windows uživatelského účtu.

Obrázek 4.3 - Instalace aplikace



4.4 Uživatelská práce s programem a základní nastavení

Nejjednodušší konfigurace programu PASS je instalace produktu na jeden počítač a využívání místní databáze (technologie LocalDB). Toto nastavení je výchozí a nevyžaduje nutnost instalace plnohodnotného Microsoft SQL Serveru, což zpřístupňuje použití tohoto programu i laickému uživateli. Mezi přínosy této konfigurace patří absence jakéhokoliv dodatečného nastavování aplikace a absence instalace dalšího software. Dalším přínosem je jednodušší zálohování databáze a vytváření databáze nové (poslední možnost poskytuje přímo program PASS).

Druhým scénářem použití je poté upravení nastavení programu prostřednictvím účtu administrátora, který je vytvořen při prvním spuštění programu. V nastavení aplikace je poté možné zvolit plnohodnotný Microsoft SQL Server, který je ale potřeba nainstalovat a

nakonfigurovat samostatně. Aplikace vyžaduje pouze zadání připojovacího řetězce k tomuto serveru. Mezi přínosy tohoto způsobu patří možnost využívat všechny možnosti neomezeného SQL Serveru a také možnost nainstalovat produkt na více počítačů současně a přistupovat z nich ke společné databázi. Toto nastavení je vhodné pro provozovny s více než jednou pokladnou.

4.4.1 První spuštění programu

Po dokončení instalace je možné program spustit přes zástupce na ploše, nebo odkaz v nabídce start. Jako první je nutné zvolit heslo správce programu, což je role nezávislá na připojené databázi a je uložena pro každý počítač zvlášť. Po zvolení možnosti „Potvrdit nastavení a spustit aplikaci“ (obr. 4.4), je provedeno základní nastavení a je vytvořena místní databáze s ukázkovými daty. Tato operace může chvíli trvat.

Obrázek 4.6 ukazuje okno přihlášení uživatele. PASS podporuje kromě administrátorského účtu tři základní uživatelské role: manažer, skladník a pokladní. Význam těchto rolí je samopopisný. Každý uživatel systému musí mít přiřazenou pouze jednu roli. Výchozí uživatelská jména a hesla popisuje sekce „Důležité informace“ na obrázku 4.2. Kromě přihlášení, okno na obrázku 4.6 obsahuje také vstup do administrátorské sekce programu, která je rovněž chráněna heslem. Administrátorská sekce (obr. 4.7) umožňuje volbu mezi plnohodnotnou databází, nebo databází LocalDB. Dále poskytuje možnost vytvořit nový místní databázový soubor, nebo například resetovat celou aplikaci.

4.4.2 Hlavní okno programu

Po přihlášení je otevřeno hlavní okno programu (obr. 4.5), které má v záhlaví menu a pod ním jednotlivé záložky – sklad, pokladna, management a profil. V případě rolí skladník a pokladní budou přístupné pouze záložky příslušné dané roli. V pravé části okna se nachází uzavíratelné a rozšiřitelné postranní menu. Ve spodní částí programu se nachází lišta s informacemi o aktuálně přihlášeném uživateli.

4.4.3 Sekce sklad

Sekce sklad představuje hlavní modul celé aplikace a je na něm přímo závislá sekce pokladna. Mezi dostupné akce skladu patří přidání nového produktu, úprava produktu existujícího, odstranění zvoleného produktu nebo všech produktů naráz a správce jednotek.

Volby „Upravit produkt“ a „Nový produkt“ vyvolají okno shodné s oknem na obrázku 4.8 v jeho vyplněné a nevyplněné alternativě. Obrázek ukazuje, že je možné přidávat jak textové, tak i obrazové popisy produktu. Konkrétně jednotlivými položkami produktu se zabývá kapitola 4.2.2.

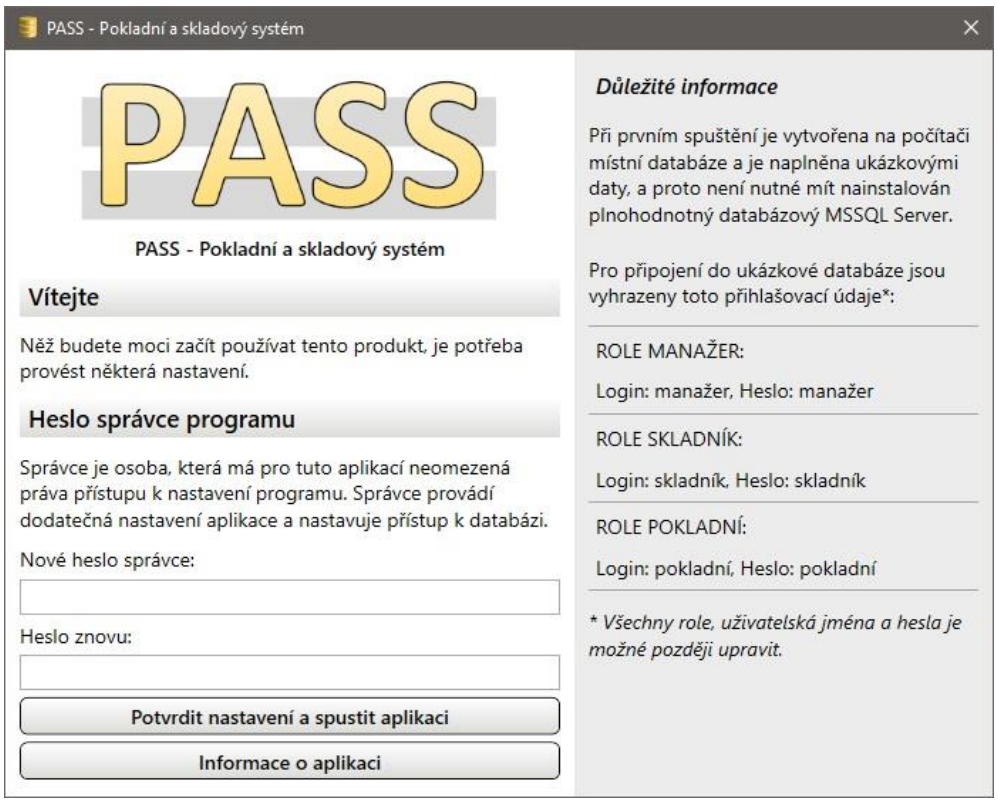
Podstatnou částí skladu je i správce jednotek, který umožňuje přidávat nové jednotky, jak ukazuje obrázek 4.9.

