Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Informační systém pro půjčovnu oděvů

Mariya Vizinska

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Mariya Vizinska

Informatika

Název práce

Informační systém pro půjčovnu oděvů

Název anglicky

Information system for clothing rental

Cíle práce

Cílem práce je navrhnout, implementovat a otestovat informační systém pro půjčovnu oděvů. Výsledná aplikace bude běžet na platformě MS Windows a MySql a bude naprogramována v prostředí jazyka C#. Součástí práce bude také uživatelská příručka k systému. Předpokládáme, že vedle této aplikace již existuje nabídka oděvů na internetu a dále předpokládáme, že objednávky budou zákazníci realizovat osobní návštěvou s možností následného upřesnění po telefonu, ale ne ve formě e-shopu.

Metodika

Nejprve bude provedena analýza požadavků a rozpoznány hlavní procesy. Celá dokumentace bude ve standardech UML a BPMN. Dále bude následovat softwarový projekt a jeho softwarová dokumentace. Výsledná aplikace bude otestována na malém vzorku reálných dat.

Doporučený rozsah práce

40-60 stran

Klíčová slova

Clothing rental, Information system, C#

Doporučené zdroje informací

NAKOV, Svetlin et col. Fundamentals of computer programming with C#: the Bulgarian C# Programming Book. Sofia, 2013. ISBN 978-954-400-773-7

TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy: nástroje ke zvyšování kvality informačních systémů. Praha: Grada, 2008. Management v informační společnosti. ISBN 978-80-247-2728-8

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 7. 3. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 7. 3. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Informační systém pro půjčovnu oděvu" jsem vypracovala 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 autorka uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 15.3.2019 _____

Poděkování

Ráda bych touto cestou poděkovala panu doc. Ing. Vojtěchu Merunkovi, Ph.D. za odborné vedení, věcné připomínky, ochotu a vstřícnost při konzultacích a vypracování bakalářské práce. Mé poděkování také patří panu Lubošovi Kosovi za cenné rady, které mi pomohly tuto práci zkompletovat.

Informační systém pro půjčovnu oděvů

Abstrakt

V této práci se zabývám analýzou, návrhem, implementací a testováním informačního systému pro půjčovnu. Na základě znalostí získaných ve škole a zkušeností z práce, a také pomocí tištěných a internetových zdrojů, je vytvořen systém v souladu s nejlepšími praktikami softwarového inženýrství. Bakalářská práce slouží jako technická a uživatelská dokumentace k vytvořenému podnikovému informačnímu systému.

Práce se skládá ze dvou části: teoretické a vlastní práce. Součástí práce je vytvořený informační systém.

V teoretické části jsou vymezeny základní pojmy související s vývojem aplikace, popsané struktura a fáze vývoje informačního systému a detailně popsaná fáze analýzy požadavků a navrhování.

Po tom, co je systém navržen, začíná samotná implementace systému v jazyce C# s využitím databáze SQL. Proces implementace a testování je popsán v praktické části práce, součástí které je také uživatelská příručka.

Klíčová slova: Půjčovna oděvu, informační systém, C#

Information system for clothing rental

Abstract

In this work I describe the analysis, design, implementation and testing of the information system for the clothing rental. Based on the knowledge gained in the university and work experience, as well as printed and internet resources, a system is created in accordance with the best practices of software engineering. The bachelor thesis serves as a technical and user documentation for the created business information system.

The work consists of two parts: theoretical and own work. Created information system is attached to the bachelor thesis.

The theoretical part defines the basic concepts related to the development of the application, describes the structure and phases of development of the information system, and describes in detail phases of analysis and design.

After the system is designed, the implementation of the system in C # language with the use of the SQL database begins. Implementation and testing processes are described in the practical part.

Keywords: Closing rental, information systém, C#

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Kořeny nápadu navrhnout a naprogramovat informační systém sahají do dob, kdy jsem si vybírala šaty na maturitní ples. Tehdy jsem zjistila, jak těžké je v Praze si rezervovat zapůjčení hezkých šatů dříve, než na půl roku dopředu. Už tehdy mě napadlo otevřít si vlastní půjčovnu oděvů jakožto podnik, který by neměl vysokou konkurenci a zároveň by byl relativně bezpečnou investicí. Kdyby se tato myšlenka realizovala a takovýto podnik se v brzké době založil, neměl by šanci být úspěšný bez podpory informačních technologií. Proto se v této práci budu zabývat analýzou, návrhem a tvorbou podnikového informačního systému (PIS), pod názvem RentApp, který se bude užívat ve firmě k podpoře a zlepšování funkcí jejich podnikových procesů. Aplikace bude navržena pro vnitřní potřeby podniku a využívána pouze jeho zaměstnanci.

Na tvorbě PIS se obvykle podílí jak znalci světa informačních technologií, tak vedení firmy, které by mělo přesně stanovit k čemu se bude aplikace využívat. Proto je důležité porozumět dané formě podnikání a ujasnit si, čím se bude daný podnik zabývat a jaké procesy v něm budou probíhat.

V první řádě bude nahlédnuto na informační systém obecně a budou popsané nejlepší praktiky jeho vývoje. Dále v rámci analýzy požadavků, kromě vymezení procesů, budou také popsané entity, případy užití a také vyřešena přístupová práva do systému. Ve druhé fázi, kterou je návrh, je sestaven datový model, a v praktické části práce je zachycena nejdelší fáze vývoje – implementace, která začíná připojováním k databázi a končí testováním.

2. Cíl práce a metodika

Cílem práce je navrhnout, implementovat a otestovat informační systém pro půjčovnu oděvů. Výsledná aplikace bude běžet na platformě MS Windows a MySql a bude naprogramována v prostředí jazyka C#. Součástí práce bude také uživatelská příručka k systému. Předpokládáme, že vedle této aplikace již existuje nabídka oděvů na internetu a dále předpokládáme, že objednávky budou zákazníci realizovat osobní návštěvou s možností následného upřesnění po telefonu, ale ne ve formě e-shopu.

Nejprve bude provedena analýza požadavků a rozpoznány hlavní procesy. Celá dokumentace bude ve standardech UML a BPMN. Dále bude následovat softwarový projekt a jeho softwarová dokumentace. Výsledná aplikace bude otestována na malém vzorku reálných dat.

3. Teoretická východiska

3.1 Úvod do terminologie

MS Windows

Microsoft Windows je operační systém navržený a vyráběný společností Microsoft Corporation. Podobně jako u ostatních operačních systémů Windows dělá počítačový systém uživatelsky přívětivý tím, že poskytuje grafické rozhraní a organizuje informace tak, aby byly snadno přístupné.

Operační systém používá ikony a nástroje, které zjednodušují složité operace prováděné počítačem. Odhady naznačují, že 90 % osobních počítačů používá operační systém Windows. Microsoft představil operační systém v roce 1985 a nadále byl široce používán i přes konkurenci operačního systému Apple Macintosh. (BusinessDictionary.com [online]. © 2019)

C#

C# je elegantní a typově bezpečný objektově orientovaný jazyk, který umožňuje vývojářům vytvářet řadu bezpečných a robustních aplikací, které běží na rozhraní .NET Framework. C# je možné použít k vytvoření klientských aplikací Windows, webových služeb XML, distribuovaných komponent, aplikací klient-server, databázových aplikací a mnoha dalším. Visual C# poskytuje pokročilý editor kódu, praktické návrháře uživatelského rozhraní, integrovaný nástroj pro ladění a mnoho dalších nástrojů, které usnadňují vývoj aplikací založených na jazyce C# a rozhraní .NET Framework.

