Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj softwaru skladového hospodářství s využitím Entity frameworku

Dominik Klodner

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Dominik Klodner

Informatika

Název práce

Vývoj softwaru skladového hospodářství s využitím Entity frameworku

Název anglicky

Development of warehouse management software using Entity framework

---

Cíle práce

Práce se zaměřuje na problematiku využití Entity Frameworku při vývoji desktopových aplikací využívajících lokální databázi. Cílem práce je navrhnout a implementovat aplikaci sloužící k jednoduché evidenci skladového hospodářství v malé firmě. Dílčím cílem bude popsat Entity Framework a možnosti jeho využití.

Metodika

Bakalářská práce sestává ze dvou částí – teoretické a praktické. Metodiky zpracování teoretické části spočívá ve studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Bude popsána technologie Entity Framework a možnosti jejího využití při tvorbě aplikací.

V praktické části bude provedena analýza a následná implementace desktopové aplikace skladového hospodářství. Při návrhu a implementaci bude využito standardních nástrojů softwarového inženýrství. Implementace bude provedena v jazyce C# v prostředí .NET a bude využívat Entity Framework pro objektové mapování dat z relační databáze. Aplikace bude otestována a budou shrnuty poznatky z jejího vývoje a navrženy případné další možnosti budoucího rozvoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Entity framework, C#, skladové hospodářství, desktopová aplikace, visual studio, sklad, WPF, .Net, informační systém

---

Doporučené zdroje informací

Čápka, David (2018). Největší český C# .NET portál a kompletní on-line kurzy. [online] Itnetwork.cz. Available at: https://www.itnetwork.cz/

Docs.microsoft.com. (2018). Technical documentation, API, and code examples. [online] Available at: https://docs.microsoft.com

Lerman, Julia (2012). Programming Entity Framework: Dbcontext

MacDonald, Matthew (2012). Pro WPF 4.5 in C#. Berkeley, CA: Apress

Msdn.microsoft.com. (2018). Learn to Develop with Microsoft Developer Network | MSDN. [online] Available at: https://msdn.microsoft.com/en-us/dn308572.aspx

---

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 26. 2. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 26. 2. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj softwaru skladového hospodářství s využitím Entity frameworku" 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 15.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi za odborné vedení, dohled a cenných konzultací při psaní této práce.

Vývoj softwaru skladového hospodářství s využitím Entity frameworku

Abstrakt

Tato bakalářská práce je založena na kompletním návrhu a realizace informačního systému zabývajícího se problematikou skladového hospodářství a firemních procesů, po testovací nasazení aplikace do provozu. Aplikace by měla být koncipována pro využívání v menších firmách nebo jednotlivým živnostníkům. Práce se zaměřuje na využívání moderních technologií programování jako Windows Presentation Foundation nebo Entity Framework.

Tyto technologie budou podrobně vysvětleny a popsány v teoretické části práce společně s dalšími programovacími technikami a metodami využívaných při návrhu aplikace. Jedná se především o metody využívané v objektově orientovaném programování a možnosti vývojového prostředí Visual Studia 2012.

V druhé části práce bude vysvětlena funkčnost a průběh návrhu aplikace, včetně modelových návrhů nebo propojení databáze s uživatelským rozhraním pomocí Entity Frameworku.

Závěrem bude provedena diskuze za účelem zlepšení služeb vyvíjené aplikace a vysvětleny možnosti jejího budoucího rozšíření.

Klíčová slova: Entity framework, C#, skladové hospodářství, desktopová aplikace, visual studio, sklad, WPF, .Net, informační systém

Development of warehouse management software using Entity framework

Abstract

This bachelor thesis is based on the complete design and implementation of an information system dealing with the issue of warehouse management and business processes, after the application is put into operation. The application should be designed for use in smaller businesses or individual entrepreneurs. The work focuses on the use of modern programming technologies like Windows Presentation Foundation or Entity Framework.

These technologies will be explained and described in detail in the theoretical part of the thesis together with other programming techniques and methods used for application designing. Mainly methods used in object-oriented programming and the possibilities of the Visual Studio 2012 development environment.

The second part of the thesis will explain the functionality and design of the application, including model designs or connecting the database with the user interface using the Entity Framework.

Finally, a discussion will be conducted to improve the services of the application being developed and explain the possibilities of its future expansion.

Keywords: Entity framework, C#, warehouse management, desktop application, visual studio, warehouse, WPF, .Net, information system

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

GC – Garbage collector
WPF – Windows Presentation Foundation
XAML – Extensible Application Markup Language
CLR – Common Language Runtime
CIL – Common Intermediate Language

1 Úvod

S rostoucím počtem firem generující zisk za účelem prodeje zboží, je nutné se čím dál častěji zabývat otázkou jeho přechovávání a uskladnění. Vzhledem k množství nabízeného zboží z hlediska efektivity již dnes není možné vytvářet papírové evidence a je tak potřeba vytvářet jejich alternativu, nejčastěji v podobě elektronických informačních systémů, které s sebou přináší řadu výhod a pouze pár nevýhod.

Hlavními výhodami správně navržených elektronických evidencí je jejich efektivita, uspořádanost informací na jednom místě, přístupnost k datům, rychlost, centrální správa nebo různé statistické nástroje.

Volbu informačního systému je nutno před zavedením do provozu dostatečně zvážit a zhodnotit jeho výhody a nevýhody. V opačném případě může docházet k jeho častým změnám a poklesu efektivity firmy.

Jednou z hlavních nevýhod spousty skladových informačních systémů je jejich složitost a uživatel tak musí podstoupit řadu školení než je schopen s programem pracovat. Z pohledu zaměstnavatele, zvláště jedná-li se o malou firmu, se proplacená školení a čas zaměstnance strávený nad pochopením funkčnosti informačního systému může finančně prodražit. Tento projekt se okrajově snaží zaměřit i na tuto problematiku a snaží se zachovat důležité funkce ostatních systémů, co nejjednodušeji je interpretovat uživateli a zkrátit tak čas, který je potřeba věnovat školení pro práci s informačním systémem.

Práce se ovšem zaměřuje především na využití moderních programovacích technologií pro práci s databázemi a návrh aplikací. Jedná se o technologie Entity Framework a WPF, které mají usnadnit a urychlit vývoj aplikací a jsou detailně popsány v teoretické části této práce.

2 Cíl práce a metodika

2.1 Cíl práce

Práce se zaměřuje na problematiku využití Entity Frameworku při vývoji desktopových aplikací využívajících lokální databázi a technologii Windows Presentation Foundation.

Cílem práce je navrhnout a implementovat aplikaci sloužící k jednoduché evidenci skladového hospodářství v malé firmě, případně jednotlivým živnostníkům. Aplikace by měla sloužit jako alternativa k vedení papírové evidence skladového hospodářství a umožnit tak jednodušší, rychlejší a efektivnější zpracovávání firemních procesů napříč jednotlivými složkami firmy. Dílčím cílem bude popsat Entity Framework a možnosti jeho využití.

2.2 Metodika

Metodiky zpracování teoretické části spočívá ve studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Bude popsána technologie Entity Framework a možnosti jejího využití při tvorbě aplikací.

V praktické části bude provedena analýza a následná implementace desktopové aplikace skladového hospodářství.

Při návrhu a implementaci bude využito standardních nástrojů softwarového inženýrství. Implementace bude provedena v jazyce C# v prostředí .NET a bude využívat Entity Framework pro objektové mapování dat z relační databáze.

Aplikace bude otestována a budou shrnuty poznatky z jejího vývoje a navrženy případné další možnosti budoucího rozvoje.

3 Teoretická východiska

3.1 Programovací jazyk C#

C# je vyšší objektově orientovaný programovací jazyk. Oproti nižším programovacím jazykům se svou mírou abstrakce snaží přizpůsobit psaní zdrojového kódu programu myšlenkovému modelu člověka. Začínající programátoři si jej proto často volí jako svůj první programovací jazyk.

Jeho první verze byla uvolněna v roce 2002 společně s platformou .Net. Jeho základ v mnohém vychází z jazyků C++ a Java a je tedy nepřímým nástupcem jazyka C, z něhož čerpá syntaxi, ale v mnoha ohledech je bližší programovacímu jazyku Java.

„V .Net lze vyvíjet v několika programovacích jazycích, z nichž je C# nejmodernější a uzpůsoben právě pro .Net“ (Čápka, 2012). Jazyk C# je též integrován jako součást vývojového prostředí Visual Studio.Net.

Je určen pro vývoj desktopových formulářových programů, webových aplikací a služeb nebo mobilních a databázových aplikací.

3.1.1 Objektově orientované programování

Objektově orientované programování je paradigmatem při vývoji softwaru, jehož filosofií je znovupoužitelnost, menší chybovost a přehlednost kódu. Je zde snaha co nejpřesněji přenést a namodelovat objekty reálného světa a napodobit jejich abstrakci a chování.

Objektově orientované programování je postaveno na objektech. Z hlediska lidského vnímání lze objekt obecně definovat jako vše co je hmotné. V programování objekt vyjadřuje trochu abstraktnější pojem a lze jej tedy charakterizovat jako entitu vykazující určitý druh chování. Může jím být například databázové připojení. „Objekty jsou potom obrazem objektů skutečného světa. Mají své vlastnosti a mohou vykonávat určité činnosti.“ (Dařena, 2004)

Šablonou při volání objektů jsou třídy. „Třídu si můžeme představit jako předpis pro vytvoření objektů. Je možné ji tedy považovat za popis množiny vlastností a činností definujících daný objekt. Objekty patřící do určité třídy, resp. vytvořené na základě třídy, jsou pojmenovány jako instance této třídy. Pojmy instance třídy a objekt je možné ve většině případů považovat za synonyma.“ (Martinů, 2017)

Třída popisuje vlastnosti i chování objektu. Vlastnosti jsou ukládány do proměnných, které drží stavy objektu. Hovoříme o nich jako o atributech objektu. Tyto proměnné lze omezit pouze pro čtení nebo zápis, či jejich modifikaci zcela zakázat. Metody pro získání hodnoty nazýváme gettery a metody pro zápis settery. Visual Studio dokáže gettery i settery automaticky vygenerovat. Nemluvíme pak již o atributech, ale o vlastnostech. Vlastnosti nám specifikují co nastane při čtení nebo modifikaci atributu. Třídy je možné mezi sebou dědit. Jedná se o poměrně častý jev. Znamená to, že nově vzniklá třída zdědí vlastnosti a funkce původní rodičovské třídy a upraví nebo přidá své vlastní.

Dalším prvkem tříd jsou metody. „Metody v objektově orientovaném programování bychom mohli přirovnat k funkcím či procedurám„ (Martinů, 2017). Metody ovlivní, jak volaný objekt bude reagovat na zadané podměty. Rozlišujeme dva základní druhy metod. Metody typu void, také nazývané jako procedury, a metody s návratovou hodnotou. Procedury jsou čistě metody vykonávající funkci, aniž by vracely reálnou výslednou hodnotu. Jejich využití je různé. Od připojení do databáze po složité logické procedury. Potřebujeme-li z metody získat konkrétní výstup, např. výsledek matematické rovnice, použijeme metody, které nám jej poskytnou. K tomu slouží metody s návratovou hodnotou. Tyto metody musí být vždy zakončeny návratovým příkazem return.

„Konstruktor se řadí ke speciálním metodám třídy. Používá se ve chvíli, kdy se vytváří instance třídy. Konstruktor má vždy stejný název jako třída, pro kterou je konstruktor definován“ (Martinů, 2017).

Základním stavebním prvkem programování jsou datové typy. „Proměnná je paměťové místo, kde je uložena hodnota. Proměnné v programovacích jazycích ukládají hodnoty, které se mohou během provádění programu měnit. Pomocí nich může program číst nebo měnit hodnotu uloženou v paměti“ (Clark, 2013). Rozhodují o velikosti a vlastnostech hodnot, kterých smí proměnná nebo konstanta nabývat a které jsou pro každý datový typ specifické. Dělí se na dva základní prameny:

- **Jednoduché** – datové typy, které již nelze dále členit
- **Složené** – datové typy, které jsou složeny z jednoduchých datových typů

Množina sdružující prvky stejného datového typu se nazývá datová struktura. Spadá do ní např. pole nebo list.

Základní datové typy			
Název	**Hodnota**	**Interval**	**Velikost**
Boolean	Logické stavy	True, False	1 Bit
Char	Jeden znak	Unicode znaky	8 Bit
Int	Celá čísla	-2^{31} až 2^{31}	32 Bit
Float	Desetinná čísla	$3{,}4*10^{-38}$ až $3{,}4*10^{38}$	32 Bit
Double	Desetinná čísla s vyšší přesností	$1{,}7*10^{-308}$ až $1{,}7*10^{308}$	64 Bit
String	Řetězec znaků	Variabilní	Variabilní

Tabulka 1 – Seznam základních datových typů

3.1.2 Vlastnosti jazyka

Programovací jazyk C# má stanovená pravidla, která musí programátor při psaní zdrojového kódu dodržovat pro jeho správnou funkčnost. Řeč je o jeho syntaxi a sémantice. Ač jsou slova často zaměňována, každé z nich má zcela specifický a rozdílný význam.