4.4.4 Sekce profil

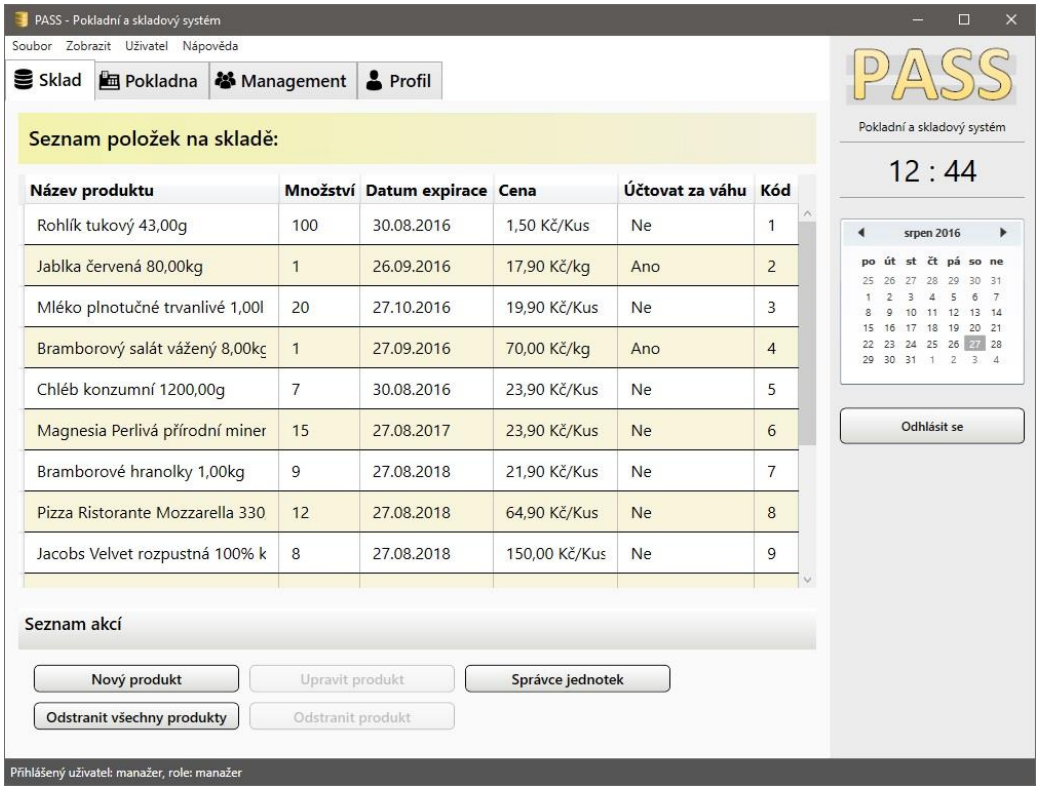
Profil nabízí možnosti týkající se aktuálně přihlášeného uživatele a obsahuje tyto podsekce:

- PŘEHLED – Podává informace o aktivním uživateli a jeho roli.
- NASTAVENÍ ÚČTU – Zde si přihlášený uživatel může změnit své heslo.
- DALŠÍ MOŽNOSTI – Odhlášení aktuálního uživatele.