Jako objektově orientovaný jazyk C# podporuje koncepce enkapsulace, dědičnosti a polymorfismu. Všechny proměnné a metody, včetně hlavní metody Main, vstupního bodu aplikace, jsou zapouzdřeny do třídních definic. Třída může dědit přímo z jedné nadřazené třídy, ale může implementovat libovolný počet rozhraní. Metody, které přepisují virtuální metody v nadřazené třídě, vyžadují přepínání klíčového slova jako způsob, jak zabránit náhodnému novému určení. (Introduction to the C# Language and the .NET Framework | Microsoft Docs. [online]. © 2015.)

Windows Forms

Windows Forms je inteligentní klientská technologie pro .NET Framework, soubor spravovaných knihoven, které zjednodušují běžné aplikační úlohy, jako je čtení a zápis do souborového systému. Při použití vývojového prostředí, jako je Visual Studio, umožňuje vytvářet aplikace Windows Forms smart-klient, které zobrazují informace, vyžadují vstup uživatelů a komunikují se vzdálenými počítači v síti.

Formulář Windows Forms je vizuální povrch, na kterém se zobrazuje informace uživateli. Obvykle se vytváří aplikace Windows Forms přidáním ovládacích prvků do formulářů a vytvářením reakcí na akce uživatele, například kliknutí myší nebo stisknutím kláves. Ovládací prvek je diskrétní uživatelské rozhraní (UI), které zobrazuje nebo přijímá data.

Formuláře Windows forms obsahují řadu ovládacích prvků, které je možné přidat do formulářů: ovládací prvky zobrazující textové pole, tlačítka, rozevírací políčka, přepínače, a dokonce i webové stránky. Windows Forms také podporuje vytváření vlastních ovládacích prvků pomocí třídy UserControl. (Windows Forms Overview | Microsoft Docs. [online]. © 2017.)

UML

UML, zkratka pro Unified Modeling Language, je standardizovaný modelovací jazyk skládající se z integrovaného souboru diagramů vyvinutých pro vývojáře softwaru pro specifikaci, vizualizaci, konstrukci a dokumentaci artefaktů softwarových systémů, stejně jako pro obchodní modelování a jiné než softwarové systémy. UML představuje soubor nejlepších inženýrských postupů, které se osvědčily při modelování velkých a složitých systémů. UML je velmi důležitou součástí vyvíjení objektově orientovaného softwaru a procesu vývoje softwaru. UML používá většinou grafické notace k vyjádření návrhu softwarových projektů. Díky UML je jednodušší pro projektové týmy komunikovat, zkoumat potenciální návrhy a ověřovat architektonický návrh softwaru.

Cílem UML je poskytnout standardní notaci, která může být použita všemi objektivně orientovanými metodami, a vybrat a integrovat nejlepší prvky zápisů prekurzorů. UML byla navržena pro širokou škálu aplikací. Proto poskytuje konstrukce pro širokou škálu systémů a činností (např. Distribuované systémy, analýzu, návrh systému a nasazení). (What is Unified Modeling Language (UML)?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration [online]. © [cit. 11.03.2019].)

BPMN

Business Process Modeling Notation (BPMN) je metoda vývojového diagramu, která od začátku do konce modeluje kroky plánovaného podnikového procesu. Je klíčem pro řízení podnikových procesů, vizuálně popisuje detailní sled podnikových činností a informačních toků potřebných k dokončení procesu.

Účelem je navrhnout způsoby, jak zvýšit efektivitu, zohlednit nové okolnosti nebo získat konkurenční výhodu. Metoda prošla v posledních několika letech standardizačním tlakem a nyní je často nazývána trochu jiným názvem: Business Process Model and Notation, který stále používá zkratku BPMN. Liší se od Unified Modeling Language (UML) používaného při návrhu softwaru. (What is Business Process Modeling Notation | Lucidchart. Online Diagram Software & Visual Solution | Lucidchart [online]. © 2019.)

SQL

SQL (vyslovuje se jako "ess-que-el") je zkratka pro Structure Query Language. SQL se používá pro komunikaci s databází. Podle ANSI (American National Standards Institute) je standardním jazykem pro systémy pro správu relačních databází. Příkazy SQL se používají k provádění úloh, jako je aktualizace dat v databázi nebo načítání dat z databáze. Některé běžné systémy pro správu relačních databází, které používají SQL, jsou: Oracle, Sybase, Microsoft SQL Server, Access, Ingres atd. Standardní příkazy SQL, jako například "Vybrat", "Vložit", "Aktualizovat", "Odstranit", "Vytvořit" a "Zahodit", lze dosáhnout téměř vše, co je třeba udělat s databází. (SQLCourse - Lesson 1: What is SQL?. SQLCourse - Interactive Online SQL Training for Beginners [online]. © [cit. 11.03.2019].)

Existuje mnoho databází, které podporují používání SQL pro přístup k jejich datům, mezi které patří MySQL a PostgreSQL. Jinými slovy, MySQL je jen značkou jednoho z mnoha databázových softwarů. Totéž platí pro PostgreSQL. Tyto dvě databáze jsou velmi populární mezi programy, které běží na webových stránkách (pravděpodobně proto, že jsou zdarma). (thesitewizard.com: Website design, promotion, CGI, PHP, JavaScript scripting, and revenue earning. [online]. © 2010.)

3.2 Informační systém obecně

3.2.1 Definice

Slovo systém se používá v různých souvislostech a jeho význam závisí na historickém vývoji poznatku. Je blízký pojmům celistvost, organizace, organizmus, struktura. V řečtině a latině znamená kombinovat či uspořádat. Později se objevila myšlenka o řadu a uspořádanosti prvků nebo částí systému. Představa o struktuře vznikla již v antickém myšleni a uplatnila se zejména v tehdejších poznatcích o stavbě živého organizmu. Otcem teorie systémů je vídeňský profesor, později žijící v Kanadě – Ludwig von Bartalanffy (1901-1972), který po 2. světové válce založil Společnost pro obecné rozvíjení teorie systémů. Dnes již teorie systémů pronikla do většiny věd a pojem "systém" má internacionální charakter. Proto také existuje celá řada definic systému. (Systém | Andromedia.cz. Andromedia.cz | Sdílením informací ke konkurenční výhodě [online]. © [cit. 2019-03-06].)

Dnes je systém chápan hlavně jako účelově definovaná množina prvku a vazeb mezi nimi a pojem systém se užívá jako označeni určité časti reálného světa s charakteristickými vlastnostmi. Informační systém je z tohoto pohledu systémem umělým a to znamená, že člověk může jeho kvalitu značnou mírou ovlivňovat. Existuje celá řada definic informačního systému.

Informační systém lze definovat jako soubor lidi, metod a technických prostředků zajišťujících sběr, přenos, uchovaní, zpracovaní a prezentaci dat s cílem tvorby a poskytovaní informaci dle potřeb příjemců informaci činných v systémech řízeni. Další definice popisuje informační systém z jiného pohledu: „Informační systém je obecně podpůrný systém pro systém řízení. Jestliže chceme projektovat systém řízení jako takový, musíme znát, jaké jsou cíle, a informační systém řešit tak, aby tyto cíle podporoval.“ Jedno mají uvedené definice společné – shoduji se v tom, že informační systém je účelnou formou využiti informačních technologii1 v sociálně-ekonomických systémech. (TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy)

Je nutné podotknout, že námi vytvářený systém je podnikovým informačním systém typu ERP. ERP (Enterprise Resourse Planning – Podnikové plánování zdrojů) je proces, při kterém podnik spravuje a integruje důležité části svého podnikání. Existuje mnoho softwarových aplikací ERP, které pomáhají firmám provádět plánování zdrojů tím, že integrují všechny procesy, které potřebují, aby zapojily firmu do jednoho systému.