„Příkazy v C# následují přesně definovaný soubor pravidel popisujících jejich formát a konstrukci. Tyto pravidla jsou souhrnně označována jako syntaxe. Naopak, specifikace toho co se děje, je souhrnně označována jako sémantika“ (Sharp, 2012).

Sémantika je obecně věda zabývající se významem slov. Původní znění slova sémantika bylo užíváno v oboru lingvistiky. S příchodem programovacích jazyků se i v oboru programování musela řešit otázka sémantiky kódu. V programování se sémantika zabývá pravidly pro správně zvolené instrukce či elementy v závislosti na konkrétním kontextu a jeho významu. Neznamená to tedy, pokud se výsledek programu vykoná bez chyby, že je sémanticky správně. Aby stroj lépe pochopil význam zdrojového kódu, je potřeba dodržovat pravidla sémantiky. Silný důraz na sémantiku je při tvorbě webových stránek.

Jazyk C# rozeznává rozdíly mezi velkými a malými písmeny, neboli je tzv. *case sensitive*. Tuto vlastnost lze řadit mezi příčiny syntaxe daného jazyka. Opět se jedná o soubor pravidel, které ovšem nezkoumají význam instrukcí, nýbrž jejich formu. Definuje, jakým způsobem by měl být zapsán např. název proměnné, nebo že každá instrukce by měla končit středníkem atd. Vzájemný vztah mezi syntaxí a sémantikou tvoří tzv. čistý kód.

Tak jako je tomu u spousty programovacích a skriptovacích jazyků, je i v jazyce C# běh programu řízen vykonáváním posloupností jednotlivých příkazů. Často není žádoucí, aby byl zdrojový kód interpretován řádek po řádku, ale aby některé řádky byly buď přeskočeny, nebo opakovaně vykonány. K tomu využíváme tzv. řídící struktury. Dělíme je následovně:

- **Sekvence**
- **Selekce** – `if-else, switch`
- **Iterace** – `for, while, foreach`

Sekvence je jistě nejprimitivnější řídící strukturou. Provádí příkazy tak, jak následují po sobě. Příkazem sekvence může být jakýkoli příkaz, který nespadá ani do jedné z kategorií selekce nebo iterace.

Selekce je další řídící strukturou, která ovlivní tok dat za běhu programu. Selekce je synonymem pro výběr, kterým je právě i v programování řízena. „Výběrové podmínky podmíněně řídí tok provedení programu“ (Albahari, a další, 2012). Zpracování části kódu je podmíněno splněním podmínky. Není-li podmínka splněna, blok je přeskočen. Bývá také označována za větvení programu. Pro větvení programu se používají bloky `if-else` nebo blok `switch`.

První z nich funguje jako klasická podmínka. `If` v kulatých závorkách definuje svou podmínku. Je-li tato podmínka splněna, provede se celý kód tohoto bloku. Skončí-li výsledek podmínky jinak než je definován v bloku `if`, přeskočí do doplňkového bloku `else`. Pokud neexistuje blok `else`, je zcela přeskočen a jsou vykonávány další instrukce. `If` bloky je možné do sebe vnořovat.

Druhý typ selekce pracuje jako jakýsi přepínač. `Switch` se rozhoduje na základě vstupů typu `int` nebo `char`, které mu jsou předkládány. Obsah jeho bloku je rozdělen do větví. Větev vždy začíná klíčovým slovem `case` s konstantou možného vstupu `switche`. Dále následuje dvojtečka, která má za úkol oddělit případně vykonávané instrukce, je-li konstanta větve shodná se vstupem `switche`. Konec větve se ukončí příkazem `break`.

Iterační struktury neboli cykly slouží k několikanásobnému opakování příkazů. Při používání cyklů by měla být zajištěna jejich konečnost, aby nedocházelo k zacyklení programu. To by znamenalo, že program vstoupí do cyklu, ze kterého již nemůže, z důvodu špatně navrženého cyklu, vystoupit.

Nejznámějším a nejlépe chápatelným cyklem je `for` cyklus. Vše se definuje již v samotné hlavičce cyklu. Jako první je nastaven počáteční hodnota proměnné, od které se odvíjí opakování cyklu. Nejčastěji je nastavena na hodnotu 0. Dále je vytvořeno omezení, které porovnáním aktuální hodnoty počáteční proměnné s další hodnotou vyhodnotí, zda se tělo cyklu provede. Poslední část cyklu vymezí, o kolik se každým opakováním posune hodnota počáteční proměnné.

Dalším z cyklů je cyklus `while`, který lze ještě doplnit o blok `do`. `While` opět tvoří omezení, které udává, zda obsah jeho bloku bude proveden. Právě zde hrozí největší nebezpečí nekonečného cyklu, jelikož jeho konečnost musí být ošetřena v jeho těle. Doplňující blok `do` je umístěn ještě před samotný `while` a je používán za účelem alespoň jednou provést tělo cyklu.

Poslední varianta z cyklů by se dala prohlásit za paralelu k cyklu `for`. Ovšem, pracujeme-li s poli, jeví se jako rozumnější využívat předností cyklu `foreach`. Ten již po každém provedení těla cyklu nevrací index aktuální položky, ale pracuje rovnou se všemi prvky z množiny a vrací už jen požadovanou položku.

3.1.3 .Net Framework

„.Net Framework je sbírka základních tříd určených k poskytování společných služeb potřebných ke spuštění aplikací“ (Clark, 2013).

.Net je framework, který sdružuje technologie, funkce a knihovny pro zjednodušení vývoje, určený modernějším programovacím jazykům od Microsoftu. Společně s jazykem C# a prostředím Visual studio vytváří již jakýsi standard. .Net má v sobě integrováno celé běhové prostředí CLR. Nezávisle na programovacím jazyku je tak aplikace nakonec vždy kompilátorem přeložena do CIL (viz. Kapitola o virtuálních strojích). Výběr jazyka je proto relevantní.

„Myšlenkou .Net Frameworku je, že místo kompilace programu C# do strojového kódu se program zkompiluje do mezikódu, který je podobný jako bytecode pro Javu. Teoreticky by šel program spustit v libovolném počítači, který má rozhraní .Net Framework. Ve skutečnosti, jediné počítače, které .Net Framework podporují jsou operační systémy Windows. To znamená, že program naspaný v rámci .Net Frameworku je omezen pouze pro systémy Microsoft Windows“ (Wang, 2008).

Poslední oficiálně vydanou verzí je .Net Framework 4.7.2, současně vyvíjen s .Net Core v aktuální verzi 2.2, který jako hlavní přednosti přináší možnosti multiplatformního

vývoje a v současnosti podporuje operační systémy Windows, MacOS a většinu linuxových distribucí. Vzhledem k tomu, že je stále ještě na počátku svého vývoje, v zájmu předejít možným komplikacím je vhodné zvážit, zda pro psaní aplikace nepoužít ověřený .Net Framework.

.Net Framework		
Verze	**Rok představení**	**Integrace do vývojového prostředí**
.Net 1.0	2002	Visual Studio .Net
.Net 1.1	2003	Visual Studio .Net 2003
.Net 2.0	2005	Visual Studio 2005
.Net 3.0	2006	Blend for Visual Studio
.Net 3.5	2007	Visual Studio 2008
.Net 4.0	2010	Visual Studio 2010
.Net 4.5	2012	Visual Studio 2012
.Net 4.5.1	2013	Visual Studio 2013
.Net 4.5.2	2014	
.Net 4.6	2015	Visual Studio 2015
.Net 4.6.1	2015	Visual Studio 2015 UPDATE
.Net 4.6.2	2016	
.Net 4.7	2017	Visual Studio 2017
.Net 4.7.1	2017	Visual Studio 2017
.Net 4.7.2	2018	Visual Studio 2017

Tabulka 2 – Verze .Net Frameworku

3.1.4 Knihovny

Je možné je používat v jednom nebo ve více programech současně. Frameworky obecně, včetně samotného .Netu, jsou založeny právě na knihovnách. „Hlavním rozdílem mezi frameworky a knihovnami je, že framework tvoří ucelený komplexní soubor řešící široké spektrum úloh, ale knihovny se zaměřují jen na řešení specifických úloh“ (Clark, 2013). Mají programátora oprostit od řešení problémů, které již někdo v minulosti vyřešil v podobě knihoven. Při vývoji v týmu slouží k efektivitě práce.

Základně se knihovny rozdělují na dva typy:

- **Statické knihovny (.lib)**
- **Dynamické knihovny (.dll)**

Statické knihovny jsou přikládány k projektu. Jsou zátěží na operační paměť, neboť jsou načítány pro každou aplikaci zvlášť a musí se tak vždy zapsat do operační paměti. Do programu jsou linkovány při kompilaci do jeho binárního souboru a tvoří tak jeho součást. Vzhledem k tomu, že jsou součástí programu, aplikace bude spustitelná i na dalších zařízeních nezávisle na dostupných knihovnách druhého stroje. Jejich aktualizace je však poměrně pracná. Tím, že jsou linkovány při kompilaci programu, tak aby se projevila změna, je potřeba znovu nalinkovat všechny programy kde se knihovna používá. Jsou tak vhodné spíše, jedná-li se o použití v jednom programu.

U dynamických knihoven je linkování prováděno až za běhu programu. V principu jsou načteny do operační paměti jen jednou a jsou využívány i více aplikacemi najednou. Výhodou je jejich snadná údržba. Pro jejich změnu stačí přepsat jen tuto jednu knihovnu, kdy jsou na ni spustitelné soubory následně odkazovány. Aby byl program spustitelný, musí mít dynamické knihovny k dispozici. Měly by proto být šířeny s programem.

3.1.5 Garbage collector

„Garbage collector“ je metoda pro automatickou správu paměti procesů. V některých programovacích jazycích, včetně jazyků C# nebo Java, je vyžadován již ve specifikaci jazyka. Automatickým uvolňováním paměti při psaní zdrojového kódu programátorovi odpadá povinnost manuálně ošetřit uvolňování objektů, které již nebudou dále potřeba. „GC má následující hlavní úkoly: zkontrolovat, kdy se alokovaná paměť již nepoužívá, uvolnit ji a zpřístupnit jej pro přidělení nových objektů.“ (Nakov, a další, 2013)

U starších jazyků jako jsou C a C++ se správa paměti realizuje manuálně. Při manuální správě paměti programátor zodpovídá za to, kdy bude paměť uvolněna. Programátor tak musí optimalizovat svůj zdrojový kód tak, aby nedocházelo k zaplnění paměti.

Nevýhodou automatické správy paměti je, že GC ke své práci potřebuje procesorový čas, aby mohl rozhodnout o tom, zda může být místo v paměti uvolněno či nikoliv. To způsobuje problémy při programování real-time systémů.

3.1.6 Virtuální stroj

Procesor každého počítače ve svém základním principu dokáže zpracovat pouze velmi malé množství strojových instrukcí. Tyto instrukce zpracovává na úrovni strojového kódu. Strojový kód se řadí do tzv. první generace programovacích jazyků. Bývá zpravidla zapsán v hexadecimálním tvaru a překládán v binární podobě. Takto napsaný program byl v minulosti špatně čitelný, nesrozumitelný a závislý na instrukční sadě konkrétního procesoru. Dnes se již přímo ve strojovém kódu neprogramuje.

Strojový kód byl v polovině 20. století nahrazován druhou generací jazyků, do níž se řadí *Assembler*. Assembler, též nazýván jako jazyk symbolických adres, si kladl za cíl usnadnit programátorovi čitelnost kódu tím, že původně číselný zápis strojové instrukce nahradil symbolickým slovním označením. Vznikl proto i první překladač, který symbolickým označením přiřazoval skutečné adresy. I v assembleru se dnes programuje velmi zřídka, např. části operačního systému.

3. generace jazyků se již relativně přiblížila vnímání člověka. Kód byl čitelnější a lépe udržitelný. Patří mezi ně jazyky jako je Java nebo C#, které se dnes běžně používají pro vývoj aplikací. Pro správnou komunikaci s procesorem je nutné jej převést do strojového kódu. Jazyky se tak dělí na kompilované, interpretované a jazyky s virtuálním strojem.

Kompilované jazyky ke svému překladu mezi zdrojovým kódem a strojovým kódem využívají kompilátor. Kompilací se přeloží celý program najednou do strojového kódu. Tento způsob překladu umožní následně velmi rychlý běh programu. Opět zde ale hrozí problémy s přetečením paměti nebo závislostí na vyvíjené platformě.