Obrázek 4.4 – První spuštění programu

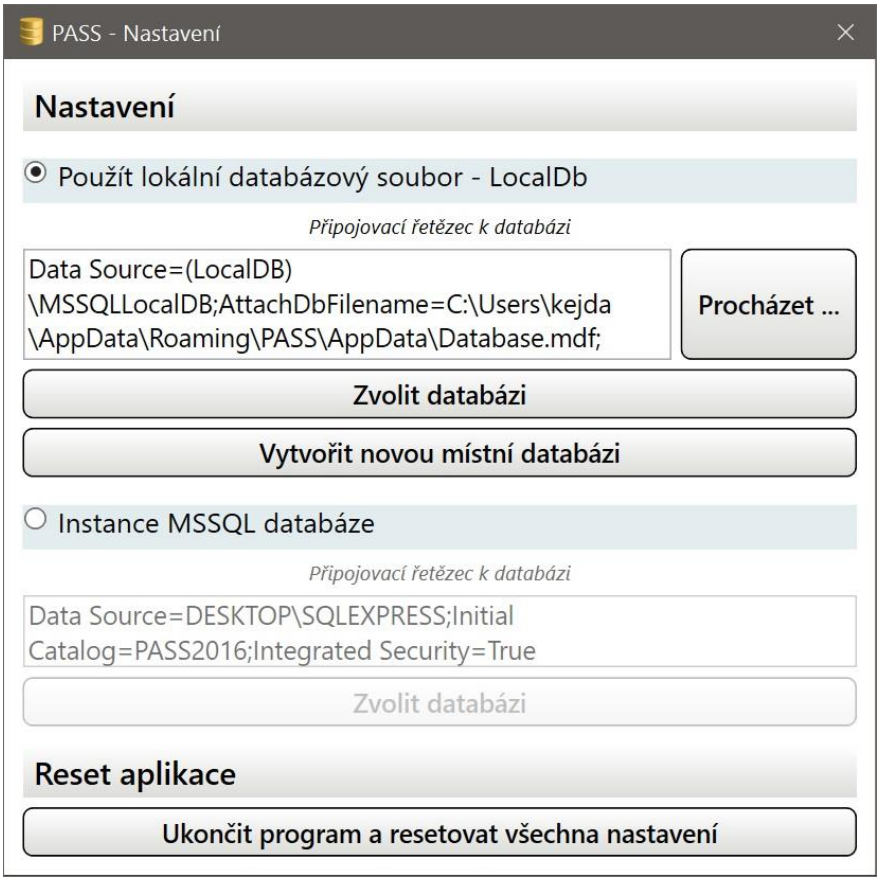


Obrázek 4.5 - Hlavní okno programu

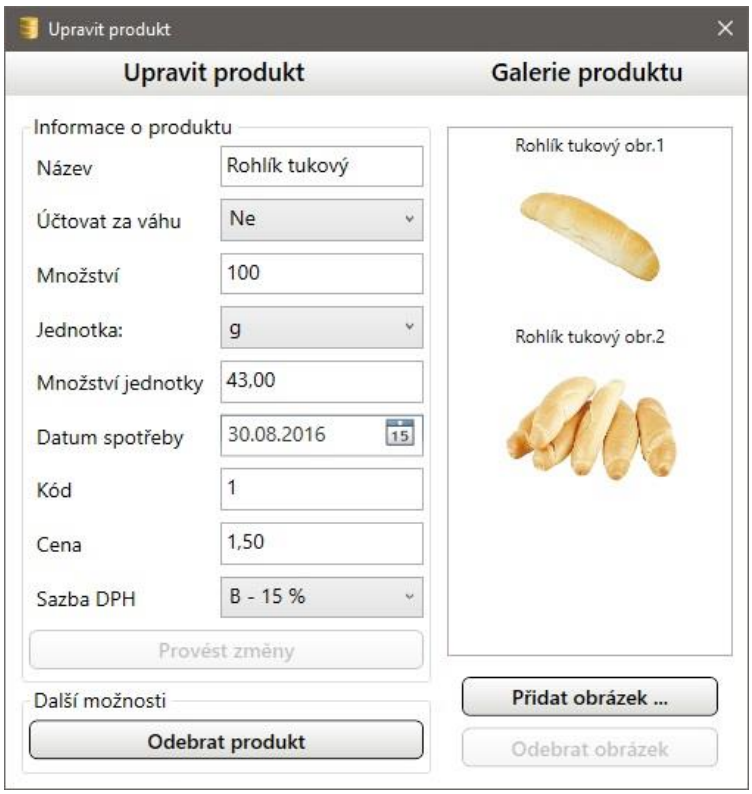


Obrázek 4.6 - Přihlašovací okno

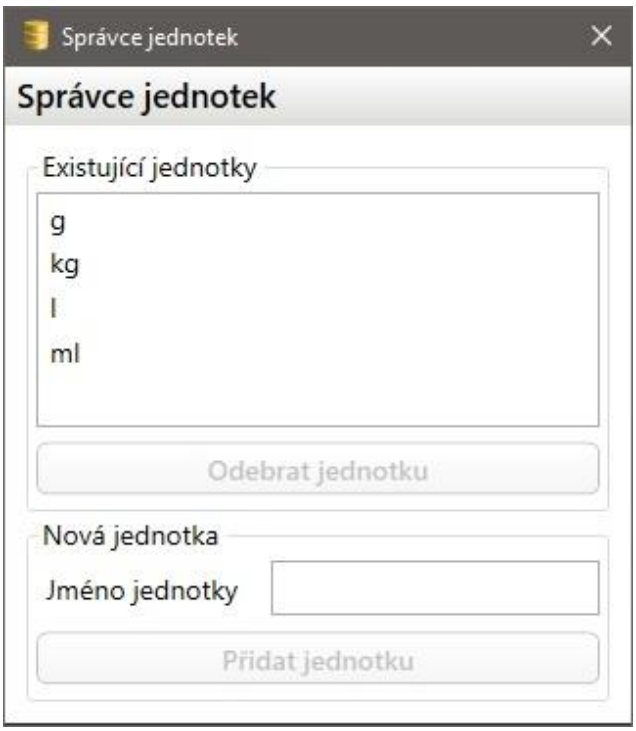
Obrázek 4.7 - Nastavení programu



Obrázek 4.8 - Úprava produktu



Obrázek 4.9 - Správce jednotek

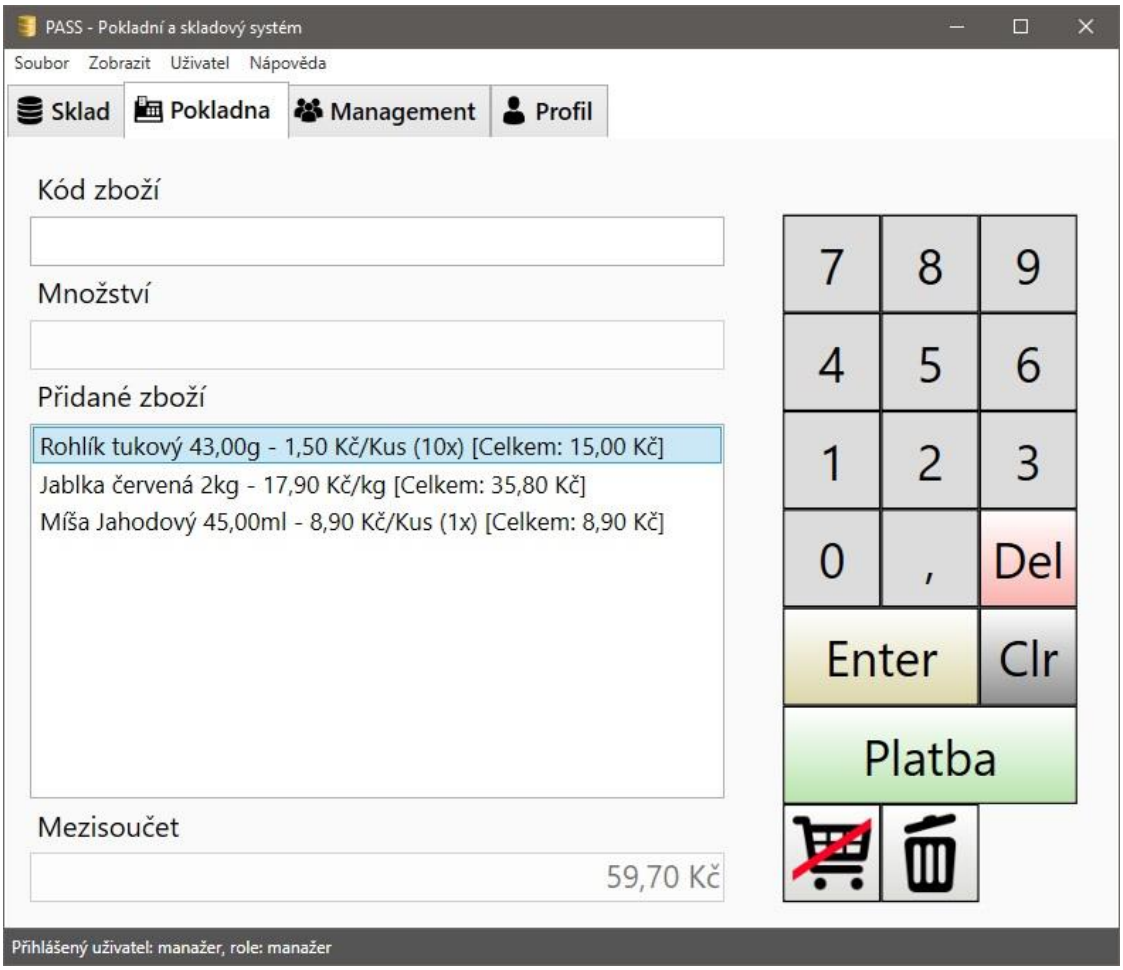


4.4.5 Sekce pokladna

Pokladna (obr. 4.10) poskytuje prostředí pro prodej zboží, které je evidováno na skladě. Grafické rozhraní obsahuje dvě podstatné části. První částí je oblast pro zadávání kódu výrobku a volba jeho množství, společně s přehledem všech přidaných položek.