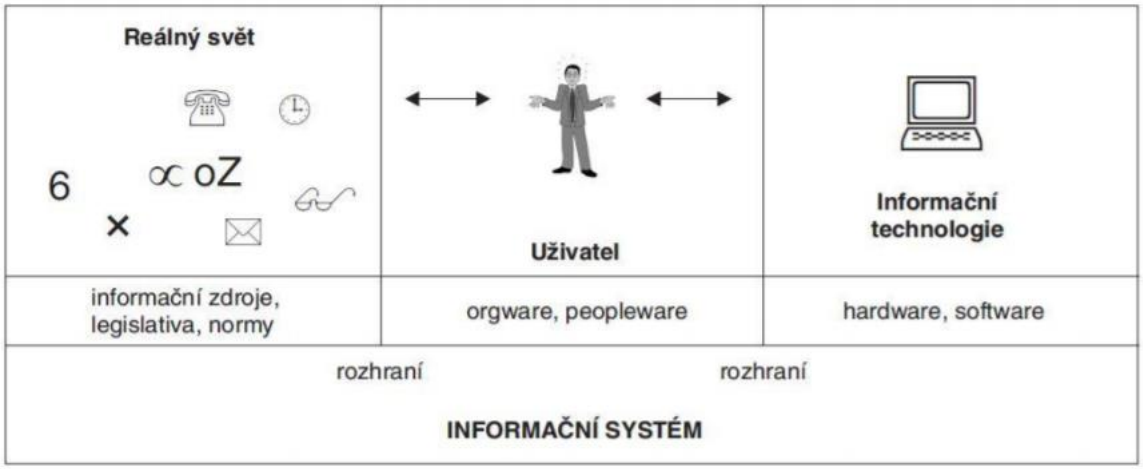
Softwarový systém ERP může integrovat plánování, nákup zásob, prodej, marketing, finance, lidské zdroje a další. (Enterprise Resource Planning (ERP) Definition. Investopedia – Sharper Insight. Smarter Investing. [online]. © 2010.)

3.2.2 **Struktura**

Informační systém se skládá z následujících komponent:

- technické prostředky (hardware) – počítačové systémy různého druhu a velikosti, doplněné o potřebné periferní jednotky, které jsou v případě potřeby propojeny prostřednictvím počítačové sítí a napojeny na paměťový subsystem pro prací s velkými objemy dat;
- programové prostředky (software) – tvořené systémovými programy, řídícími chod počítače, efektivní prací s daty a komunikaci počítačového systému s reálným světem, a programy aplikačními, řešícími určité třídy úloh určitých tříd uživatelů;
- organizační prostředky (orgware) – tvořené souborem nařízení a pravidel, definujících provozovaní a využívání informačního systému a informačních technologii;
- lidská složka (peopleware) – řešení otázky adaptace a účinného fungovaní člověka v počítačovém prostředí, do kterého je vražen;
- reálný svět (informační zdroje, legislativa, normy) – kontext informačního systému. (TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy)

Obrázek 1 Prvky informačního systému



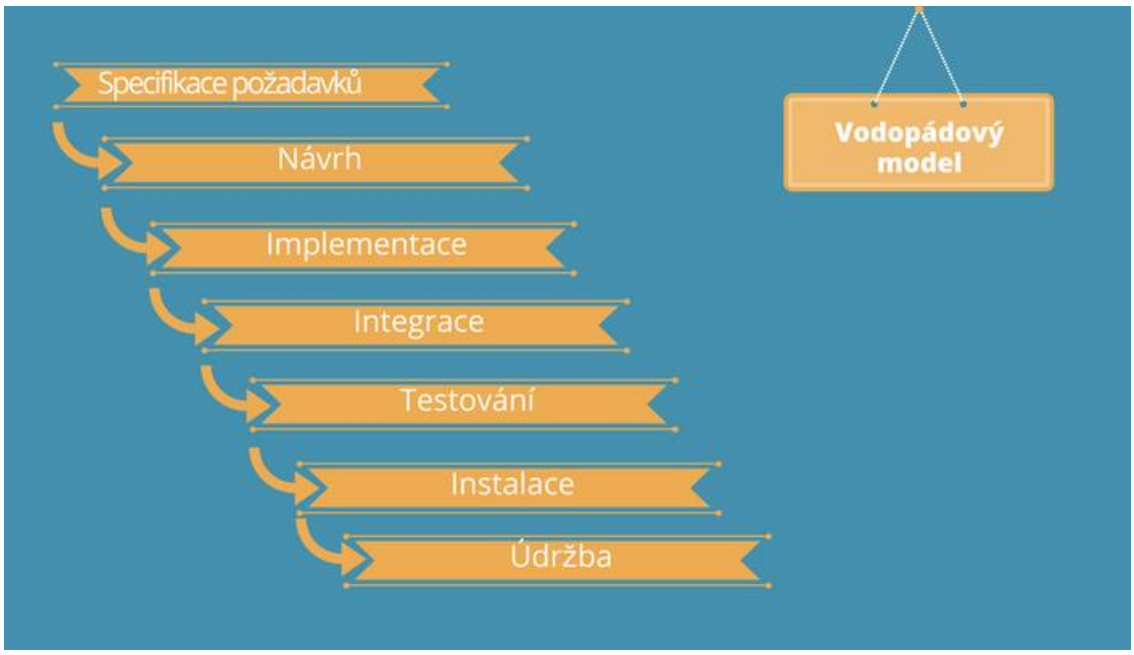
Zdroj: TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy

3.2.3 Etapy vývoje

Životní cyklus vývoje softwaru (anglicky SDLC – Software Development Life Cycle) je systematický proces budování softwaru, který zajišťuje kvalitu a správnost vytvářeného softwaru.

SDLC se skládá z podrobného plánu, který vysvětluje, jak plánovat, vytvářet a udržovat konkrétní software. Každá fáze životního cyklu SDLC má svůj vlastní proces a výstupy, které se přivádějí do další fáze.

Obrázek 2 vývojový cyklus softwaru (vodopadový model)



Zdroj: Metody vývoje aplikací. Waterfall, V-model, Inkrementální model. iQuest blog [online]. © 2010

Na obrázku je uveden vodopadový model, což znamená, že byla použitá metoda, kdy každá další fáze začíná až po skončení fáze předchozí.

Tato metoda není obvyklá pro větší firmy, kde se zabývají spíš většími projekty, protože jejím hlavním nedostatkem je velká doba od zahájení do konce projektu, jelikož se mezitím často mění požadavky a ztrácí se kontakt s potřebami uživatelů. Proto je ve větších firmách zvykem že se fáze vzájemně překrývají, např. ještě před skončením návrhu začne implementace. Také se provádí časté kontroly a rekapitulace a zákazník je seznámen s průběžnými výsledky.

Metoda vodopad je ale naprosto vhodná k uplatnění během vývoje našeho informačního systému, jelikož ten veškerými fázemi vývoje je provázen jedním člověkem.

Etapa 1: analýza požadavků

Ve fázi analýzy se snažíme o vytvoření seznamu známých problémů a návrhů jejich řešení. Často se vytváří **případy užití** známé pod výrazem **use case**, které je možné zapisovat ve formě textu nebo častěji pomocí use case diagramů (nejčastěji v jazyce UML). Prostřednictvím případů užití je popsáno to, jaké funkčnosti jsou od aplikace vyžadovány, ale ještě není definováno to, jakým způsobem bude aplikace určité věci dělat.

Dále jsou definované osoby, které budou systém používat a podle toho je pak upravována přehlednost systému a navrhováno grafické rozhrání.

Analýza požadavků se obecně provádí s cílem udržovat dohody se zákazníky a dalšími zainteresovanými stranami o tom, co by systém měl dělat a proč, a také proto, aby vývojáři lépe pochopili požadavky na systém. Jelikož v případě daného systému v roli zákazníka, návrháře a vývojáře vystupuje stejná osoba, analýza se provádí hlavně s účelem odhadu času na vytvoření systému. (Disciplína sběr a analýza požadavků [online]. © [cit. 2019-03-06].)