Filozofie interpretovaných jazyků je přesným opakem jazyků kompilovaných. Překlad neprobíhá najednou, ale za běhu programu po jednotlivých instrukcích. To s sebou samozřejmě nese mírné zpomalení celého běhu aplikace. Výhodou je pak stabilita či přenositelnost na jiné platformy.

Spojením obou typů výše uvedených přístupů vznikne virtuální stroj. Ten je v současnosti explicitně nejrozšířenějším přístupem pro vývoj aplikací. Aplikace vyvíjené pro virtuální stroj jsou spustitelné na jakékoli platformě, pro kterou existuje příslušný virtuální stroj a odpadá tak nutnost psát jednotlivé verze aplikací pro jednotlivé operační systémy. Virtuální stroj kombinuje přednosti obou zmíněných typů jazyků kompilátoru a interpreta. Nejznámějším virtuálním strojem je bezpochyby Java Virtual Machine pro programovací jazyk Java. Do jazyků využívající pro svůj běh virtuální stroj se řadí i C#

s obdobou Java Virtual Machine pod názvem Common Language Runtime. „CLR spravuje prováděný kód a poskytuje vrstvu abstrakce mezi kódem a operačním systémem“ (Clark, 2013). Zdrojový kód je v první fázi kompilátorem přeložen do tzv. mezikódu. V Javě je tento mezikód nazýván Java *bytecode*. Microsoft ho u svých jazyků označuje jako Common Intermediate Language (CIL). Chápat jej můžeme jako strojový kód se zjednodušenou instrukční sadou. Kód v tomto tvaru poté dokáže virtuální stroj již relativně rychle interpretovat do nativního strojového kódu.

Virtuální stroj řeší problematiku na úrovni komunikace hardwaru a instrukčních sad procesoru. Ačkoliv je C# a .Net určen primárně pro systém Microsoft Windows, je možné programy vyvíjené v jazyce C# spouštět i na jiných systémech s odlišnou architekturou jako Linux, MacOS a další. Mono je opensource projekt obsahující nástroje emulující .Net a práci s ovládacími prvky WinForms. Momentálně jej začíná nahrazovat .Net Core od Microsoftu v aktuální verzi 2.2.

3.2 Technologie WPF

Windows Presentation Foundation je moderní grafické rozhraní, které je součástí .Net frameworku a nahrazuje původní návrhové prostředí Windows Forms. „V porovnání s předchozími technologiemi je to radikální změna, která přichází s inovativními funkcemi jako je zabudovaná hardwarová akcelerace a nezávislost na rozlišení obrazovky“ (MacDonald, 2012). Technologie WPF si klade za primární cíl oddělit oddělit aplikační logiku od grafického návrhu programu za pomoci jazyka *XAML*.

Rendering veškeré grafiky aplikací založených na WPF probíhá skze *DirectX,* tedy přes grafickou kartu. To platí v případě, že grafická karta podporuje aplikační rozhraní DirectX 7 a vyšší. V ostatních případech je renderováním zatěžován procesor buď částečně, v horším případě se CPU stará o vykreslování celého designu aplikace. Rendering pomocí grafické karty značně urychlí běh programu, neboť není vykreslováním uživatelského prostředí zatěžován procesor.

WPF se svým přístupem zaměřuje na graficky náročnější a bohatější aplikace. Jazyk *XAML* umožňuje pracovat s vektorovou grafikou, obdobně, jako je tomu u webových stránek.

3.2.1 Xaml

Myšlenkou vývoje aplikací technologií WPF je oddělit dvě odlišné komponenty. Oddělit logický model od grafického návrhu a vzájemně je mezi sebou provázat. Pro oddělení logiky a designu aplikace je využíván značkovací jazyk *XAML*. Svou syntaxí připomíná ostatní známé značkovací jazyky jako *XML* nebo *HTML* resp. *XHTML* z nichž také vychází. *XAML* je však primárně koncipován pro tvorbu vizuálního obsahu.

„*XAML* není jedinou možností. Ve skutečnosti lze celou aplikaci vytvořit pomocí C# nebo Visual Basic, případně využít libovolný jazyk .Net bez jediné značky *XAML*. To by však bylo mnohem obtížnější a náchylné k chybám s vysokými náklady na údržbu“ (Yosifovich, 2012). V porovnání se staršími možnostmi návrhu uživatelského rozhraní je *XAML* svobodnější, přehlednější a lépe udržovatelnější.

Jeho obsah je tvořen elementy založenými právě na *XML*. Pomocí elementů komunikuje s objekty, které jsou součástí .Net frameworku, případně s objekty třetích stran. Jelikož má *XAML* stromovou strukturu, je možné provádět vnořování elementů. Stromová struktura znamená, že dokument obsahuje vždy jeden kořenový element, do kterého jsou umisťovány další elementy, které opět mohou obsahovat další elementy. Tímto způsobem je možné poměrně efektivně rozšiřovat ovládací prvky.

3.2.2 Výhody práce s WPF

Pro jakoukoli vyvíjenou aplikaci není striktně předepsáno, jaká technologie je pro její vývoj vhodnější. Je třeba si nejprve dobře stanovit, co konkrétně se od aplikace očekává a k jakým účelům bude sloužit. Ačkoli technologie WPF přináší spoustu inovací, nemusí být její použití vždy přínosem, i když počet výhod převládá. Jedná-li se o menší aplikaci, která si vystačí s omezeným množstvím ovládacích prvků, je *WinForms* stále pohodlnou alternativou. Vzhledem k tomu, že prvky jsou vytvořeny pro použití s jasným scénářem, práce s nimi je často pohodlnější.

WPF ovšem přináší svobodu kódu v mnoha ohledech. Mimo to, že je lépe přizpůsoben současným normám, při jeho plném ovládnutí získá vývojář spoustu přidané hodnoty v podobě stylů, *bindingu*, tvorbu webových aplikací až po celkové oddělení logické a prezentační vrstvy.

3.2.3 Styly

WPF se vyznačuje možností tvořit graficky bohaté aplikace. Je však dobré udržovat v uživatelském rozhraní jednotný vzhled. Aby nedocházelo k opakovanému psaní téhož kódu, používají se styly. Styly nám umožní nadefinovat vizuální vjem prvku a následně ho opakovaně prvku implementovat. Kromě již předdefinovaných ovládacích prvků dovoluje WPF tvorbu vlastních prvků různých tvarů, barev a efektů.

„Styly ve WPF svou podstatou použití kopírují funkci standardu *CSS* na webových stánkách. Stejně jako *CSS* jsou definovány v samostatném souboru, který je svázán s aplikací a umožňují definovat společnou sadu vlastností formátování a aplikovat je v celé aplikaci a zajistit tak její konzistenci“ (MacDonald, 2012). Přistupovat k nim lze pomocí identifikátoru x:Key. Základním elementem je tzv. *Setter*. Ten definuje hodnotu pro konkrétní vlastnost prvku. Vlastností prvku rozumíme např. barvu pozadí, písmo atp.

Je-li vhodné prvku nastavit nějaký speciální interaktivní efekt, je možné elementy *Setter* ještě obohatit spouštěči reagujícími na konkrétní události prvků jako např. kliknutí nebo přejetí myší přes prvek. Element s identifikátorem *Triggers*, stejně jako *Setter*, nastavuje název události s příslušnou hodnotou.

Obdobně jako se v C# dědí třídy, i styly lze mezi sebou dědit. Nutným kritériem dědícího stylu je, aby se v elementu Style parametrem *BasedOn* odkazoval na rodičovský styl, ze kterého se chystá dědit. Vlastnosti zděděného stylu je možno následně ještě modifikovat nebo přidat nové.

3.2.4 App.config

Jednou z vlastností .Net Frameworku je možnost vytvářet konfigurační soubory. Jedná se v podstatě o klasický XML dokument, kam jsou ukládány potřebná nastavení aplikace.

Vzhledem k tomu, že .Net poskytuje velmi efektivní práci s databázemi, je možné zapsat připojovací řetězec přímo do konkrétního elementu konfiguračního souboru. Tento řetězec s charakteristickým označením connectionStrings obsahuje informaci o zdroji dat a údaje pro přístup k bázi dat, kterou kromě klasické databáze může být i tabulka nebo textový soubor.

3.3 Entity Framework

„Entity Framework je strategickým přístupem společnosti Microsoft k technologii přístupu k datům pro vytváření softwarových aplikací. Na rozdíl od dřívějších technologií pro přístup k datům, Entity Framework spolu s Visual studiem dodává komplexní ekosystém založený na modelu, který umožňuje vyvíjet širokou škálu datově orientovaných aplikací včetně desktopových, webových a aplikací založených na službách“ (Driscoll, a další, 2013).

Entity Framework je framework pevně integrován do Visual studia s univerzálním přístupem k datům. Technika má ucelit pohled na relační databáze a objektově orientované programování, přistupovat k datům z relačních databází jako k objektům a v ideálním případě tak zcela odstínit vývojáře od potřeby používání sql dotazů, které si framework generuje sám na pozadí. Je založen na ADO.Net datovém modelu v případě přístupu *Database First* a *Model First* nebo DbContextu v případě *Code First*.

Datový model generuje automaticky Entity Data Model Designer ve Visual studiu na základě tabulek databáze. V grafické podobě nese informaci o tabulkách uložených v databázi a vazbách mezi nimi. „Datový model v aplikaci popisuje strukturu obchodních objektů. To je, jako by bylo uděleno povolení k restrukturalizaci databázových tabulek a pohledů v databázi, aby tabulky a vztahy vypadaly jako místo normalizovaného schématu, které bylo navrženo správci databází“ (Lerman, 2010).

Dalším nástrojem pro objektivně relační mapování je např. NHibernate.

3.3.1 Architektura

Entity Framework se z hlediska přístupu použití rozděluje dvěma různými směry. Každá z metod stojí na své vlastní architektuře. Jsou to přístupy z pohledu:

- **Code First**
- **Database First**
- **Model First**

Model Database First je z historického pohledu vývoje nejstarší technologií, která byla přístupná již od první verze Entity Frameworku, která tím byla podstatně omezená. Při použití metody Database First je zapotřebí mít již předem vytvořenou databázi, aby z ní Entity Framework mohl automaticky vytvořit potřebné instance tříd, včetně kontextové třídy a jejich vlastností pro objektové mapování databáze. Kontextová třída je mostem mezi třídou objektů a databází, zodpovídá za jejich interakci. Vzhledem ke změnám, ke

kterým dochází při úpravách v databázi, lze aktualizovat koncepční model v závislosti na těchto změnách databáze. Aktualizaci změn je možné automaticky generovat přes edmx model funkcí Update Model from Database.

Do 4. verze Entity Frameworku přibyl další přístup objektového mapování pod názvem Model First, která je opakem předchozího přístupu Database First. Jako první se přes návrhář Visual studia navrhne koncepční model tabulek a vazeb, z které se následně generuje skript pro vytvoření kompletní databáze.

Brzy poté vývojáři vydali další menší aktualizaci a ve verzi Entity Frameworku 4.1 byl přidán další a momentálně nejmodernější přístup Code First. Jeho záměrem je oprostit se od tvorby koncepčních modelů, které zapouzdřují modelové a mapové informace. Vše je tvořeno v třídách programovacího jazyka, z kterých je Entity Frameworkem vygenerováno schéma databáze a třída kontextu přes kterou bude probíhat komunikace s databází. Přístup Code First je doplněn o funkci migrace, kterou lze měnit původní strukturu databáze vytvořenou přístupem Code First.

3.3.2 Verze Entity Frameworku

Entity Framework		
Verze	**Rok představení**	**Popis**
EF 3.5	2008	První verze EF, vychází společně s .Net 3.5, přináší ORM s modelem Database First
EF 4	2011	Vychází společně s .Net 4, nový přístup objektově relačního mapování Model First
EF 4.1	2011	První oficiálně dostupná verze EF přes NuGet
EF 4.3	2012	Rozšíření přístupu Code First o novou funkci Code First Migration
EF 5	2012	Větší stabilita a optimalizace, lepší začlenění do Visual studia 2012
EF 6	2013	
EF Core 1.0	2016	Vychází společně s prvním .Net Core Frameworkem, první multiplatformní EF
EF Core 2.0	2017	

Tabulka 3 – Verze Entity Frameworku

3.4 Visual Studio

Microsoft Visual studio je jedním z nejpokročilejších vývojových prostředí na světě od společnosti Microsoft. První oficiálně vydanou verzí Visual studia byla verze z roku

1997, jež nesla název Visual studio 97 a Microsoft zde dohromady sdružil mnoho ze svých programovacích nástrojů. Nové verze vychází přibližně jednou až za dva roky. Ke dnešnímu dni je nejaktuálnější dostupnou verzí Visual studio 2017, které by ještě v roce 2019 mělo nahradit již potvrzené Visual studio 2019.