Druhou částí je postranní ovládací panel skládající se z číselníku a tlačítek pro pohodlnou obsluhu pokladny na dotykových zařízeních. Specifikem tohoto panelu je automatická změna jeho velikosti dle potřeb zobrazovacího zařízení. Důsledkem toho je zvýšená čitelnost ovládacího panelu jak na malých, tak i velkých obrazovkách. Tlačítko „Del“ slouží ke smazání kódu zboží nebo množství zboží. Tlačítko „Clr“ slouží k resetování aktuálně zvoleného zboží a jeho množství. Volba „Enter“ umožní přidat další položku. Možnost „Platba“ umožní zadat přijatou částku a dokončit nákup.

Obrázek 4.10 - Pokladna



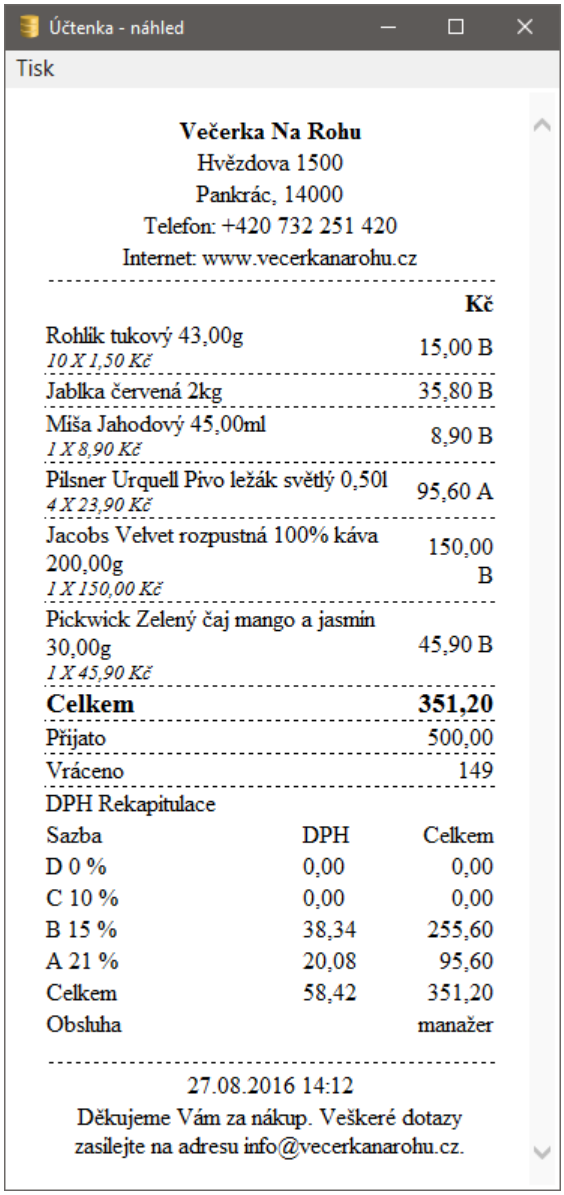
Po zvolení možnosti „Platba“ je potřeba zadat přijatou částku, a nakonec je možné zobrazit účtenku a vytisknout jí, jak ukazuje obrázek 4.11. Účtenka obsahuje záhlaví, ve kterém se nacházejí informace o podniku, které se dají upravit v sekci management, přičemž jedinou povinnou položkou je název provozovny.

Po záhlaví následuje seznam položek, který je reprezentován názvem výrobku, množstvím jednoho kusu výrobku v příslušných jednotkách a také počet kusů. Na konci řádku je suma za příslušný řádek spolu s označením sazby DPH.

Následující řádek „Celkem“ udává nezaokrouhlenou výslednou cenu. O řádek níže se nachází částka přijatá při platbě a na konci je vypsána vrácená částka.

Další sekcí je DPH rekapitulace, která uvažuje 0 %, 10 %, 15 % a 21 % sazbu.

Obrázek 4.11 - Účtenka



Na konci účtenky se nachází ještě uživatelské jméno aktuálně přihlášeného uživatele, informace o času nákupu a volitelné sdělení zákazníkům, které lze také upravit.

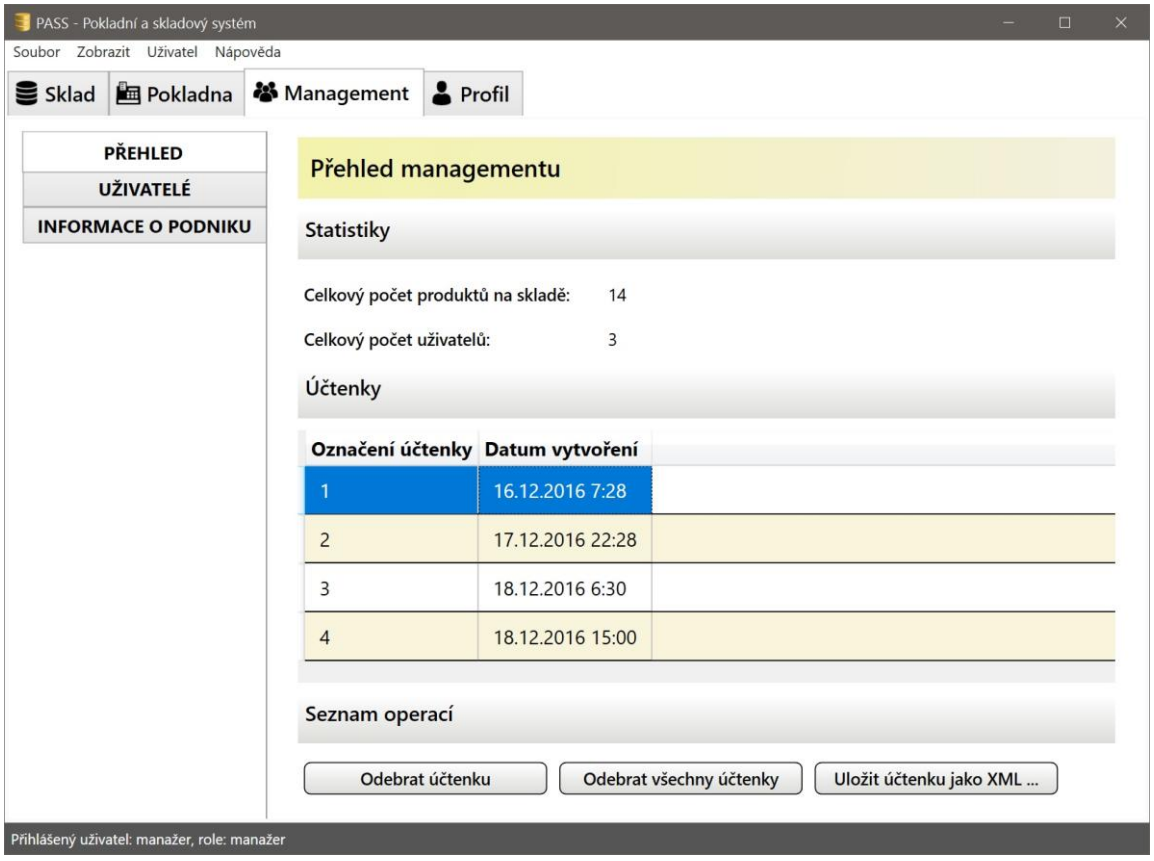
Výsledná účtenka je po dokončení nákupu uložena do databáze a je jí možné zpětně zobrazit, popřípadě odstranit v sekci management. Výslednou účtenku je možné také vytisknout pomocí volby „tisk“ v menu náhledu účtenky. Samotnou realizací účtenky se zabývá podrobněji sekce 4.2.5. Navíc, jak ukazuje obrázek 4.12, je možné uložit jakoukoliv vydanou účtenku ve formátu XML na disk počítače.