Podrobně je tato fáze popsaná v kapitole 3.3.

Etapa 2: návrh

Zahrnuje návrh aplikace, síť, databáze, uživatelská rozhraní a systémová rozhraní. kontrolujte navrhovaný návrh.

Během fáze návrhu dochází k postupnému upřesňování základních požadavků. Projekt se případně dělí na menší části a subprojekty. Jsou modelovány funkce a objekty, navrhuje se datový model a rozhraní. Nejprve je vytvořen hrubý návrh, který je následně podroben opětovnému rozboru, během něhož se jednotlivé součásti systému definují do co nemenších detailů. Je zkontrolováno, že konečný návrh splňuje požadavky uvedené v první fázi. (Rizeni-kvality-softwaru. soom.cz. [online]. © 2019.)

Na datový model se můžete podívat v kapitole 3.4. Rozhraní jsou popsaná v kapitole 3.10.

Etapa 3: implementace

Po skončení fáze návrhu systému následuje další fáze – implementace neboli kódování. V této fázi začínají vývojáři vytvářet celý systém napsáním kódu pomocí vybraného

programovacího jazyka, v našem případě C#. Ve fázi kódování bývají úkoly rozděleny do jednotek nebo modulů a přiděleny různým vývojářům.

V této fázi je potřeba dodržovat se určitých předdefinovaných pokynů pro kódování. Používají se programovací nástroje jako kompilátor, překladač a debugger pro generování a implementaci kódu. Výstup této fáze je testovatelný funkční software. Jedná se o nejdelší fázi životního cyklu vývoje softwaru. Celá tato fáze je popsaná v kapitole Vlastní práce. (SDLC (Software Development Life Cycle) Tutorial: What is, Phases, Model. [online]. © [cit. 12.03.2019].)

Etapa 4: testování

Jakmile je software dokončen, je nasazen v testovacím prostředí. Testovací tým začne testovat funkčnost celého systému. To je provedeno, aby ověřila, zda celá aplikace pracuje podle požadavků zákazníka. Ověřuje se hlavně, zda aplikace vrací na konkrétní vstupy očekávané výstupy.

Během této fáze mohou QA a zkušební tým najít některé chyby / závady, které sdělují vývojářům. Vývojový tým tuto chybu opraví a pošle zpět na QA k opětovnému testování. Tento proces pokračuje, dokud není software bezchybný, stabilní a pracuje podle obchodních potřeb tohoto systému. (SDLC (Software Development Life Cycle) Tutorial: What is, Phases, Model. [online]. © [cit. 12.03.2019].)

Testy se dělí do dvou hlavních kategorií. První z nich je testování funkčností (functional testing), druhá pak takzvané nefunkční testování (non-functional testing). (Funkční a nefunkční testy | Testování softwaru. Testování softwaru [online]. © [cit. 12.03.2019].)

Functional testing

Functional testing je proces, který ověřuje, zda program, aplikace atd. fungují tak, jak bylo specifikován. To znamená, že základním principem těchto testů je otestovat každý use case ze specifikace.

Functional testing je typ black-box testingu. To znamená, že se jedná o takové testování, které testuje pouze funkčnost a nehledí na to, jak je tato funkčnost implementována. Opak black-box testingu je white-box testing (White-box, neboli průhledné/skleněné testování), při kterém se tester soustředí ne jen na funkčnost, kterou aplikace poskytuje, ale také na to, jak je tato funkčnost implementována. Do functional-

testingu patří zejména tyto typy testování: unit testing, integration testing, smoke testing, system testing. (Types of Software Testing: Different Testing Types with Details. Software Testing Help – A Must Visit Software Testing Portal[online]. © 2006 [cit. 12.03.2019].)

Unit testing je jeden ze základních typů testování, kterým by měla projít každá aplikace nebo program. Jedná se o testy, které testují funkčnost jednotlivých units – jednotek. Jednotkou se v tomto případě může rozumět softwarová komponenta, modul nebo i jednoduchá a základní třída programu. Unit testy píše většinou sám programátor. (Integrační Testování | Testování softwaru. Testování softwaru [online]. © [cit. 12.03.2019].)

K unit testům bývají zařazené do fáze implementace a mohlo by se k nim přistupovat dvěma způsoby. První z nich je ten, že unit test se napíše až po implementování samotné funkčnosti. Tedy napíše se, například, metoda na sčítání dvou čísel, a až pak se k ní napíšou unit testy, které ověřují její funkčnost.

Druhým způsobem, jak k unit testům přistupovat je takový, že se testy nepíší až po naimplementování funkčností, ale ještě před začátkem implementace. V praxi to vypadá tak, že se napíše mnoho testů, které ze začátku budou končit neúspěchem. Postupně se k testům připisuje implementace.

V kapitole 4.3.1 je popsané testování pomocí unit testů, které byly provedeny až po implementaci.

Smoke testing

Smoke testing přijde na řadu tehdy, když je aplikace provozuschopná, tedy je dokončen její vývoj a lze ji spustit na testovacím prostředí. Tester v těchto testech provede krátké ověření, že hlavní části aplikace fungují a při letmém pohledu vše funguje tak, jak se očekává. Smoke testy se pokrývají většinou jen hlavní funkčnosti aplikace/programu. Rozsah smoke testů je tady malý, a proto jsou často automatizovány pro ulehčení práce.
(Types of Software Testing: Different Testing Types with Details. Software Testing Help – A Must Visit Software Testing Portal[online]. © 2006.)

Integration testing

Integrační testování přichází na řadu po vývojářských testech. Tyto testy připravuje testovací tým. Úkolem těchto testů je otestovat komunikaci a spolupráci mezi komponentami aplikace. Lze testovat nejen mezikomponentovou spoluprácí, ale také spolupráci mezi jednotlivými

komponentami a operačním systémem nebo hardwarem, se kterým daná aplikace/program spolupracují a jsou závislé na fungování těchto částí.

Integračního testování se využívá spíše pro velké aplikace, které se skládají z mnoha komponent (někdy i z několika aplikací). (What is Integration Testing (Tutorial with Integration Testing Example). Software Testing Help – A Must Visit Software Testing Portal [online]. © 2006.)

System testing

Systémové testování, které se označuje jako SIT (System Integration Tests) většinou následují po integračním testování. Toto testování mají za úkol ověřit funkčnost aplikace jako celku. Testovací tým dá dohromady testovací scénáře, které simulují různé události a kroky, které v aplikaci mohou nastat. Pokud tyto testy odhalí chybu, chyba je opravena a testy jsou provedeny znovu. (Software Testing Help – A Must Visit Software Testing Portal [online]. © [cit. 12.03.2019].)

Akceptační testování

UAT – zkratka pro User Acceptance Test. Akceptační testování probíhá ze strany zákazníka, kterému software dodáváme. Většinou probíhá na jeho prostředí (serverch, pc atd.). Tato fáze testování nastává až na samém konci testování, a to pouze v případě, že všechny ostatní testy dopadly relativně uspokojivě. Testovací scénáře akceptačních testů si může vymyslet zákazník sám, ale nejčastěji jsou scénáře vymýšleny dohromady společně s dodavatelem softwaru. Průběh akceptačních testů je klíčový pro úspěch celého vývoje, tedy pokud jsou při tomto testování nalezeny závažné chyby, které mají vliv na požadovanou funkčnost programu, tým vyvíjející aplikaci je musí co nejdříve opravit a nasadit opravu na prostředí u zákazníka a testovat znovu. Integrační Testování | Testování softwaru. Testování softwaru [online]. © [cit. 12.03.2019].)