Větším firmám (tj. více než 250 počítačů nebo roční výnos 1 milion dolarů) jsou poskytovány placené licence. Cena za licenci se pohybuje v závislosti na zakoupené verzi produktu v rozmezí od 15 000 do 90 000 korun. Lze využít i bezplatné edice Community, která je srovnatelná s verzí Professional a je určena jednotlivým vývojářům, školní výuce, vědeckému výzkumu a open source projektům.

V rámci vzdělávacího programu od Microsoftu, jsou studentům a vyučujícím poskytovány bezplatné licence jinak placeného softwaru skrze program DreamSpark, včetně sady Microsoft Visual Studio.

„Visual Studio nabízí dva pohledy na grafickou aplikaci: zobrazení návrhu a zobrazení kódu. Textový editor je využíván k úpravě a udržování kódu a programu logické aplikace a pro grafický návrh okno návrhové zobrazení, které slouží k rozložení uživatelského rozhraní. Lze přepínat mezi oběma pohledy, kdykoli chcete“ (Sharp, 2012).

3.4.1 Grafický designer

Jedním z mnoha nástrojů, kterým Visual studio disponuje, je grafický návrhář implementovaný již do základu vývojového prostředí. Součástí designéru je panel nástrojů, známější pod názvem *Toolbox*. V tomto panelu se nachází ovládací prvky, které je možno přidat do projektu. Visual studio obsahuje několik grafických návrhářů v závislosti na vyvíjené platformě. Panel nástrojů zobrazí pouze prvky, které lze použít v aktuálním návrháři. Jsou zde obsaženy prvky jako *Button, Label, TextBox* atd. Prvky *toolboxu* fungují na principu *drag and drop*, neboli táhni a pusť.

3.4.2 Editor kódu

Visual studio kromě perfektního grafického designéru obsahuje samozřejmě i velice propracovaný a návykový editor kódu. Editor kódu má v sobě implementovanou funkci našeptávače *IntelliSense*. „Seznam našeptávače obsahuje všechna klíčová slova a datové typy, které jsou platné z vyplynutí daného kontextu“ (Sharp, 2012). Ten zvládá našeptávat v závislosti na dostupných frameworkcích a knihovnách pro aktuální aplikaci

nebo přímo ze zdrojových kódů připojených k projektu. V případě potřeby jej lze snadno vypnout klávesovou kombinací Ctrl + mezerník.

Pro editor existuje spousta přednastavených klávesových zkratek. V možnostech nastavení je možnost tyto klávesové zkratky rozšířit, zakázat nebo zaměnit pro lepší komfort během psaní zdrojového kódu.

Pro lepší komfort a lepší orientaci v kódu dokáže editor barevně rozlišit prvky syntaxe, podtržením upozornit na chyby v zápisu, automaticky strukturovat text či sledovat provedené změny v podobě žlutého pruhu v levém okraji editoru. Přejít k definici je funkcí, která má usnadnit hledání v kódu tím, že se po zvolení barevně odlišeného elementu lze přes pravé tlačítko myši přesunout do místa definice daného elementu.

3.4.3 Ladění

Během vývoje programů lze využít řadu ladících nástrojů přímo v prostředí Visual studia. Při psaní zdrojového kódu programátoři naráží na nespočet výjimek a chyb. Chyby ve zdrojovém kódu mohou způsobit zpomalování, padání nebo absolutní nefunkčnost aplikace. Hledání těchto chyb může programátorům zabrat i dlouhé hodiny práce. Odvádí jej to poté od řešení skutečných programovacích paradigmat. Pomocí ladících nástrojů je možné tyto chyby poměrně snadno objevit a opravit.

Výjimkou se označují neočekávané situace, které mohou nastat při běhu programu. Taková situace může nastat například, chceme-li dělit nulou, odmocnit záporné číslo apod. Z hlediska ladění programu pohlížíme na výjimku jako na zprávu poskytující chybové hlášení o právě proběhlé neočekávané situaci. Zachycení této chybové zprávy je prováděno klíčovým slovem `throw`, pro lepší pochopení a následné ošetření vyvolané výjimky. K ošetření těchto chybových stavů jsou využívány tzv. bloky. Blok `try` je určen pro vykonávání metod, u nichž se předpokládá vyvolání výjimky. Selže-li pokus o provedení `try` bloku, je zaznamenána výjimka a pokus o provedení následujícího jednoho nebo více bloků `catch`. „Blok `catch` má přístup k objektu Exception, který obsahuje informace o chybě. Použití bloku buď kompenzuje chybu, nebo vrátí výjimku“ (Albahari, a další, 2012).

Krokování programu je dalším velice užitečným a hojně využívaným ladícím nástrojem. Jeho podstata spočívá v procházení programu krok po kroku za běhu aplikace. Lze tak zobrazit obsah proměnných a zaměřit se na chybnou část kódu v daný moment.

Jedná se převážně o chyby logického charakteru. Krokováni se ve Visual studiu provádí pomocí kláves F10 a F11.

Není-li žádoucí krokovat program od začátku do konce, je možnost do zvolené části programu umístit *breakpoint*. Jeho volba umístění je nejčastěji volena v místě s podezřením na chybnou část kódu. Program tak běží až do chvíle, než jej *breakpoint* ve zvoleném bodě zastaví. „Pokud je tedy program dlouhý 10 000 řádků a víte, že problém je někde v posledních 1000 řádcích kódu, není to žádný bod procházejícími těmi prvními 9000 řádky kódu“ (Wang, 2008). Dále je možné krokovat další jednotlivé příkazy.

Do technologií pro ladění zdrojového kódu lze zařadit i pojem *Unit testing.* Smyslem jednotkových testů je otestovat a prověřit funkčnost jednotlivých částí zdrojového kódu. Jejich tvorba je ve Visual studiu relativně nenáročnou záležitostí. Unit testy jsou nejčastěji prováděny u opakovaně používaných metod s předpokladem vyvolání neočekávaných událostí, kde programátor vyžaduje 100% funkčnost. Jsou vytvořeny testovací scénáře, které na základě různých vstupů testují validitu výstupů. Testy jsou následně vyhodnoceny jako úspěšné nebo neúspěšné.

3.4.4 Package manager

Package manager, neboli správce balíčků *NuGet,* je open source balíčkovací systém zaměřený na vývojáře. Funguje jako katalog rozšíření pro .Net. Přes webové rozhraní nebo rozhraní Visual studia programátorovi umožní najít, nainstalovat, odinstalovat a aktualizovat balíčky *NuGet*. Ovládat jej lze pomocí uživatelského nebo konzolového rozhraní na bázi *PowerShellu*.

Skrze uživatelské rozhraní lze procházet vždy aktuální a stabilní verze balíčků nebo mezi nimi pomocí jednoduchých nástrojů filtrovat potřebný obsah. Výhodou konzolového rozhraní je, že zde lze stáhnout jakoukoliv verzi balíčku.

Při instalaci balíčků se vytvoří adresář s názvem *packages*, kde jsou uloženy jednotlivé balíčky, včetně potřebných knihoven a souborů. NuGet zvládá i konfigurace stažených knihoven, včetně všech závislostí.

Kromě užívání již hotových řešení je možnost si vytvářet vlastní balíčky. Při tvorbě nového balíčku jsou definovány dva základní soubory s příponami „.nuspec“, který v podobě XML souboru popisuje vlastnosti vytvářeného balíčku jako popis, verzi, autora atd. a „.nupkg“, ve kterém je definována přesná struktura balíčku.

4 Vlastní práce

4.1 Koncept vlastní aplikace

Tato část práce je zaměřena na praktickém využití teoretických předpokladů z předchozí části. Na konkrétní aplikaci skladového hospodářství bude představen Entity Framework, konkrétně přístup DatabaseFirst, její kompletní návrh, funkcionalita a testování. Aplikace je vyvíjena ve vývojovém prostředí Visual Studio 2012 pro jeho lepší kompatibilitu s LocalDB a je postavena na technologii Windows Presentation Foundation.

4.1.1 Požadavky na aplikaci

Jedním ze zásadních problémů dnešního podnikání, obzvláště pak v oblasti obchodu, je udržovat přehled nad vlastní skladovou evidencí. Při množství skladových položek a jejich častému oběhu není možné vytvářet a uchovávat papírové evidence. Aplikace byla vyvíjena za účelem v elektronické podobě co nejjednodušeji spravovat skladové položky, informovat o jejich stavu v malých firmách a umožnit jejich aktuální přehled. V základu má aplikace umožnit naskladnit položky na sklad, vyskladnit položky ze skladu a provádět jejich přesuny mezi sklady. Aplikace navíc poskytuje generování jednoduchých tisknutelných výstupů v podobě reportů.

4.1.2 Systémové požadavky

Pro správnou funkčnost a plynulost programu jsou zde uvedeny doporučené požadavky na systém:

- Operační systém Windows 7 SP1 (x86 a x64) a vyšší
- Microsoft .NET Framework 4.5
- SQL Server 2012 Express LocalDB
- 200 MB volného místa na disku
- 2 GB paměti RAM

4.2 UI specifikace

Před samotným vývojem aplikace byla zhotovena funkční UI specifikace pro lepší rozvržení aplikace, pochopení vztahů a možných scénářů mezi jednotlivými okny programu během jejich přepínání. Pro návrh logických wireframů byl použit open-source prototypovací nástroj Pencil Project.

4.2.1 Motivace

Motivací k vytvoření vlastního informačního systému pro evidenci skladového hospodářství je poskytnout menším podnikatelům a majitelům malých firem komfortnější správu a přehled nad vlastními hospodářskými prostředky.

4.2.2 Definice cíle

Hlavním cílem informačního systému je vytvoření jednoduché aplikace užívané v malých firmách vhodné i pro osoby se sníženou orientací práce na počítači, která bude poskytovat veškeré důležité nástroje pro elektronickou evidenci skladových položek. Vzhledem k časté změně dat by aplikace měla splňovat podmínku stability a zajistit co největší plynulost aplikace.

4.2.3 Rozvržení oken

1. **Přihlašovací formulář**
   Při přihlašování formulář vyzve uživatele k zadání korektních informací, které uživatel vyplní
2. **Hlavní pracovní plocha**
   Poskytne informační tabuli a na jejím základě budou generována další podokna.
3. **Založení nové skladové položky**
   Umožní vytvořit nebo modifikovat skladovou položku a zobrazí seznam již vytvořených položek.
4. **Naskladnění, vyskladnění, přesuny**
   Vypíše seznam s jednotlivými elektronickými doklady konkrétního procesu a nabídne vytvořit nový nebo upravit stávající, případně umožní sestavit report daného procesu.
5. **Přidat nebo upravit počty položek u procesu**
   Nabídne základní definici informací o dokladu a možnost přiřadit k němu správné množství položek.
6. **Inventura**
   Vypíše stavy položek jednotlivých skladů za dané období a umožní porovnat evidované stavy s jejich reálnými hodnotami.

4.2.4 Personifikace

Informační systém je určen pro:

- Věkovou kategorii 18 – 70 let
- Uživatele, kteří se systémem budou pracovat denně
- Uživatele méně znalé v oblasti IT
- Uživatele znalé v oblasti procesů firmy
- Uživatele se zájmem o obchod
- Majitelé a zaměstnance firmy

4.2.5 Vzorové persony

1. **Ing. Pavel Stejskal**
    - **věk:** 53 let
    - **pohlaví:** muž
    - **povolání:** majitel maloobchodu s potravinami, správce systému
    - **zájmy:** cestování, lyžování, obchod, četba

Ing. Pavel Stejskal je majitelem prodejny a zároveň správcem interního systému. Rád zkouší nová řešení, aby maximalizoval své zisky, spokojenost svých zákazníků a mohl v budoucnu rozšiřovat síť vlastních prodejen. Rád by proto měl přehled o majetku jednotlivých prodejen.

Řešení: Možnost vytváření přehledných reportů a přehled o aktuálních stavech aktiv firmy.

2. **Bc. Eva Mašková**
    - **věk:** 32 let
    - **pohlaví:** žena
    - **povolání:** účetní, manager prodejny
    - **zájmy:** Jóga, fotografování

Bc. Eva Mašková ve firmě zastává pozici manager prodejny a zároveň působí jako účetní. Stará se o chod prodejny, vedení účetní knihy a řízení vnitropodnikových procesů pro manažerská rozhodnutí a další rozvoj firmy.

Řešení: Zakládání elektronických účetních dokladů s informací o cenách, jejich zpracování a převedení skladových položek na sklad.