4.4.6 Sekce management

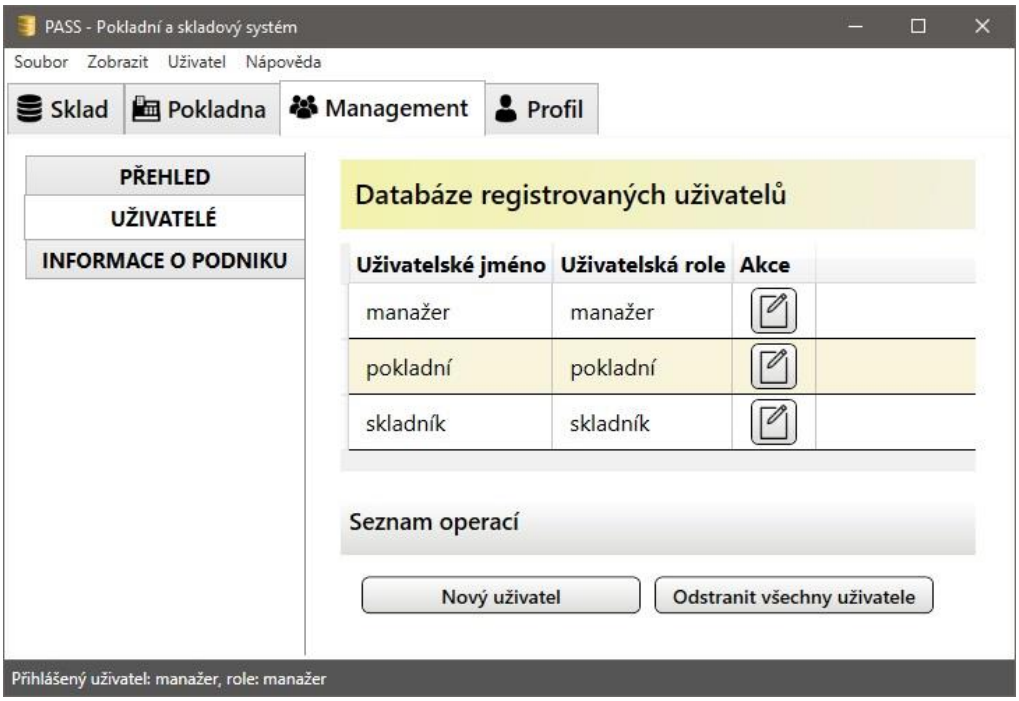
Tato sekce je přístupná pouze uživatelům s rolí manažer a sestává ze tří záložek:

- PŘEHLED – Poskytuje základní statistiky a umožňuje spravovat účtenky (obr. 4.12).
- UŽIVATELÉ – Poskytuje nástroje pro správu uživatelů (obr. 4.13).
- INFORMACE O PODNIKU – Tato záložka slouží pro nastavení informací, které se zobrazují na účtence (obr 4.14).

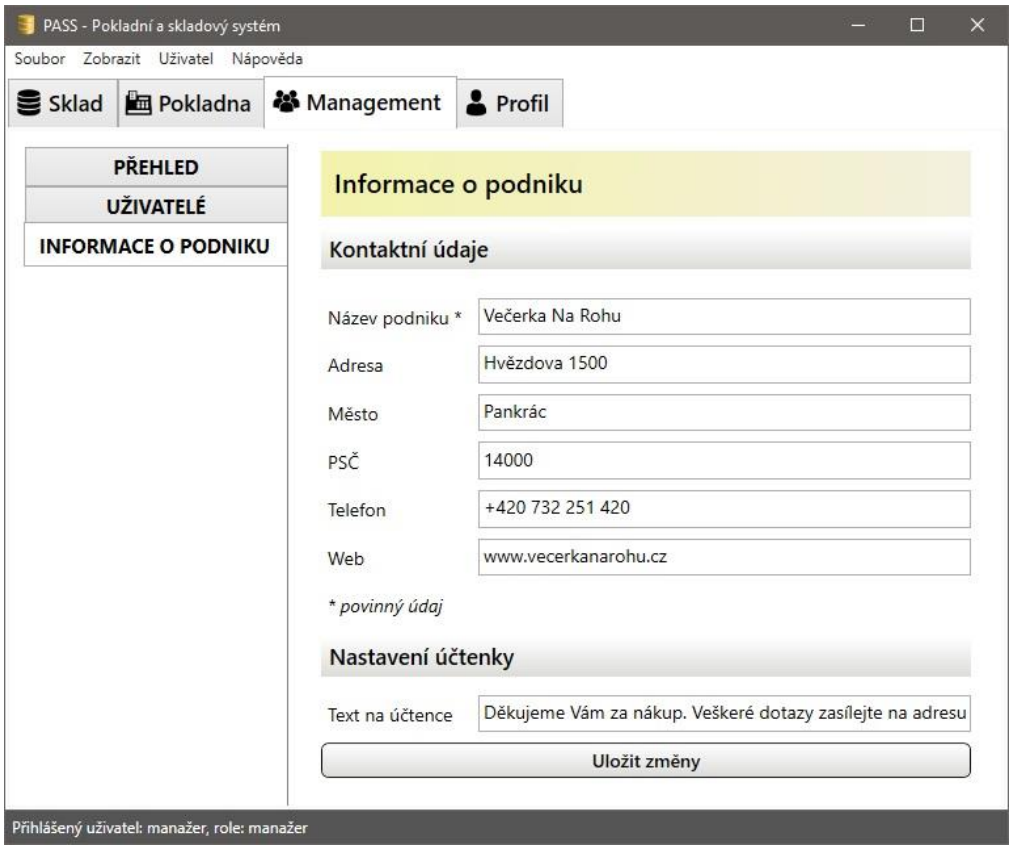
Obrázek 4.12 - Sekce management, přehled