Performance testing

Úkolem pefromance testingu je otestovat aplikaci z hlediska rychlosti. Zda například načítání jednotlivých oken netrvá příliš dlouho nebo zda aplikace při velké zátěži nespadne. Do performance testingu řadíme například tyto testy:

Volume testing – aplikace je zahlcena velkým množstvím dat. Tester sleduje, zda se aplikace v tomto případě chová správně.
Load testing – úkolem load testingu je zjistit, jak velké zatížení aplikace snese.
Dalším důležitým testováním je Recovery testing. Toto testování ověřuje, jak se aplikace dokáže vypořádat s neočekávanými chybami (například že při jedné chybě celá aplikace nespadne, ale běží a funguje normálně dál.) Například když aplikace přijímá tok dat a tento tok se v půli přeruší. (What is Integration Testing (Tutorial with Integration Testing Example). Software Testing Help – A Must Visit Software Testing Portal [online]. © 2006.)

Regresní testování

Regresní testování je jedním z nejdůležitějších typů testování vůbec. Pokud se k aplikacím vydávají nějaké updaty (nové verze), což bývá u většiny větších aplikací, bez regresních testů by to nešlo. Úkolem těchto testů je otestovat, zda nová funkcionalita neovlivnila a nerozbila stávající, již otestovanou a funkční aplikaci. Tyto testy by se měli provádět vždy po každé střední a větší změně v aplikaci. (Regression Testing Complete Guide: Tools, Method, and Example. Software Testing Help – A Must Visit Software Testing Portal[online]. © 2006.)

Etapa 5: nasazení

Až když je fáze testování softwaru ukončena a v systému nejsou chyby, začne proces konečného nasazení. Na základě zpětné vazby poskytnuté manažerem projektu je konečný software vydán a zkontrolován, zda existují nějaké problémy s nasazením. Při této fázi dochází nejen k instalaci softwaru u zákazníka ale také ke školení cílových uživatelů. Součástí je také počáteční podpora, dohled nad zkušebním provozem a případné odstraňování chyb. (Rizeni-kvality-softwaru. soom.cz. [online]. © 2019.)

Etapa 6: údržba

Jakmile je systém nasazen a zákazníci začnou používat vyvinutý systém, dojde ke třem aktivitám:

- Opravy chyb – chyby jsou hlášeny kvůli některým scénářům, které nejsou vůbec testovány.
- Aktualizace – aktualizace aplikace na novější verze.

- Vylepšení – přidání některých nových funkcí do stávajícího softwaru.

Hlavním cílem této fáze SDLC je zajistit, aby byly i nadále splněny potřeby a aby systém pokračoval v plnění podle specifikací uvedených v první fázi. (SDLC (Software Development Life Cycle) Tutorial: What is, Phases, Model. [online]. © [cit. 12.03.2019].)

3.3 Analýza požadavků

Vývoj systému začíná analýzou požadavků. Cílem je definovat požadavky na software a popsat jeho funkčnost.

V první řadě si stanovíme, kdo v rámci firmy bude aplikaci využívat a co od systému očekává. Předpokládáme, že vyvíjíme pro malou firmu s pěti až patnácti zaměstnanci. Organizační struktura se skládá z vedení a obsluhy půjčovny. Informační systém bude využíván hlavně obsluhou, která se stará o správu položek. Vedení bude systém využívat spíše pro správu zaměstnanců.

3.3.1 Procesy v podniku

Proto, aby informační systém plnil svoji funkci je potřeba vymezit jaké přesně procesy v podniku probíhají a k uchování jakých informací dochází. Podnikový proces poskytuje správnou informaci správnému jednotlivci ve správný čas k vykonání potřebného úkolu, a tak říká co – jak – kdy – kdo má dělat v rámci podniku. Každý proces musí mít zodpovědnou osobu, která se nemusí nutně procesu účastnit, ale je zodpovědná za jeho celkový výsledek. (Podnikový proces (Business process) - ManagementMania.com. [online]. © 2011.)

Pro lepší pochopení běhu podniku jsou v příloze číslo 1 graficky znázorněné podnikové procesy podle pravidel BPMN (Business Process Model and Notation).

3.3.2 **Entity**

Hlavní funkcionalitou našeho informačního systému je evidence položek, objednávek, zákazníků a zaměstnanců. Podíváme se podrobněji na každou z těchto entit a vymezíme si vlastnosti (atributy), které budeme chtít u každé entity sledovat a ukládat do databáze.

3.3.2.1 Položky

Jedním z hlavních důvodu, proč vytváříme informační systém pro půjčovnu je ten, že chceme evidovat položky půjčovaného oděvů. Kontrolovat jejich dostupnost a zjišťovat stav. Dále následuje výčet atributů, které sledujeme u položek:

- název
- ilustrace – fotografie dané položky, důležitá pro srovnání s fyzickým objektem, nebude implementovaná
- velikost
- barva
- popis – možnost ke každé položce přidat komentář
- stav – během životního cyklu se může položka nacházet v různých stavech: dostupná, zapůjčena, v čistírně, vyřazená z evidence.
- dostupnost – termíny, kdy je položka dostupná
- cena za půjčení
- prodejní cena – cena za případný odprodej položky zákazníkovi

3.3.2.2 Rezervace

U rezervací sledujeme následující atributy:

- položka
- zákazník
- popis
- datum vytvoření objednávky
- datum zahájení rezervace – datum, kdy je zákazníkovi položka fyzicky předaná
- datum ukončení rezervace – datum, kdy má zákazník povinnost vrátit položku na půjčovnu
- zrušená – určuje, zda byla objednávka zrušena
- pojištění – částka, na kterou bylo sjednáno pojištění položky
- jiné náklady – cena za případné dodání, pokuta za zpožděné vrácení atd

- sleva
- záloha
- celková cena
- zbývá uhradit

3.3.2.3 Lidé

Je také potřeba evidovat lidí, které se účastní procesů ve firmě, a to zejména zákazníky a zaměstnance. U obou dvou budeme sledovat stejné atributy:

- jméno
- příjmení
- datum narození
- adresa
- telefonní číslo
- e-mail

Každá z těchto entit má ale také další atributy. U zákazníku budeme navíc ukládat informace o tom, zda souhlasí se zasíláním notifikací na uvedenou e-mailovou adresu. U zaměstnanců si zase uložíme informace o tom, kdy k nám nastoupili, jestli ve firmě pořád působí, jejich pracovní pozici a hodinovou mzdu.

3.3.3 **Uživatele**

Zavedení informačního systému můžeme nazvat úspěšným teprve tehdy, když jsou s ním koncové uživatelé spokojení, protože splňuje jejích požadavky na funkčnost a nabízí příjemné uživatelské prostředí a intuitivní ovládaní. Proto je důležité ještě před začátkem návrhu zjistit, kdo je cílovou skupinou uživatelů vytvářeného softwaru. Je totiž velký rozdíl v tom, kdo bude informační systém používat (aplikace pro lidí v důchodu by měla mít jiný vzhled než aplikace pro dospívající, u kterých se navíc očekává, že bez problémů zvládnou použití aplikaci, která nabízí víc funkcí, ale horší přehlednost). Proto si nyní namodelujeme typické uživatelé systému, tak zvané **Persony.**

<u>Persona 1</u>

Jmeno: Zdena Melicharová.
Věk: 21.

Popis: Zdena je studentkou medicíny, která pracuje v půjčovně o prázdninách jako brigádnice. Je praktická, šetrná a má ráda pořádek. Ráda komunikuje s lidmi, na zákazníky je vždy příjemná a milá.
Co od systému očekává: Především přehlednost a rychlou odezvu.