3. **Michal Svoboda**
    - **věk:** 25 let
    - **pohlaví:** muž
    - **povolání:** skladník
    - **zájmy:** počítačové hry, sport, automobily

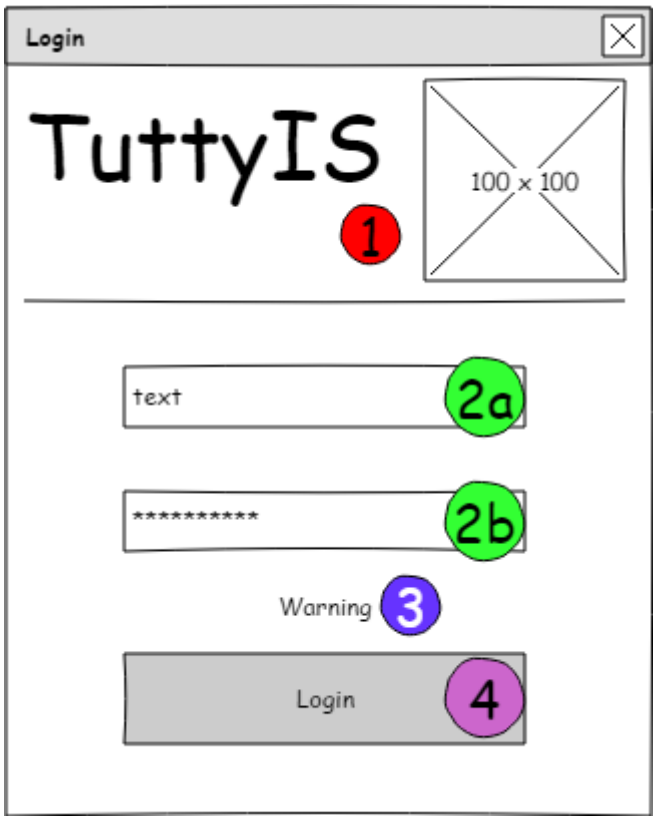
Michal Svoboda pracuje jako skladník prodejny. Zodpovídá za správné množství skladových položek na skladě. Zajišťuje příjem a výdej skladových zásob a provádí jejich inventarizaci.

Řešení: Správa naskladňování, vyskladňování a přesunů skladových zásob. Nástroj pro jejich inventarizaci.

4.2.6 Model přihlašovacího formuláře

1. **Use case**
    - Při přihlašování do systému uživatel očekává zadání:
        - Uživatelského jména
        - Hesla
    - Po zadání a potvrzení validních údajů pro přihlášení očekává:
        - načtení hlavní plochy programu
    - Po zadání špatných přihlašovacích údajů očekává:
        - upozornění na nesprávnost zadaných údajů
2. **Scénář**
    - Systém zobrazí přihlašovací formulář s aktivními prvky:
        - Systém od uživatele požaduje vyplnění údajů:
            - Textové pole – „Uživatelské jméno“
            - Textové pole – „Heslo“
        - Systém vyžaduje interakci s aktivním prvkem:
            - Tlačítko – „Login“
    - Po stisku tlačítka se systém spojí s databází, validuje zadané údaje a přechází do stavu:
        - Přihlášen – zobrazí hlavní plochu programu
            - Administrator – superuser
            - User – omezená práva
        - Chyba – systém vypíše hlášení o špatně zadaných údajích

3. Logický model



Obrázek 1 – Logický design přihlašovacího formuláře

1) **Informační pole**
2) **Textové pole**
   a) Uživatelské jméno
   b) Heslo
3) **Label** – chybové hlášení
4) **Tlačítko** – přihlásit

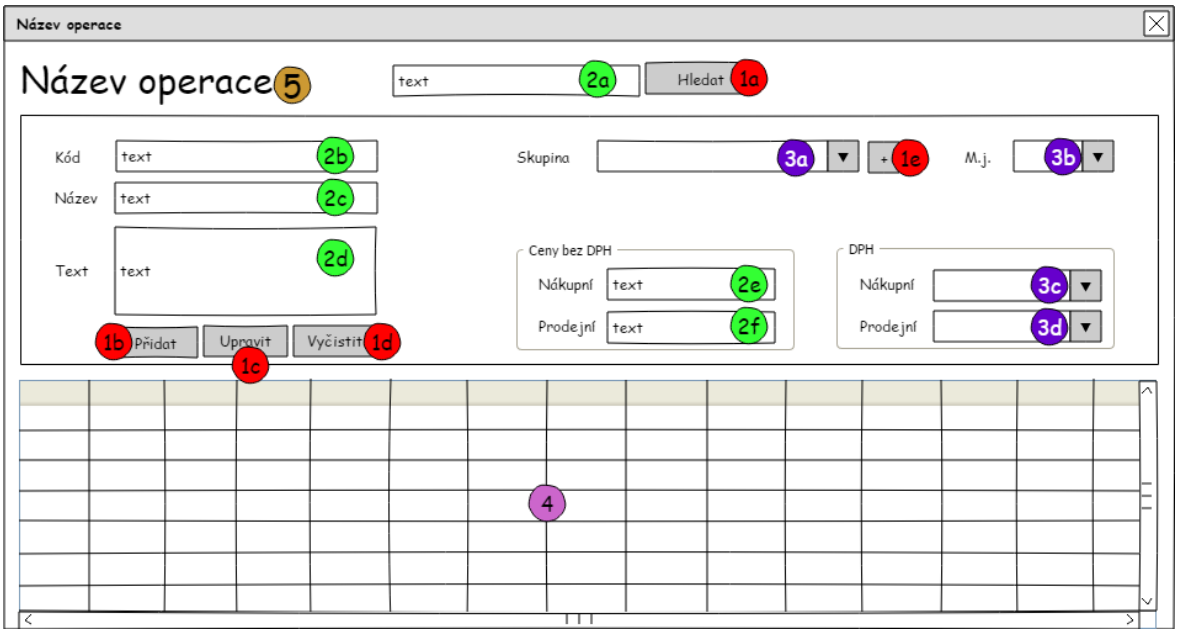
4.2.7 Model přidat novou položku

1. Use case

- Uživatel po otevření okna funkce očekává:
  - Výpis skladových položek
  - Možnost vyhledávat skladové položky
- Dále uživatel očekává manipulaci s položkami:
  - Vytvořit novou skladovou položku
  - Upravit vybranou skladovou položku
  - Mazat skladové položky

2. **Scénář**
    - Systém po načtení okna vypíše do gridu seznam již vytvořených položek a vyčkává na interakci uživatele
    - Pro vyhledávání položky v seznamu položek systém vyžaduje:
        - Vyplnění textového pole pro vyhledávání
        - Potvrzení stiskem tlačítka „Hledat“
    - Při tvorbě nové položky systém přechází do stavu:
        - Úspěšně
            - Systém vyžaduje korektní vyplnění všech povinných polí a potvrzení tlačítkem „Přidat“
                - Položka je úspěšně přidána
                - Seznam se aktualizuje
        - Neúspěšně
            - Uživatel je o neúspěšnosti akce informován chybovou hláškou
                - Položka není přidána do seznamu
    - Pro úpravy systém vyžaduje:
        - Načtení hodnot do příslušných polí
            - Dvojité označení stávající položky
        - Modifikaci stávající položky
            - Korektní vyplnění všech polí
        - Potvrdit provedené změny tlačítkem „Upravit“
    - Při mazání položky ze seznamu systém vyžaduje:
        - Označit položku k odstranění
        - Potvrdit operaci

3. Logický model



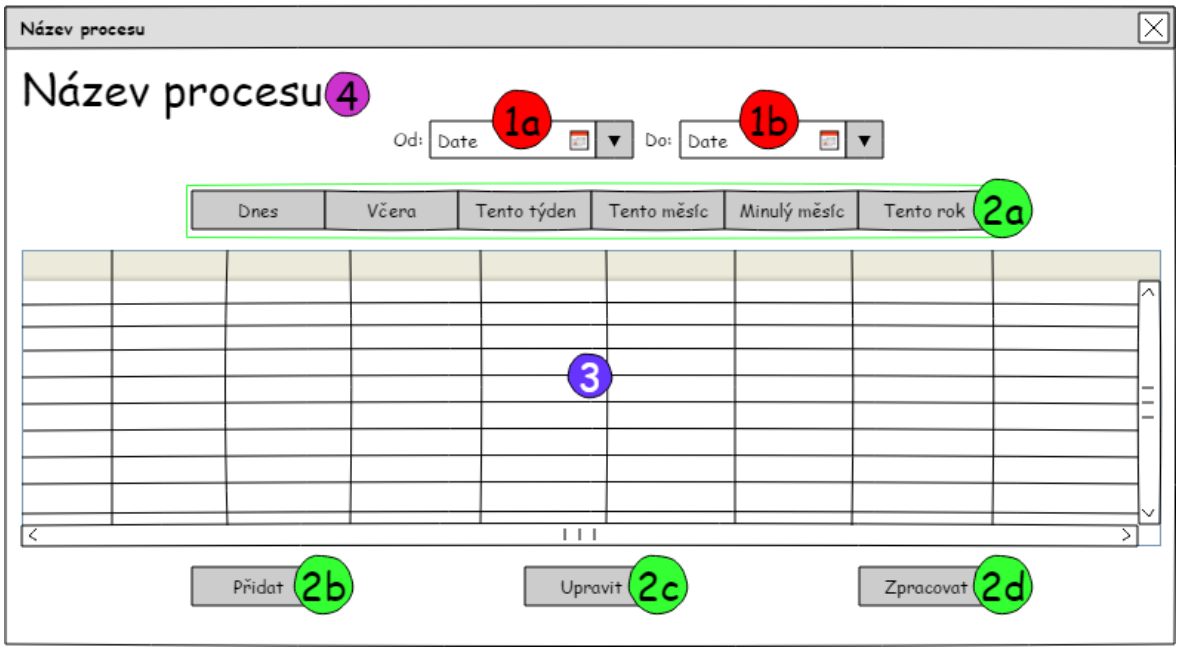
Obrázek 2 – Logický design přidat novou položku

1) **Tlačítko**
   a) Vyhledat skladovou položku
   b) Vytvořit novou položku
   c) Upravit stávající položku
   d) Vyčistit textová pole
   e) Otevřít seznam skupin
2) **Textové pole**
   a) Vyhledávač položek
   b) Kód položky
   c) Název položky
   d) Poznámka
   e) Nákupní cena
   f) Prodejní cena
3) **ComboBox**
   a) Seznam dostupných skupin
   b) Seznam měrných jednotek
   c) Seznam nákupních DPH
   d) Seznam prodejních DPH
4) **Grid** – Seznam skladových položek
5) **Label** – Informační pole

4.2.8 Model procesu

1. **Use case**
   - Uživatel v první řadě očekává:
     - Vypsání seznamu procesů na základě nastaveného časového rozptylu
   - Dále očekává možnost:
     - Vytvořit nový proces
     - Upravit vybraný proces
     - Smazat zvolený proces
     - Zpracovat konkrétní proces
     - Vytvářet reporty procesů
2. **Scénář**
   - Po načtení okna systém vykoná:
     - Nastavení časového rozptylu v rozmezí aktuálního dne
     - Výpis položek v závislosti na časovém rozptylu
   - Po stisku tlačítka „Přidat“ systém provede:
     - Nastavení řídící proměnné na hodnotu 0 (0 => nový záznam)
     - Otevření formuláře pro definici záznamu s defaultním nastavením
   - Pro stisknutí tlačítka „Upravit“ systém požaduje označit konkrétní záznam a provede:
     - Nastavení řídící proměnné na ID záznamu
     - Otevření formuláře pro modifikaci záznamu s hodnotami přiřazenými k záznamu
   - Pro zpracování záznamu je systémem vyžadováno označení konkrétního záznamu:
     - Je-li záznam nezpracován, přechází do stavu:
       - Zpracovat:
         - Přiřazené položky jsou zahrnuty v položkách na skladě
         - Přiřazené položky jsou zahrnuty do seznamu inventury
     - Je-li záznam již zpracován, přechází do stavu:
       - Zpracovat zpět:
         - Přiřazené položky nebudou zahrnuty v položkách na skladě
         - Přiřazené položky nebudou zařazeny do seznamu inventury

3. Logický model



Obrázek 3 – Logický design procesu

1) **DatePicker**
   a) Datum – „Od“
   b) Datum – „Do“
2) **Tlačítko**
   a) Množina tlačítek pro nastavení časového intervalu
   b) Vytvořit nový záznam procesu
   c) Upravit stávající záznam procesu
   d) Zpracovat označený proces
3) **Grid** – Seznam procesů
4) **Label** – Informační pole

4.3 Databáze

Aplikace pro práci s daty využívá *SQL Server 2012 Express LocalDB*. Pro správnou funkčnost programu je tak potřeba před spuštěním informačního systému mít nainstalovanou službu *LocalDB*, která je přikládána k programu v přílohách této práce. Jedná se o lokální databázi určenou primárně pro menší databáze a jednouživatelský přístup. K datům je tak možné přistupovat bez ohledu na připojení k internetu. V rámci vývoje byla zvolena jako ideální varianta pro svou snadnou manipulaci a přenositelnost. Díky kompatibilitě databázových serverů Microsoftu je možnost v budoucnu databázi

směřovat na plnohodnotný SQL Server. Momentálně celou databázi tvoří dva soubory s příponami „.mdf“ a „.ldf“. Datový MDF soubor obsahuje zpracovávaná data a strukturu celé databáze. Transakční LDF soubor slouží k uchování provedených operací během práce s databází. Jejich spojení s aplikací je realizováno přes připojovací řetězec (ConnectionString).

```
Data Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\TuttyDB.mdf;Integrated Security=True
```

Obrázek 4 – Propojení databázových souborů s aplikací pomocí ConnectionStringu

Celá databázová vrstva běží pod Entity Frameworkem. Pro přístup a manipulaci s daty nejsou tedy generovány klasické sql příkazy. K datům se přistupuje objektově v podobě syntaxe jazyka C#. K jednotlivým tabulkám je přistupováno jako ke třídám, které obsahují vlastnosti na základě atributů tabulky. Pro každou třídu je vytvořen konstruktor, přes který je možné se odkazovat na konkrétní vlastnosti. Celá databáze má pak vlastní *partial* třídu, která dědí *DbContext*, jenž byl automaticky vygenerován Entity Frameworkem.