Obrázek 4.13 - Sekce management, uživatelé



Obrázek 4.14 - Sekce management, informace o podniku



5 Výsledky a diskuse

Instalace Pokladního a skladového systému (PASS) spolu s lokální databází zabírá asi 200 MB (bez .NET Frameworku) a dá se proto bez problémů nainstalovat nejen na stolní počítače, ale i na menší přenosná zařízení s omezenou pamětí, v tomto případě nejspíše tablety s operačním systémem Windows. Využití tohoto programu na zařízeních s menšími obrazovkami nahrává i částečná responzibilita aplikace (například responzivní číselník pokladny, možnost schovat menu a responzivní chování ovládacích panelů a některých nabídek). Vše zmíněné nadále podporuje i fakt, že použitá technologie WPF používá tzv. device independent pixels, což je jednotka, která složí pro abstrakci pixelů. Ve výsledku se velikost ovládacích prvků aplikace upravuje poměrně dle nastavení DPI na daném systému a prvky GUI nejsou na obrazovkách s velkou hustotou pixelů příliš malé a naopak.

Co se týče použité technologie WPF obecně, důvodem volby oproti technologii Windows Forms bylo například to, že WPF lépe podporuje databinding ke kolekcím dat, návrh uživatelského rozhraní je oddělen od kódu aplikace a je realizován prostřednictvím jazyka XAML, technologie WPF je vektorově orientovaná a při změně přiblížení nedochází ke ztrátě kvality obrazu. WPF nabízí nejenom okna (objekt *Window*), ale také *Pages*, které přispívají ke snadnější organizaci GUI a v tomto projektu jsem je proto často využíval. Bylo by také chybou opomenout, že WPF používá hardwarovou akceleraci spolu s technologií DirectX a proto je často výsledná uživatelská zkušenost s programem plynulejší, než u starší technologie WinForms, jejíž vykreslování grafických prvků využívá v době psaní této práce již poměrně zastaralou technologii GDI/GDI+.

Jako výchozí možnost uložení dat je v programu nastaveno použití lokální databáze (LocalDB 2014), což je dle MSDN prostředek určený spíše pro vývojáře než pro nasazení do reálného provozu, protože přináší řadu omezení, například nemožnost současného přístupu více uživatelů do jedné databáze. Za předpokladu, že se bude tento produkt předvádět zákazníkovi, nebo bude pracovat pouze v režimu pouze jedné instance programu, dají se tyto nevýhody zanedbat. Pro reálné použití aplikace v praxi by ve všech ostatních scénářích měla dominovat volba plnohodnotné databáze, kterou program PASS podporuje také. Lokální databáze by mohla popřípadě sloužit jako databáze záložní pro případ ztráty spojení se serverem a po navázání spojení by došlo k synchronizaci dat.

Jako prostředek pro objektově relační mapování byla použita technologie LINQ To SQL. Tuto technologii jsem zvolil kvůli její jednoduchosti a možnost rychlého vývoje aplikace bez nutnosti psát většinu příkazů v jazyce SQL, bohužel ale za cenu ztráty výkonu, který je způsoben skutečností, že příkazy jazyka SQL jsou generovány automaticky a mohou být hypoteticky i značně neefektivní. V rámci tohoto projektu jsem tedy upřednostnil komfort a rychlost vývoje před optimalizací aplikace.

Aplikace byla navržena pro potřeby této práce s následujícími zaměřeními:

- Aplikace slouží k demonstraci perzistentního uložení dat prostřednictvím databáze, konkrétně dochází k ukládání jednotlivých položek skladu, uživatelských účtů a informací o společnosti používající aplikaci.
- Aplikace slouží k demonstraci perzistentního uložení dat prostřednictvím souborů, konkrétně k uložení jednotlivých účtenek na lokální počítač, uložení zašifrovaného hesla administrátora programu a tvorba nových lokálních databázových souborů.
- Aplikace slouží k demonstraci práce s různými typy souborů, zde se jedná například o obrázky (binární data), texty (např. výpis chyb do souboru errorLog.txt) a databázové soubory (.mdf a .ldf).
- Aplikace pracuje s více zdroji dat (databáze, soubory, data v paměti získaná od uživatele za běhu programu). Tyto data musí odpovídajícím způsobem načíst, zpracovat (realizováno pomocí LINQ) a popřípadě konvertovat a uložit, což je reprezentováno například procesem tvorby a tisku XML účtenky v závislosti na nákupu uživatele a záznamech v databázi.

Pokladní a skladové aplikace jsou navíc v praxi velmi uplatnitelné a jsou klíčovou záležitostí fungování mnoha podniků, přičemž jejich přínos není pouze v ukládání a zpracování dat, ale také v automatizaci procesů, diagnostice problémů – typicky nedostatek zboží na skladě, statistické analýze dat a tvorbě prodejních reportů.

Hypotetická možnost využití projektu PASS v praxi v jeho aktuální podobě je spíše nepravděpodobná, vzhledem k zavedení elektronické evidence tržeb, které proběhlo v prosinci roku 2016. Zmíněná absence funkcionality pro práci s EET snižuje

použitelnost programu v praxi, ačkoliv rozšíření programu o modul pro komunikaci s EET by bylo relativně snadno realizovatelné, protože aplikace v aktuální podobě eviduje prodeje nejenom do XML, ale především do společné databáze. Komunikaci s modulem EET je navíc možné si vyzkoušet v rámci projektu EET Playground[6].

Dále je možné program provozovat pouze na operačním systému Microsoft Windows, což neumožňuje jeho používání na široce rozšířených přenosných zařízení se systémem Android, což je výhoda například komerčního systému Dotykačka[7].

Navzdory řečenému, program poskytuje solidní základ pro pokladní a skladový systém pro nasazení v soukromé sféře, zejména pro svou schopnost evidovat výrobky, generovat účtenky a ukládat je do místní nebo centrální databáze, vést systém uživatelů s odstupňovanými právy, podporou více pokladen, částečnou responzibilitou, intuitivním uživatelským rozhraním a snadnou standardní instalací.