Persona 2

Jméno: Leo Kowalski.
Věk: 25.
Popis: Leo je budoucí módní návrhář. Je stálým zaměstnancem půjčovny. Vyzná se v módě a zákazníkům vždy dobře poradí. Má rád pastelové barvy.
Co od systému očekává: Přehlednost a hezký design. Jelikož často přichází do styku s informačním systémem, je pro něj důležité, aby byl proveden v příjemných, ne moc výrazných barvách, které spolu dobře ladí.

Snahou bude udělat systém co nejvíc přehledný, responzivní, intuitivní, a nakonec navrhnout hezký design.

3.3.4 Systémové role

Jedním z hlavních požadavků na informační systém je jeho bezpečnost. To znamená, že by se v rámci systému každý mohl dostat jen k těm informacím a spouštět jen ty funkčnosti, na které pravomoc má a nesměl se dostat k informacím a funkčnostem, které mu nesmějí být přístupné. Proto nepovolíme všem uživatelům přístup ke všemu v systému, ale obsadíme je do rolí, ke kterým jsou v rámci systému přiřazená oprávnění. Tyto role definují rozsah pravomoci do ní obsazeného uživatele. Vytváříme tyto role:

Obrázek 3 Systémové role

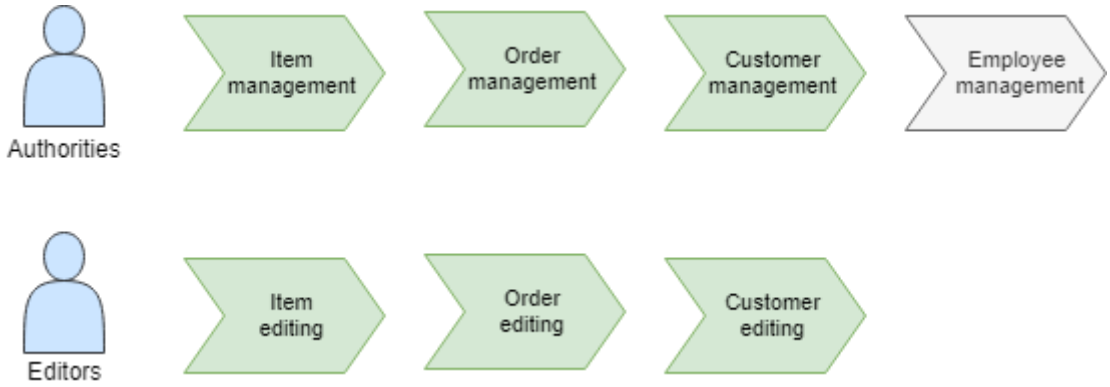
Authorities – nejsilnější role. Reprezentuje vedení firmy.

Editors – reprezentuje obsluhu půjčovny. Jedná se o zaměstnance, kteří spravují sortiment půjčovny.

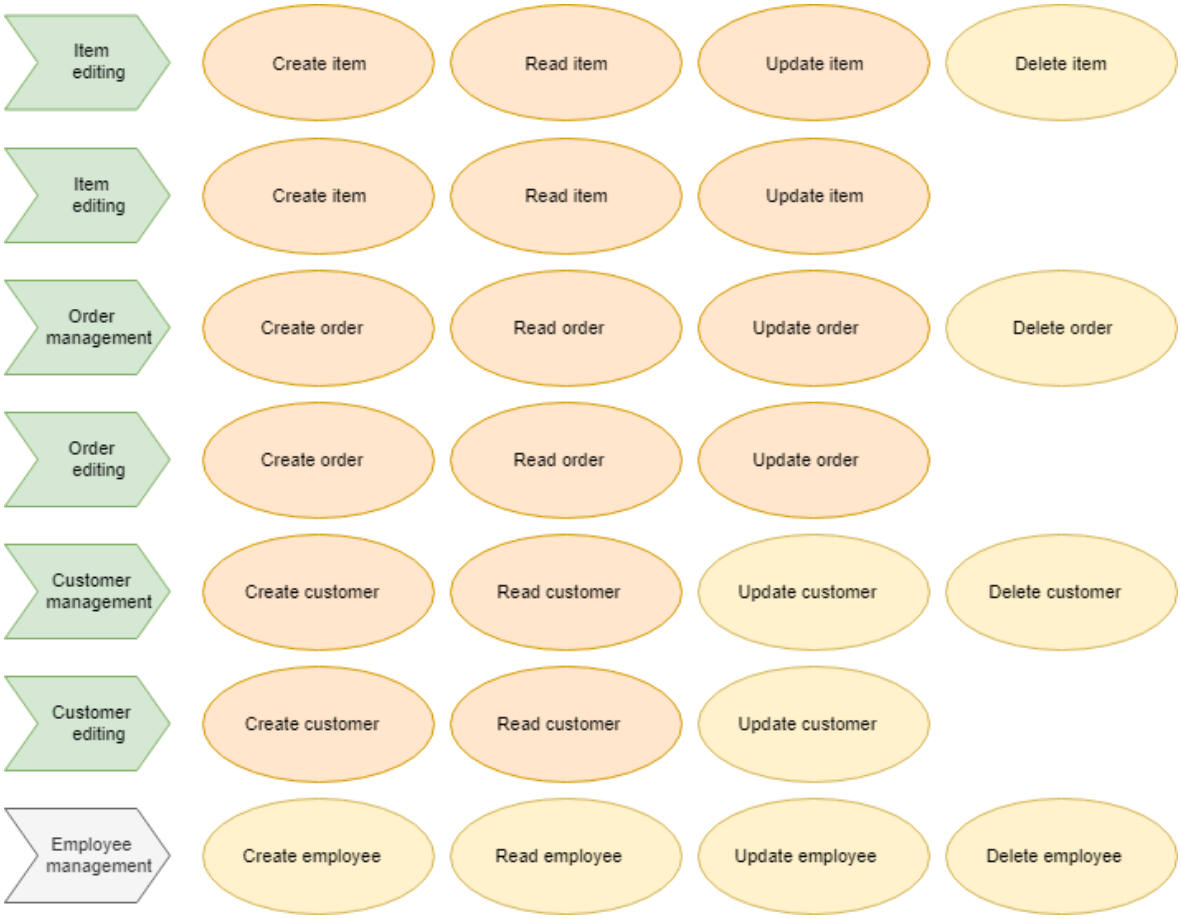
3.3.5 Procesní dekompozice a případy užití

V této kapitole jsou popsané procesy, podporu kterých informační systém nabízí, a případy užití (use cases) které je tvoří. Je nutné podotknout, že dále nejsou uvedené všechny procesy, ale jen ty, které souvisí se správou entit. Není zde uveden proces registrace či přihlašování do systému.

Obrázek 4 **Procesní dekompozice**



Obrázek 5 **Use casy**



Růžovou barvou jsou označeny případy užití, které jsou pro běh firmy důležitější, tj. mají větší význam v kontextu hlavního cíle podniku, kterým je dosahování zisku. To stejné platí pro procesy, které jsou označeny zelenou barvou.

Tabulka 1 **Informace o use casech**

Název use case	**Priorita**	**Popis**
Create item	1	Vytváří novou položku oděvu.
Read item	1	Vrací vybranou položku nebo číslovaný seznam položek.
Update item	1	Modifikuje existující položku – upravuje její stav či popis.
Delete item	3	Smaže položku. V případě, že není rezervovaná žádným zákazníkem.