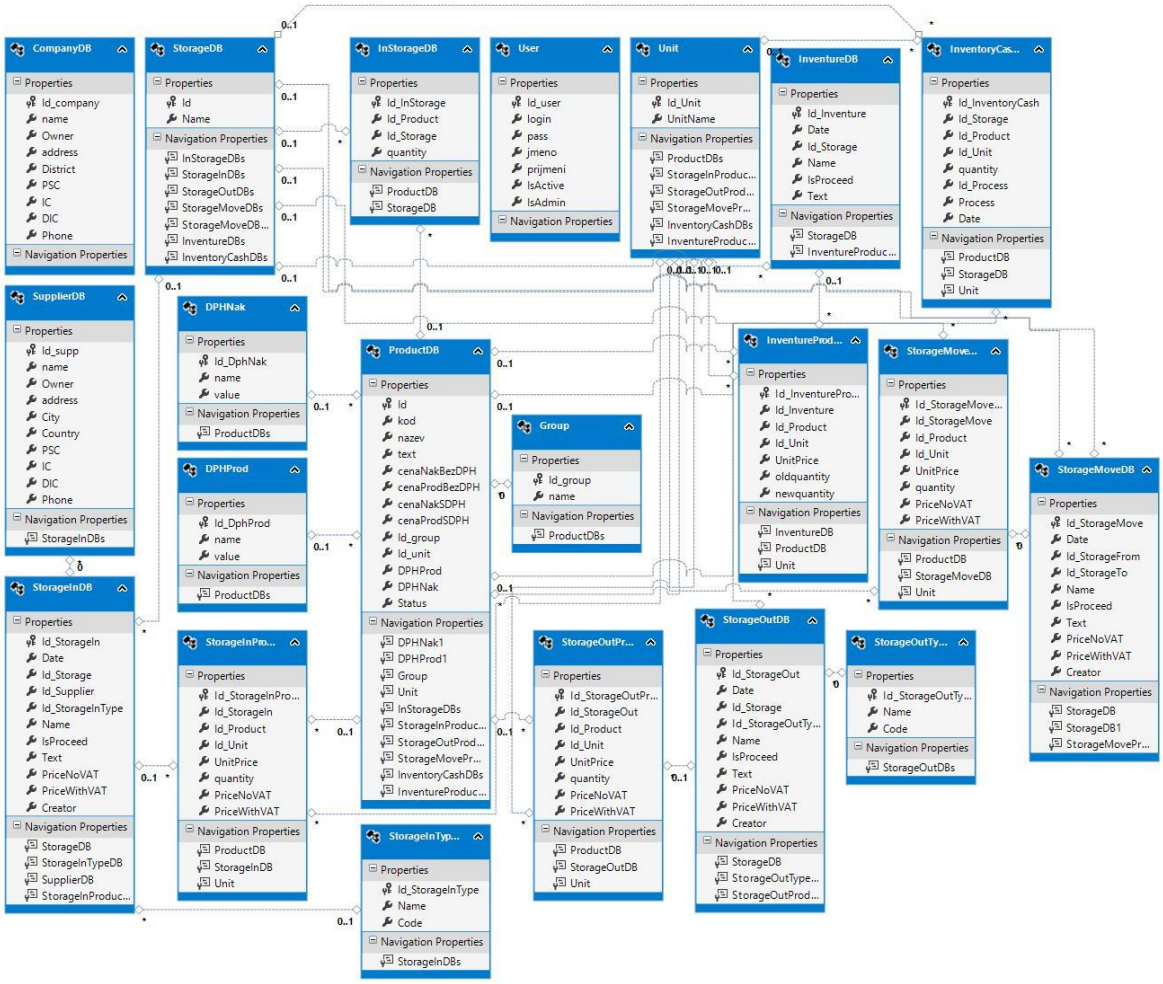
4.3.1 Databázová struktura

Kompletní databázová struktura je uložena v souboru MDF s názvem TuttyDB. Aktuálně je tvořena celkem 21 tabulkami, které jsou navzájem provázány cizími klíči. Entity Framework vyžaduje přítomnost primárních klíčů tabulky, jsou tak splněna entitní i referenční integritní omezení. Primární klíče jsou generovány automaticky po vytvoření nového řádku v databázi funkcí *„AUTOINCREMENT“*, resp. *„IDENTITY“* pro servery Microsoftu.

Pro reporty byly vytvořeny speciální procedury s omezujícími podmínkami, které odpovídají potřebám konkrétního reportu. Tyto procedury mají primárně zajistit seskupení položek stejného typu a eliminovat tak výpis duplicitních položek do více řádků.

4.3.2 Databázový model

Vzhledem ke zvolené metodě přístupu DatabaseFirst je jako první vytvořena samotná databázová struktura. Pomocí Entity Frameworku je z ní následně automaticky generován její databázový model přes který se při změně struktury databáze kompletně aktualizuje kontextová třída. Jedná se tak o velmi rychlou synchronizaci při změně databázové struktury přetransformovanou do syntaxe jazyka C#.



Obrázek 5 – Databázový model aplikace

4.3.3 DataSet

Výpis dat do aplikace reprezentuje datová sada *DataSet*, která je základně vázána na databázový model, z něhož následně vychází. Svou strukturou kopíruje vlastnosti databázové struktury a pomocí DataBindingu přenáší její datový obsah do prvků uživatelského rozhraní. Některé databázové struktury jsou v DataSetu částečně upraveny, v případě, má-li být vypsán obsah na základě konkrétní proměnné, jejíž hodnota je definována za běhu aplikace.

DataSet je v projektu využíván primárně pro přenos dat z databáze do gridu nebo pro výpis databázových procedur při sestavě reportů. Při každém použití DataBindingu musí být programově sestrojen propojovací adaptér a datový zdroj naplněný daty.

```
public void LoadGrid()
 {
    TuttyIS.UserDataSet userDataSet = ((TuttyIS.UserDataSet)(this.FindResource("userDataSet")));

    TuttyIS.UserDataSetTableAdapters.InventureDBTableAdapter userDataSetInventureDBTableAdapter =
    new TuttyIS.UserDataSetTableAdapters.InventureDBTableAdapter();

    userDataSetInventureDBTableAdapter.Fill(userDataSet.InventureDB,
   FromDatePicker.SelectedDate.Value.Date, ToDatePicker.SelectedDate.Value.Date);

    System.Windows.Data.CollectionViewSource inventureDBViewSource =
    ((System.Windows.Data.CollectionViewSource)(this.FindResource("inventureDBViewSource")));
 }
```

Obrázek 6 – Propojení a naplnění datové sady DataSet

4.4 Uživatelské rozhraní

Uživatel veškeré operace, včetně manipulace s daty v databázi, provádí skrze jednoduché okenní rozhraní. Logika oken je rozvržena tak, aby předcházela neočekávaným událostem způsobených možnou uživatelovou nepozorností při přepínání mezi jednotlivými otevřenými aktivními okny. Většinu funkcí aplikace je možné používat nezávisle na datovém připojení (tj. v offline režimu), některé další nadstandartní funkce pro svou správnou funkčnost ovšem potřebují připojení k internetu.

4.4.1 MainWindow

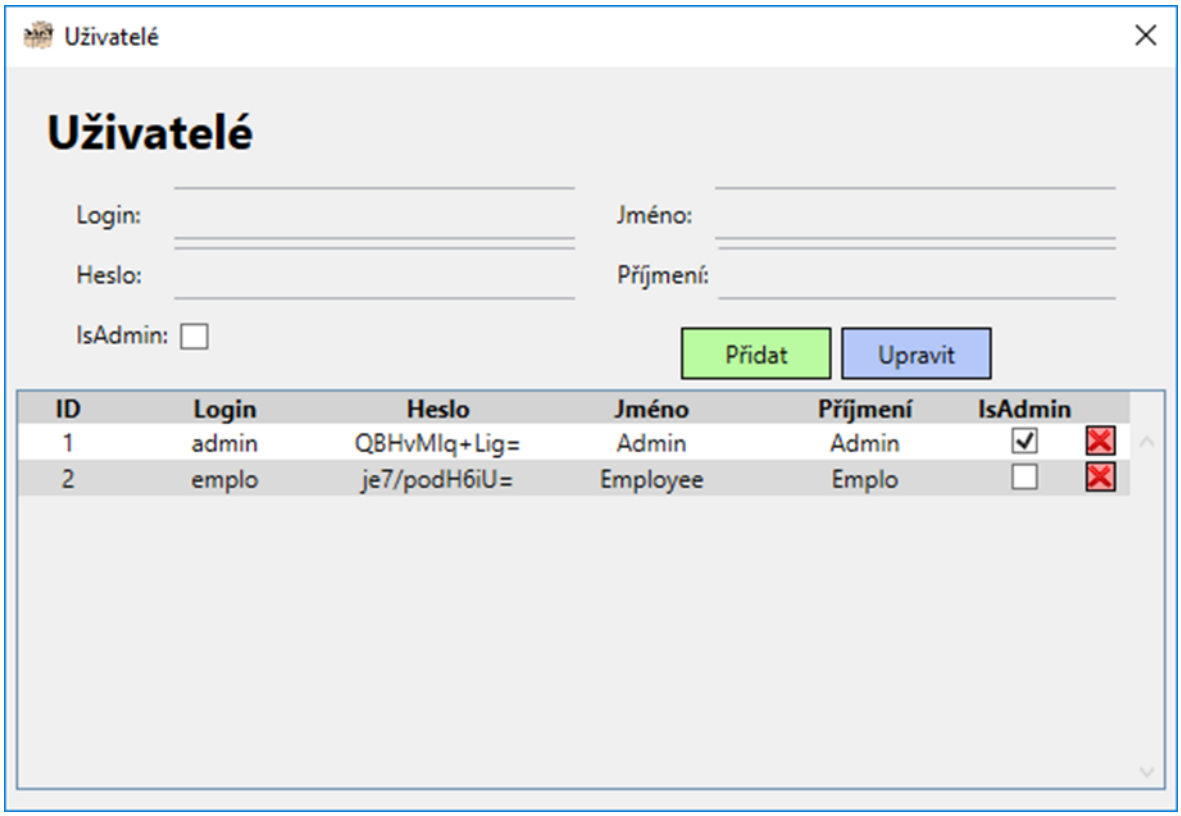
Jedná se o okno s nejvyšší prioritou, které je dostupné po celou dobu běhu aplikace, na jehož základě je možné otevírat další okna. Slouží jako jakási nástěnka, která obsahuje tabuli rychlých přehledů a přes kterou je možné přistupovat k ostatním funkcím. Na horní části je fixně umístěno menu, skrze které lze v zásadě přistupovat ke všem funkcím celé aplikace. Pro efektivnější práci s programem jsou vybrány nejčastěji používané funkce, ke kterým je možné přistupovat přes tlačítka s rychlou volbou situovaných do oblasti pod menu společně s informačními údaji o provozované firmě a tlačítkem pro odhlášení ze systému. Převládající obsah je tvořen nástěnkou s aktuálním přehledem o stavu skladových položek. Na spodní straně je vyhrazen pruh s informačním dosahem.

4.4.2 Přihlašovací formulář a uživatelské účty

Po spuštění aplikace je uživatel vyzván se do systému přihlásit platnými údaji. Údaje se skládají z existujícího uživatelského jména a příslušného validního hesla. Formulář obsahuje stavový řádek, který v případě špatně zadaných přihlašovacích údajů uživatele informuje výstražnou hláškou. V případě dodržení správnosti platných údajů je požadavek odeslán a zpracován.

Obrázek 7 – Přihlašovací formulář

Manipulovat s uživatelskými účty může pouze osoba s oprávněním na úrovni administrátora. Běžný uživatel nemá k účtům přístup. Jejich změna se provádí po přihlášení oprávněného uživatele do systému v nastavení účtů. Seznam uživatelů je dostupný přes rozhraní DataGridu v podobě tabulky. Účty je možné následně přidávat, upravovat a mazat. Změnu je možné vyvolat dvojitým poklepáním na řádek s příslušným účtem. V rámci bezpečnosti a ochraně dat jsou hesla plošně šifrována rovnou v databázi metodou MD5.