[6] Více informací na http://www.etrzby.cz/cs/technicka-specifikace
[7] Více informací na https://www.dotykacka.cz/

6 Závěr

Hlavním cílem teoretické části práce bylo představit prostředky .NET Frameworku vztahující se k souborovému a relačně databázovému uložení a zpracování dat. Kapitola 3.1 tyto dva hlavní přístupy nejprve obecně rozlišila a kapitoly 3.2 a 3.3 poskytly nezbytný základ pro pochopení prostředí .NET Frameworku a jazyka C#. Následující kapitola 3.4 se věnovala již samotné implementaci perzistence dat prostřednictvím jak souborů, tak i databázového systému. Nakonec kapitola 3.5 představila možnosti platformy .NET a jazyka C# týkající se přístupu k datům a jejich zpracování, především prostřednictvím jazyka LINQ.

Navazujícím cílem práce bylo využít popsaných prostředků k implementaci pokladního a skladového systému, který pracuje s daty jak prostřednictvím samostatných souborů, tak i databáze. Tento cíl byl naplněn ve čtvrté kapitole této práce, jejímž výstupem je návrh a implementace Pokladního a skladového systému (PASS), který využívá většinu popisovaných prostředků a realizuje jejich prostřednictvím perzistentní ukládání dat.

Na základě faktu, že platforma .NET představuje velice rozsáhlý komplex knihoven, vývojových nástrojů, podporovaných programovacích jazyků, má volně dostupnou a srozumitelnou dokumentaci a množství prostředků pro zpracování dat a zajištění datové perzistence, které byly popsány v teoretické části a využity v části praktické, jsem došel k závěru, že tato platforma spolu s jazykem C# poskytuje více než dostatečné kvantum kvalitních a vhodných prostředků pro pro realizaci datové perzistence a zpracování dat.

7 Seznam použitých zdrojů

Tištěné zdroje

ALBAHARI, J. -- ALBAHARI, B. *C# 5.0 in a nutshell*. Sebastopol: O'Reilly, 2012. ISBN 978-1-449-32010-2.

NAGEL, Christian. *C# 2008: programujeme profesionálně*. Brno: Computer Press, 2009. ISBN 978-80-251-2401-7.

SHARP, J. *Microsoft Visual C# 2008: krok za krokem*. Brno: Computer Press, 2008. ISBN 978-80-251-2027-9.

TROELSEN, A. *Pro C# 5.0 and the .NET 4.5 framework*. Berkeley: Apress, 2012. ISBN 978-1-4302-4234-5.

VIRIUS, M. *C# 2010: hotová řešení*. Brno: Computer Press, 2010. ISBN 978-80-251-3730-7.

VOSTROVSKÝ, Václav. *Vytváření databází v ORACLE*. Praha: Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta, 2012. ISBN 80-213-1191-6.

WIENER, Norbert. *Kybernetika a společnost*. Praha: Československá akademie věd, 1963.

Elektronické zdroje

BĚHÁLEK, Marek. *CLR – Common Language Runtime* [online]. [cit. 2016-06-09]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch01s02.html

HRACH, Václav. *Podpora persistence v desktopových aplikacích na platformě .NET/C#* [online]. Jihlava, 2013 [cit. 2016-06-05]. Dostupné z: https://is.vspj.cz/bp/get-bp/student/17119/thema/3624

Introduction to the C# Language and the .NET Framework [online]. 2015 [cit. 2017-02-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

MERUNKA, Vojtěch, Robert PERGL a Marek PÍCKA. *Objektově orientovaná tvorba softwaru* [online]. Praha, 2004 [cit. 2016-06-05]. Dostupné z: https://www.researchgate.net/publication/40319704_Objektove_orientovana_tvorba_softwaru.

Microsoft Developer Network: .NET Framework Class Library [online]. [cit. 2016-06-05]. Dostupné z: https://msdn.microsoft.com/en-us/library/mt472912(v=vs.110).aspx

Programming Languages [online]. 2003 [cit. 2017-02-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa292164(v=vs.71).aspx

VESELÝ, Arnošt. *Unixové operační systémy* [online]. Praha, 2013 [cit. 2017-02-22].

Zdroje obrázků zboží a informace o produktech

1. *Akční ceny - zboží* [online]. [cit. 2016-09-08]. Dostupné z: http://www.akcniceny.cz/
2. *Tesco - Potraviny oline* [online]. [cit. 2016-09-08]. Dostupné z: https://nakup.itesco.cz

Zdroje grafických prvků použitých v projektu PASS

1. *Flaticon* [online]. [cit. 2016-09-08]. Dostupné z: http://www.flaticon.com/authors/freepik

2. *Flaticon* [online]. [cit. 2016-09-08]. Dostupné z: http://www.flaticon.com/authors/chris-veigt

3. *Flaticon* [online]. [cit. 2016-09-08]. Dostupné z: http://www.flaticon.com/authors/madebyoliver