Create order	1	Vytváří novou rezervaci.
Read order	1	Vrací vybranou rezervaci nebo číslovaný seznam rezervací.
Update order	2	Modifikuje existující rezervaci – upravuje její stav či popis.
Delete order	3	Smaže rezervaci. Provádí se automaticky v případě, kdy je smazán zákazník, ke kterému se rezervace vztahuje.
Create customer	1	Vytváří nový objekt zákazníka.
Read customer	1	Vrací vybraný objekt zákazníka nebo číslovaný seznam všech zákazníků.
Update customer	2	Modifikuje existující objekt zákazníka – upravuje např. jeho adresu či telefonní číslo.
Delete customer	3	Smaže záznam o zákazníkovi.
Create employee	2	Vytváří nový objekt zaměstnance.
Read employee	2	Vrací vybraný objekt zaměstnance nebo číslovaný seznam všech zaměstnanců.
Update employee	3	Modifikuje existující objekt zaměstnance – upravuje např. jeho adresu či telefonní číslo.
Delete employee	3	Smaže záznam o zaměstnanci.

Role authorities a editors mají podobná oprávnění k práci s položkami oděvu (items), s rezervacemi (customerOrders nebo jenom orders) a se zákazníky (customers). Mohou tyto entity vytvářet, číst nebo upravovat. Authorities navíc mohou tyto entity mazat. Dále je

rozdíl ve správě zaměstnanců (employees): zatímco authorities mají přístup k informacím a funkčnostem spojenými se všemi zaměstnanci, editors mohou vidět jen informace o sobě (v případě, že jsou zaměstnanci firmy). Zpřístupňování pracovníkům informací o jiných zaměstnancích nebo jejích upravování nejsou pro běh firmy nijak přínosné, a dokonce i nežádoucí.

3.3.6 **Vymazávání objektů**

CRUD – zkrátka, která shrnuje čtyři základní operace nad záznamem v trvalém úložišti. Create – vytváření, Read – čtení, Update – aktalizace, Delete – odstraňování. Tento systém je ale navrhován tak, že žádný uživatel obsazený v roli editors není oprávněn provádět operaci delete nad žádným objektem.

Informace v podniku mají hodně vysokou cenu a proto, aby se tyto informace neztrácely, obecně platí, že vše, co kdy bylo do systému zaznamenáno, by nemělo jít jednoduše smazat. V informačním systému půjčovny existuje jen málo případů, kdy je potřeba trvale odstranit záznamy z databáze. Může se stát, že někdo omylem vytvoří objekt a chce ho odstranit z důvodu chyby, což by se nemělo stávat často. Nicméně i tohle je možné a v tomhle případě stačí objekt jen upravit a není ho nutné odstraňovat. Znázorňování neexistence či neplatnosti entit je potřeba řešit jiným způsobem než vymazáním. U každého objektu se to řeší jinak:

Item

K fyzickému odstranění položky z evidence může dojít ze dvou hlavních důvodů: položka byla odkoupena zákazníkem nebo odepsaná z důvodu opotřebení či poškození. Trvale odstranění položky by vedlo k tomu, že by se pomocí systému nikdy nedalo zjistili, že daná položka byla v systému evidována. Pro znázornění absence položky použijeme operaci update. V rámci aktualizace změníme stav položky např. na written-off(odepsaný). Tímhle způsobem odstraníme položku z nabídky dostupných a zároveň ji nevymažeme z databáze. Nicméně role authorities má oprávnění položky mazat.

Order (CustomerOrder)

Z důvodu pozdějších reklamací či nejasností nebo pro statistické účely je potřeba uchovávat všechny rezervace. Nejsou závažné důvody proč vymazávat rezervace z databáze. K rezervaci je bezpochyby vázaná položka, která tím pádem se zobrazuje v systému jako

nedostupná po dobu spojitosti s rezervaci. V případě, že zákazník rezervaci zruší, je potřeba dané položky "uvolnit". Můžeme to udělat bez odstraňování rezervace. Stejně jako v případě položky stačí nastavit stav rezervace např. na cancelled(zrušená). Záznam o rezervaci se z databáze odstraní jenom v případě, když musíme z databáze smazat zákazníka, který je k rezervaci vázaný. Tato operace se provádí automaticky při odstranění příslušného zákazníka. Jinak rezervaci smazat nelze.

Customer

Ukládáme v systému veškeré potřebné informace spojené se zákazníkem: v případě nedodržení smlouvy a nevracení zákazníkem zapůjčených položek včas, musíme evidovat jeho kontaktní údaje, také můžeme dohledat jaké položky si kdy půjčoval, pomocí případných komentářů můžeme zjistit, jestli je vracel v pořádku atd. Není v zájmu firmy odstraňovat z databáze zákazníky. Nicméně zákazníci jsou entitou, smazání, které musí být umožněno ze zákona. Předpokládá se ale, že k tomu dojde jenom v případě, že si o smazání informaci o sobě řekne sám zákazník. Při odstraňování zákazníka jsou vždy odstraněny všechny jeho objednávky.

Employee

Uživatel obsazený do role Authorities oprávnění odstraňovat záznamy o zaměstnancích. Tuto funkčnost by ale neměl zneužívat. Platí tu stejné pravidlo jako při smazání zákazníka – doporučuje se mazat zaměstnance jen v případě, kdy o to sám požádá.

3.4 Návrh datového modelu

Databázový model definuje logický návrh a strukturu databáze a definuje, jak budou data uložena, přístupná a aktualizována v systému správy databáze. Existují tyhle typy datových modelů:

- Relační model
- Hierarchický model
- Síťový model
- Model modelu entit

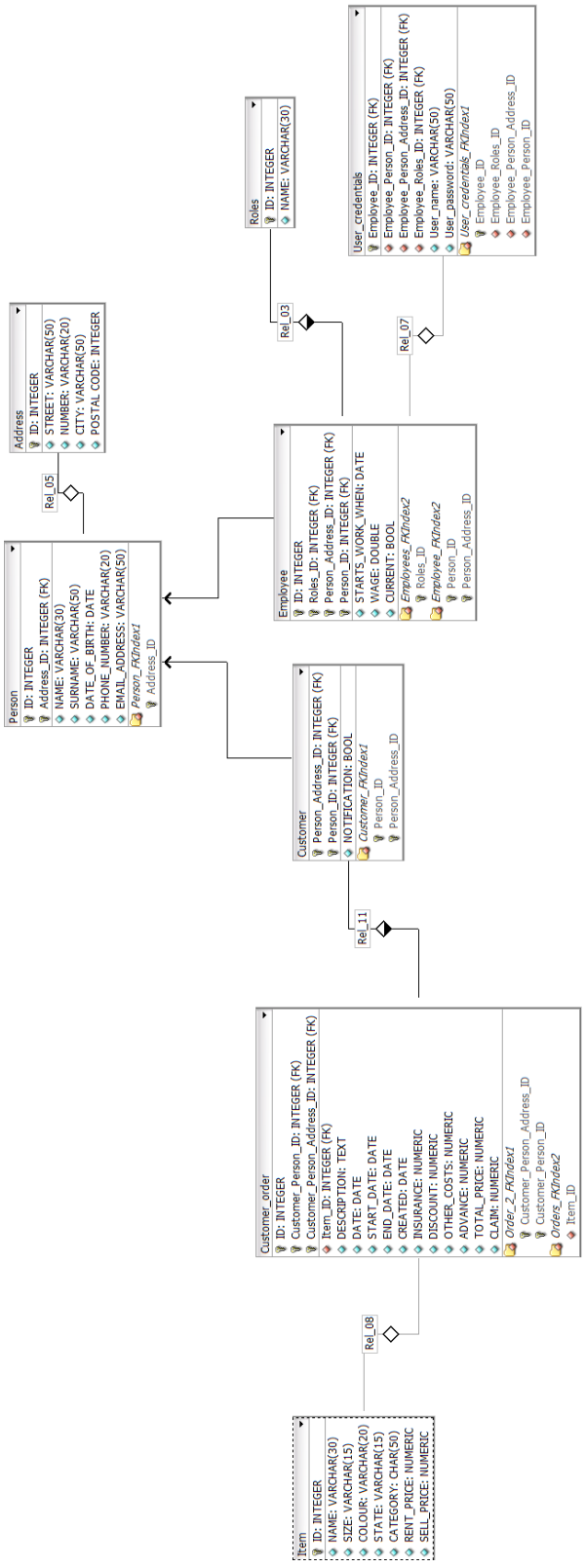
My že použijeme ten nejpopulárnější z nich – relační. V tomto modelu jsou data uspořádána ve dvourozměrných tabulkách a vztah je udržován uložením společného pole.
Relační model byl zaveden E.F Coddem v roce 1970 a od té doby byl nejpoužívanějším databázovým modelem, v podstatě se dá říct, že je to jediný databázový model používaný po celém světě.