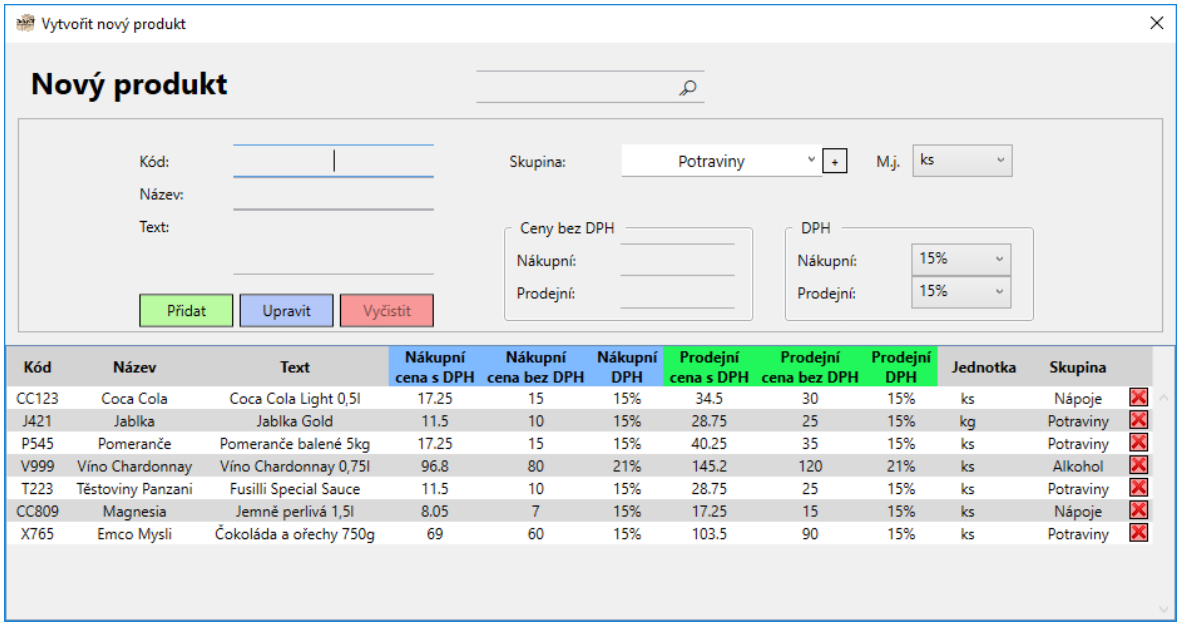


Obrázek 8 – Nastavení uživatelských účtů

4.4.3 Vytváření nových položek

Aby bylo možné s položkami jakkoliv manipulovat (tzn. naskladnit, vyskladnit, přesouvat a provádět jejich inventarizaci), je nejprve nutné položku nadefinovat. To zajišťuje speciální rozhraní sdružující všechny dostupné položky, včetně položek již zahrnutých v procesu. Opět je s nimi možné provádět trojici klasických operací (přidat, upravit, smazat). V případě, že je položka obsažena v dalších databázových tabulkách v podobě cizího klíče, není možné takovou položku smazat. V takovém případě je uživatel na situaci upozorněn a poučen.

Po vložení nákupní a prodejní ceny dochází v závislosti na zvoleném DPH k přepočtu jejich celkových cen včetně sazby DPH. Pro lepší orientaci jsou oba typy cen následně barevně rozlišeny. Zelené sloupce představují prodejní ceny, modré pak ceny nákupní, včetně příslušných sazeb DPH.



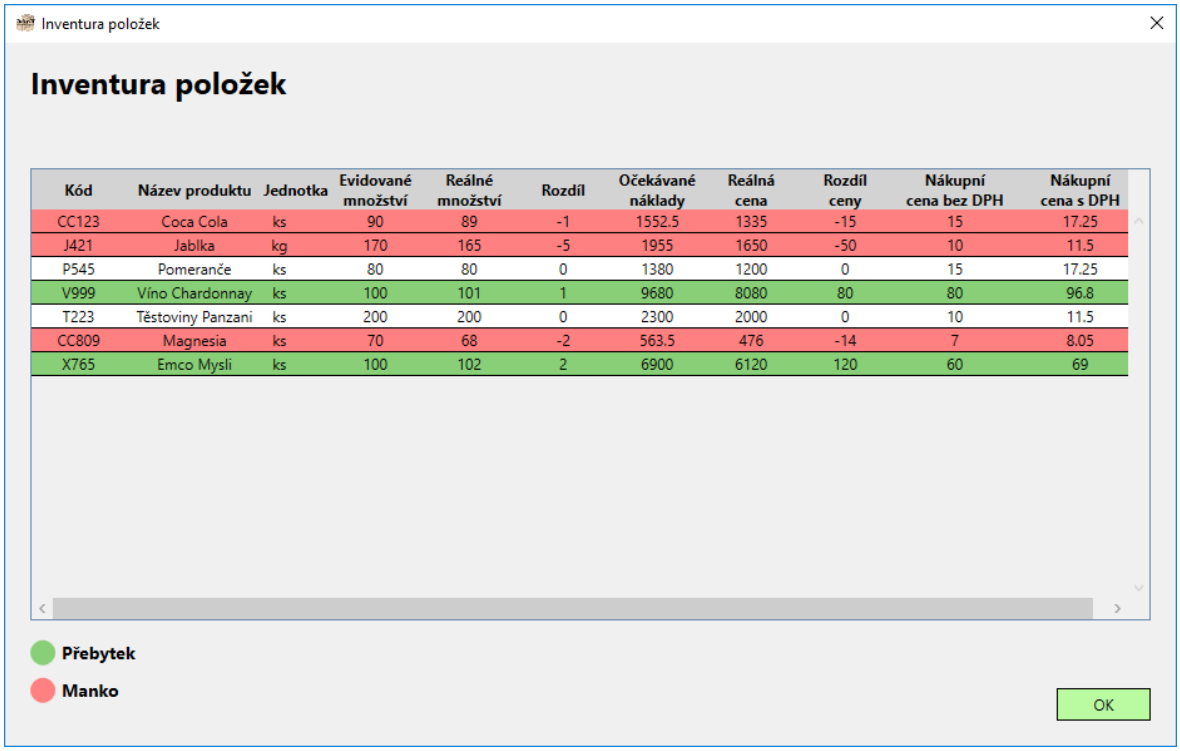
Kód	Název	Text	Nákupní cena s DPH	Nákupní cena bez DPH	Nákupní DPH	Prodejní cena s DPH	Prodejní cena bez DPH	Prodejní DPH	Jednotka	Skupina
CC123	Coca Cola	Coca Cola Light 0,5l	17.25	15	15%	34.5	30	15%	ks	Nápoje
J421	Jablka	Jablka Gold	11.5	10	15%	28.75	25	15%	kg	Potraviny
P545	Pomeranče	Pomeranče balené 5kg	17.25	15	15%	40.25	35	15%	ks	Potraviny
V999	Víno Chardonnay	Víno Chardonnay 0,75l	96.8	80	21%	145.2	120	21%	ks	Alkohol
T223	Těstoviny Panzani	Fusilli Special Sauce	11.5	10	15%	28.75	25	15%	ks	Potraviny
CC809	Magnesia	Jemně perlivá 1,5l	8.05	7	15%	17.25	15	15%	ks	Nápoje
X765	Emco Mysli	Čokoláda a ořechy 750g	69	60	15%	103.5	90	15%	ks	Potraviny

Obrázek 9 – Formulář pro založení nové skladové položky

4.4.4 Inventury

Inventura je úkon správy majetku platný zákonem, při němž dochází k porovnávání skutečných stavů zásob se zásobami evidovaných v účetnictví. Výstupy inventur mají uživatele aplikace informovat o těchto stavech u jednotlivých skladových položek, se kterými bylo v minulosti manipulováno. Ostatní položky nejsou do inventury zahrnuty. V případě jejich rozdílů se jedná o manka či přebytky. Inventarizace by měla být vykonána v období uvedeném zákonem, v případě potřeby systém umožňuje vytvářet inventury i v průběhu měsíce.

Po změně skutečných stavů aplikace provádí okamžitý přepočet rozdílů v reálném čase. Je tak možné ihned sledovat konkrétní manka či přebytky u skladových položek, včetně rozdílů jejich cen, ke konkrétnímu datu. Manka a přebytky jsou v systému rozlišeny barevným označením. Po dokončení inventury je možnost pořídit její tisknutelný report.



Obrázek 10 - Inventura

4.4.5 Procesy

Aplikace obsahuje sadu tří základních firemních procesů, které se týkají zásob na skladě. Jedná se o procesy naskladnění, vyskladnění a přesuny položek mezi jednotlivými sklady podniku. Z pohledu elektronické evidence se jedná o ekvivalentní druh činnosti s rozdílným výsledkem a je proto k těmto procesům takto přistupováno. Jejich grafický návrh je z většiny analogický, stejně jako celý soubor úkolů potřebných vykonat pro jejich vytvoření.

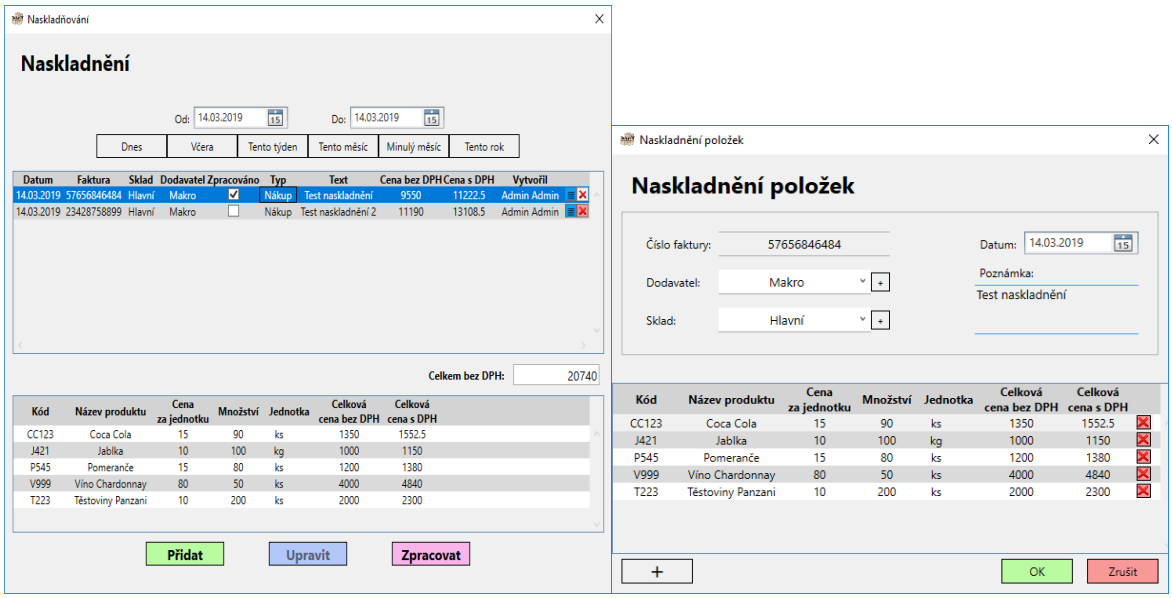
Tvorba procesu by měla být pro uživatele co nejefektivnější a co nejjednodušší. K jeho vytvoření je tak zapotřebí pouhých tří aplikačních oken.

První z nich slouží k celkovému přehledu již vytvořených procesů a možnosti jejich úpravy, případně k vytvoření zcela nového procesu. V tomto okně lze také jednotlivé procesy mazat nebo zpracovat, případně sestavit tisknutelný report. Dojde-li ke zpracování některého z již vytvořených procesů, jeho obsah bude v závislosti na typu procesu evidován na zvoleném skladě a zároveň bude zařazen mezi položky pro přepočet inventur. V nastavení systému je možné zvolit omezení, které uživateli znemožní zpracovat doklad, který obsahuje větší množství položek než je aktuálně na skladě a jejich množství by tak mělo nabývat záporných hodnot. V takovém případě bude uživatel informován seznamem s položkami, které nesplňují podmínky zpracování Toto omezení lze v nastavení kdykoli

zrušit. Úpravy a mazání jednotlivých procesů je možné provádět pouze za předpokladu, že daný proces ještě není zpracován.

Druhé okno slouží k definici právě vytvářeného nebo upravovaného procesního dokladu. Jedná se o základní informace hlavičky dokladu, která definuje jeho název, datum vytvoření a další doplňující informace o dokladu pro jeho lepší rozpoznání. V neposlední řadě obsahuje seznam všech přiřazených položek s možností změnit jejich aktuální množství.