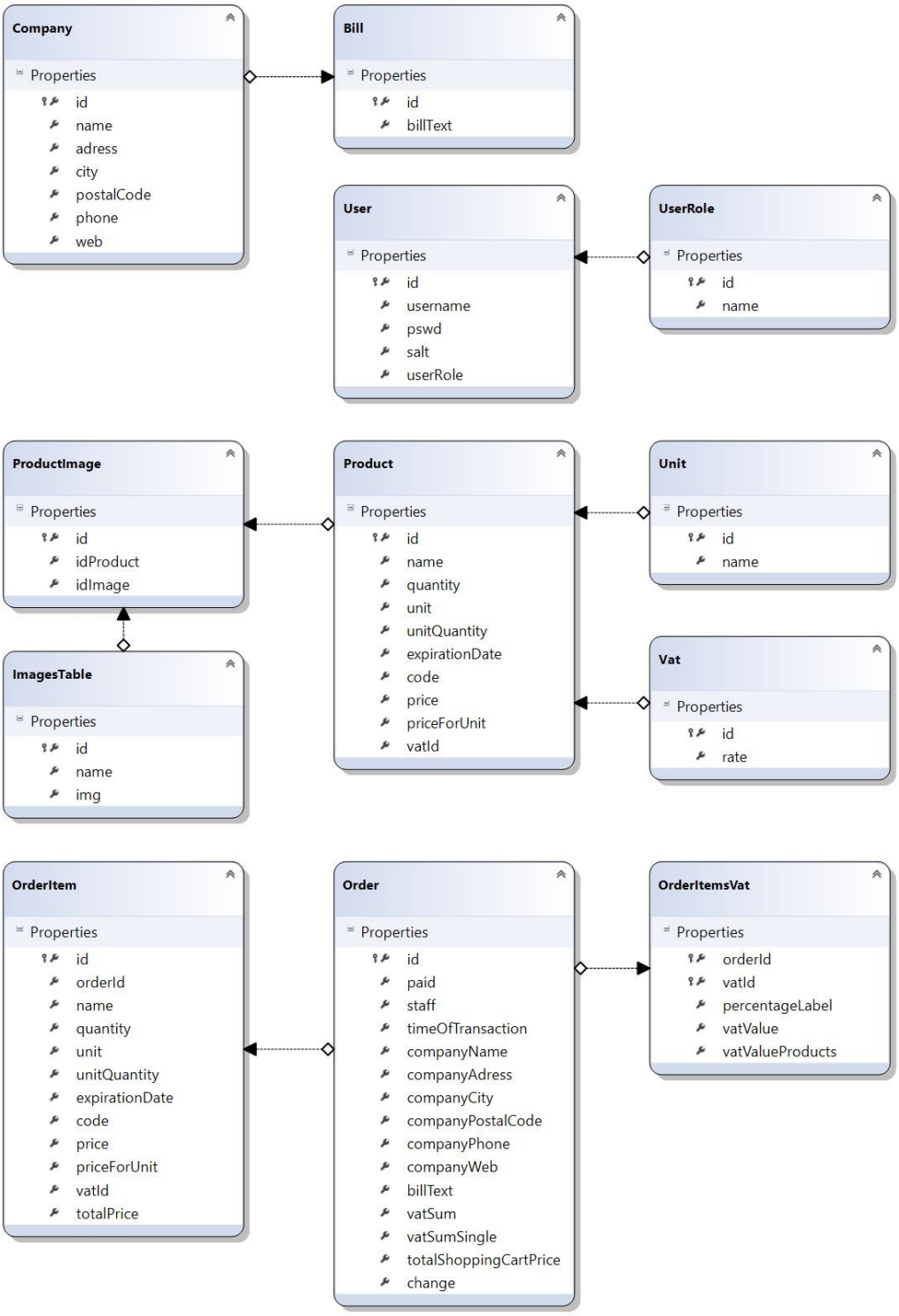
4. *Flaticon* [online]. [cit. 2016-09-08]. Dostupné z: http://www.flaticon.com/authors/stephen-hutchings

5. *Flaticon* [online]. [cit. 2016-09-08]. Dostupné z: http://www.flaticon.com/authors/sergiu-bagrin

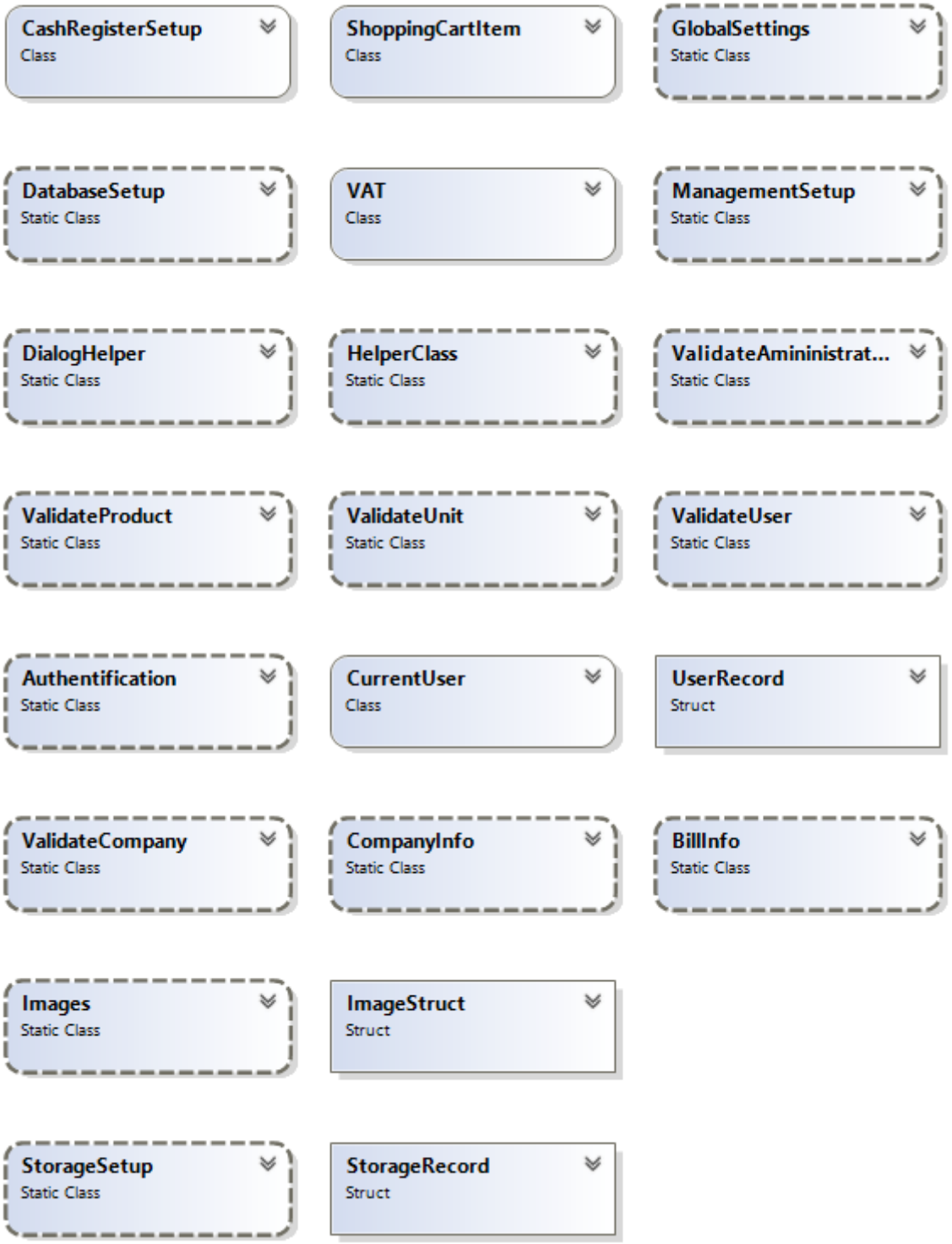
6. *IconArchive* [online]. [cit. 2016-09-08]. Dostupné z: http://www.iconarchive.com/show/ravenna-3d-icons-by-double-j-design/Database-Active-icon.html

8 Přílohy

A. Schéma objektově relačního mapování



B. Třídní diagram



C. Optický disk

Na přiloženém optickém disku se nachází zdrojový kód aplikace spustitelný přes soubor projektu prostřednictvím Microsft Visual Studia. Dále se zde nachází instalační soubor pro nasazení aplikace do provozu.