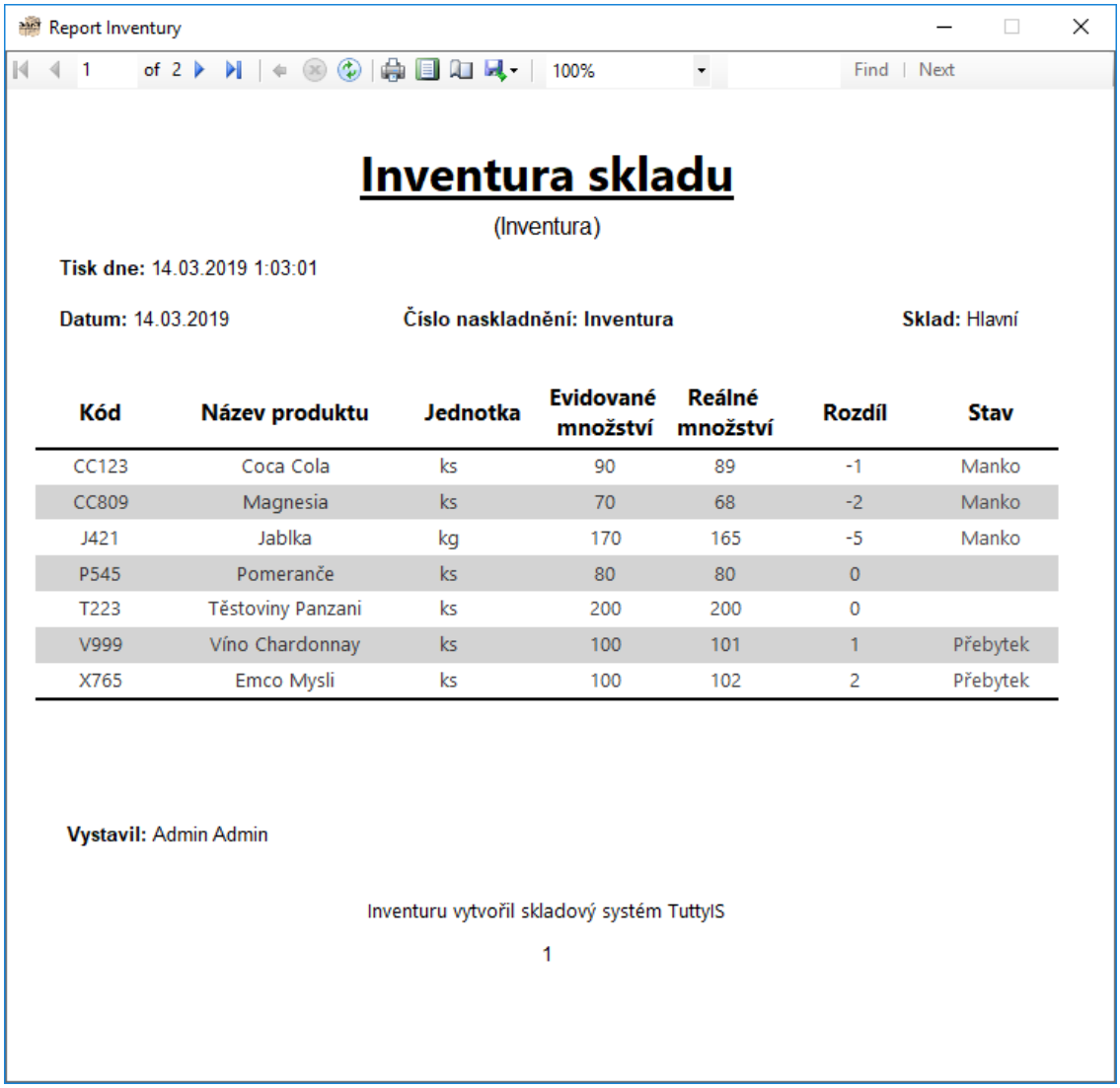
Poslední z oken je tvořeno obyčejným seznamem již vytvořených položek, které je možné zařadit do procesu. Defaultně je jejich množství nastaveno na jednu jednotku dané veličiny vztahující se k vybrané položce.



Obrázek 11 – Formuláře procesu naskladnění

4.4.6 Reporty

Reporty jsou v systému tvořeny za účelem tisknutelného účetního dokladu. Visual studio pro starší technologii WinForms umožňuje designovat a vytvářet jednoduché reporty již v samotném základu. Report tak byl vytvořen na této technologii a integrován do novější technologie WPF. Pro data obsažené v reportu byly sestaveny speciální databázové procedury, které seskupují položky a sumarizují jejich množství.



Obrázek 12 – Ukázka reportu inventury

5 Výsledky a diskuse

5.1 Výsledky

Výsledek aplikace vychází z cílů stanovených na začátku vývoje. Byla tedy vyvinuta funkční verze vlastní aplikaci evidující skladové hospodářství, která demonstruje použití Entity Frameworku společně s moderními technikami programování v programovacím jazyce C#. Aplikace byla vyvíjena pro platformu běžící pod systémy Microsoft Windows a mohou jí tak využívat výhradně uživatelé těchto operačních systémů.

5.1.1 Testování

Testování aplikace bylo rozděleno do několika různých a na sobě nezávislých fází. Do jednotlivých fází testování byly zahrnuty úkoly, které zkoumaly uživatelskou přívětivost a použitelnost jednotlivých částí aplikace na základě připravených testovacích scénářů, kompatibilitu různých verzí operačního systému a částečnou hardwarovou náročnost z hlediska optimalizace.

Vyvíjená aplikace byla testována na všech aktuálně podporovaných verzích operačního systému Windows od verze Windows 7 a výše. Starší verze systému, vzhledem k jejich již ukončené podpoře, nebyli do testování zařazeny. Jednotlivé verze testovaných operačních systémů vykazovali analogické chování.

Aplikace je v současné době poskytována ve své první beta verzi a byla tak v rámci testováni zpřístupněna vybrané skupině potencionálních uživatelů, jejichž úkolem bylo založení nové skladové položky, evidence jejich pohybů mezi jednotlivými sklady a následná inventarizace všech dostupných skladů. Testování proběhlo nadmíru úspěšně a většina dotázaných dokázala splnit všechny úkoly bez potřeby nápovědy. Největší chybovost nastávala v momentě, kdy měl uživatel provést inventury skladů. Chyba byla z většiny případů způsobena nezpracováním vytvořených procesů a položky tak neodpovídaly reálným hodnotám.

Při testování optimalizace bylo možné sledovat automatickou správu paměti, kterou zajišťuje GC. Aplikace tak běží relativně svižně a v rámci tolerancí využívá a pracuje s operační pamětí počítače.

5.2 Diskuze

Tím, že se aktuálně jedná o první oficiálně dostupnou verzi aplikace, je možné během jejího používání narazit na několik menších a zatím neopravených chyb, které by však měly být opraveny v její nejbližší aktualizaci. Aplikace se drží moderních technik programování, které umožňují rozmanité možnosti interakce s uživatelem a snadnou správu kódu. To by mělo usnadnit možnosti pro budoucí vývoj a rozšíření aplikace o nové funkce.

Budoucí vývoj by měl přinést řadu novinek a vylepšení. Jedním z nich by mohlo být přenesení lokální databáze na plnohodnotný sql server. Aplikace by tak získala široké možnosti rozšíření. Aplikace, která byla navržena pro víceuživatelský přístup, by tak umožňovala práci se systémem více uživatelům současně a zároveň by mohl být implementován modul pro centrální správu skladů. V této fázi vývoje by také mohlo být vytvořeno webové rozhraní, které by umožňovalo zobrazit rychlý přehled a případně jednoduchou administraci nad skladovým systémem z jakéhokoliv zařízení přímo v internetovém prohlížeči.

Lze předpokládat, že by portfolio skladového systému bylo v další fázi vývoje rozšířeno o nezávislou aplikaci pokladního systému, který by ovšem byl napojen na současnou aplikaci skladového systému. Vznikla by tak provázanost mezi oběma systémy. Pokladní systém by čerpal dat uložených ve skladech a vytvářel reporty o prodaných položkách a celkových tržbách. Zároveň by prodané položky automaticky odečítal z vybraného skladu.

6 Závěr

Bakalářská práce má čtenáře obeznámit se základy programování v jazyce C# a vývojovým prostředím Visual studio 2012 společně s používanými moderními technikami programování, které zahrnují objektově orientované programování ve spojení s technologií WPF nebo využití frameworku pro objektové mapování databází.

Cílem bakalářské práce bylo popsat metody používané při objektově relačním mapování lokální databáze pomocí Enity Frameworku v jazyce C#. Cíle bylo dosaženo a výsledkem je funkční verze elektronické evidence skladového hospodářství s možnostmi rozsáhlého rozšíření.

Nejdříve musely být stanoveny konkrétní požadavky na aplikace. Dále mohl být navržen logický model aplikace pomocí testovacích scénářů v rámci UI testování. Na základě návrhu uživatelského rozhraní byla navržena databázová struktura lokální databáze. Realizace byla postavena na moderních programovacích technologiích jako Enity Framework a Windows Presentation Foundation. Závěrečná část zahrnovala testování aplikace na potencionálních uživatelích vyvíjeného informačního systému.

Teoretická část práce podrobně vysvětluje principy objektově orientovaného programování s důrazem na znovupoužitelnost kódu. Dále popisuje moderní a v současné době velice populární a hojně využívanou technologii WPF zabývající se grafickou stránkou návrhu aplikace ve spojení s programovacím jazykem C#, kde se pokouší vyzdvihnout její přednosti a rozebrat její nedostatky. V neposlední části se snaží zaměřit na využití Entity Frameworku při vývoji desktopových aplikací a přiblížit čtenáři výhody které s sebou přináší jak z hlediska krátkodobého, tak dlouhodobého. V závěru teoretické části je čtenář stručně obeznámen s vývojovým prostředím Visual studio od společnosti Microsoft.

Praktická část následně čerpá a vychází z provázanosti znalostí popsaných v teoretické části práce, které byli aplikovány na aplikaci skladového hospodářství během jejího vývoje. Bylo tak dosaženo cílů bakalářské práce stanovených před vývojem aplikace a praktická část vlastní práce čtenáře provádí od prvotních požadavků na aplikaci po návrh databázové struktury a uživatelského rozhraní v podobě UI specifikace až k její konečné realizaci. Závěrem je tedy shrnut aktuální stav aplikace a jsou navržena možná rozšíření a její budoucí vývoj.

7 Seznam použitých zdrojů

Albahari, Joseph a Albahari, Ben. 2012. *C# 5.0 Pocket Reference.* Sebastopol : O'Reilly Media, Inc., 2012. 978-1-449-32017-1.

Clark, Dan. 2013. *Beginning C# Object-Oriented Programming.* New York City : Apress, 2013. 978-1-4302-4936-8.

Čápka, David. 2012. Lekce 1 - Úvod do C# a .NET frameworku. *ITnetwork.cz.* [Online] ITnetwork, 10. Květen 2012. [Citace: 20. Leden 2018.] https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework.

Dařena, František. 2004. Objektově orientované programování. *akela.mendelu.cz.* [Online] 2004. [Citace: 13. Leden 2019.] https://akela.mendelu.cz/~darena/Perl/objekty.html.

Driscoll, Brian, a další. 2013. *Entity Framework 6 Recipes.* New York City : Apress, 2013. 978-1-4302-5789-9.

Lerman, Julia. 2010. *Programming Entity Framework.* Sebastopol : O'Reilly Media, Inc., 2010. 978-0-596-80726-9.

MacDonald, Matthew. 2012. *Pro WPF 4.5 in C#.* New York City : Apress, 2012. 978-1-4302-4366-3.

Martinů, Jiří. 2017. *Objektové programování.* [Studijní opora] Olomouc : Moravská vysoká škola Olomouc, 2017.

Nakov, Svetlin a Kolev, Veselin. 2013. *Fundamentals of Computer Programming with C#.* Bulharsko : Sofia, 2013. 978-954-400-773-7.

Sharp, John. 2012. *Microsoft Visual C# 2012 Step by Step.* California : O'Reilly Media, Inc., 2012. 978-0-7356-6801-0.

Wang, Wallace. 2008. *Beginning Programming All-In-One Desk Reference For Dummies.* Hoboken : Wiley Publishing, Inc, 2008. 978-0-470-10854-3.

Yosifovich, Pavel. 2012. *Windows Presentation Foundation 4.5 Cookbook.* Birmingham : Packt Publishing Ltd., 2012. 978-1-84968-622-8.

8 Přílohy

Příloha 1 Přiložené CD, které obsahuje:

- Zdrojové kódy aplikace
- Spustitelnou aplikaci
- Soubor lokální databáze
- SQL Server 2012 Express LocalDB
- Bakalářskou práci ve formátu PDF