Základní strukturou dat v relačním modelu jsou tabulky. Veškeré informace týkající se konkrétního typu jsou uloženy v řádcích této tabulky. Proto jsou tabulky také známé jako relace v relačním modelu. (Database Models in DBMS | Studytonight. Studytonight - Best place to Learn Coding Online [online]. © 2019.)

Jinými slovy, datový model je schéma, které znázorňuje, jaké objekty se do databáze ukládají a jakými vazbami jsou provazovány. S každým objektem se také ukládají jeho atributy a jejich datový typ. Z přednášek vím, že model je zjednodušení reality, kterou se snažíme popsat co nejpřesněji, proto je důležité danou etapu nepodcenit a zachytit veškeré informace, které je potřeba ukládat. Entity a jejich vlastnosti jsou popsané v kapitole 3.3.2, z níž budeme vycházet.

Pro tvorbu daného modelu je použita předem instalovaná aplikace DBDesigner Fork. Výhodou použití tohoto softwaru je to, že je možné vytvořený model transformovat přímo do databázového klienta, kde se na jeho základě automaticky vytvoří příslušné tabulky.

Obrázek 6 **Datový model**



4. Vlastní práce

4.1 Backend

4.1.1 Databáze

Nejdůležitější funkcí informačního systému je ukládání dat. Proto ještě před vytvořením nového projektu ve Visual Studio je navržené databázový model a zajištěno spojení se serverem.

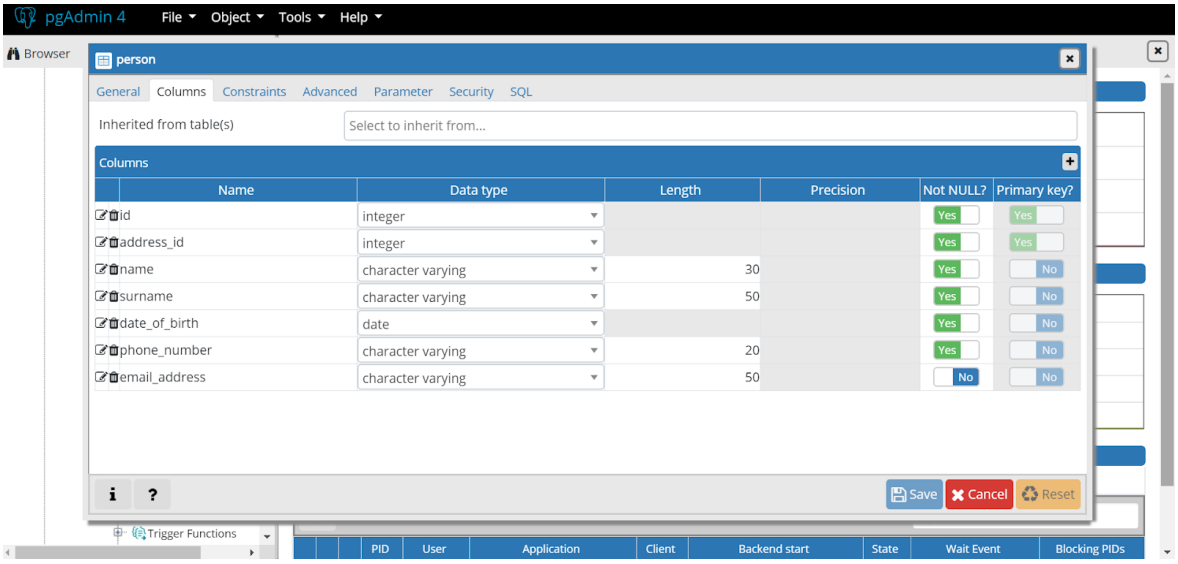
4.1.1.1 Přístup k databázovému stroji

Jako databázový klient pro PostgreSQL je použit pgAdmin III. Byl vybrán, protože je na jednu stranu grafickou aplikací se snadným ovládáním, současně ale umožňuje přímou práci v jazyce SQL a i poskytuje pro ni vydatnou podporu.

K databázovému stroji (serveru) se lze připojovat jak přímo (samozřejmě s možností využití SSL/TLS, a to i klientským certifikátem, pokud je třeba), tak i SSH tunelem. Heslo pro ověření uživatele si lze uložit, ale tahle funkčnost z bezpečnostních důvodu není použitá. Vytvoříme si databázi na localhostu, což znamená že v téhle fázi.

Jako ukázku prostředí pgAdmina uvádím sloupce tabulky person.

Obrázek 7 **Prostředí pgAdmin – tabulka person**



4.1.2 Vytváření nového projektu a připojování k databázi

Informační systém je vytvářen ve vývojovém prostředí Visual Studio, kde si pod názvem RentApp založíme nový projekt. Jelikož výsledný systém bude mít grafické rozhraní, pro nový projekt je použitá aplikace Windows Forms App (.NET Framework).

Dále je založená složka DatabaseConnection, do které se vytváří stejnojmenná třída, kde bude popsané připojení k databázi a odpojení od ní. Pro přístup do PostgreSQL použijeme data provider Npgsql. Metoda pro vytváření připojení vypadá následovně:

```
        public static NpgsqlConnection createConnection()
        {
            NpgsqlConnection connection = new NpgsqlConnection("Host=localhost;
Username=postgres; " +
                "Password=postgres; Database=renting");
            if (connection.State != ConnectionState.Open)
            {
                connection.Open();
                log.info("Database connection opened.");
            }
            return connection;
        }
```

4.1.3 Logování

Je dobrou zkušeností zaznamenávat informace o činnosti a běhu softwaru – tak zvané logy. Takové to informace si budeme ukládat do textového souboru, který si pojmenujeme Main_log.txt. V případě, poruchy systému v provozu, logy slouží jako pomocník pro administrátora při zpětné analýze k rozpoznání toho, jak k jaké chybě došlo, a taky kdy přesně a jakým způsobem se to stalo. Také může být logování firemních systému užitečné

v případě zpětného vyšetřování závažné bezpečnostní události, např. vyšetřování interního zneužití informací. Na dané etapě, budeme do logovacího souboru zapisovat hlášení, upozornění a chyby, a využijeme ho jako pomocníka během vytváření systému.

Třída Logger vypadá následujícím způsobem:

```
class Logger
    {
        public static string LOGGER_FILE_PATH =
"C:/Users/User/Desktop/Bakalarka/Main_log.txt";

        public static string LEVEL_INFO = "INFO";
        public static string LEVEL_WARNING = "WARNING";
        public static string LEVEL_ERROR = "ERROR";

        public void info(string message)
        {
            File.AppendAllText(LOGGER_FILE_PATH, DateTime.Now.ToString() + " " +
LEVEL_INFO + ": " + message + Environment.NewLine);
        }

        public void warning(string message)
        {
            File.AppendAllText(LOGGER_FILE_PATH, DateTime.Now.ToString() + " " +
LEVEL_WARNING + ": " + message + Environment.NewLine);
        }

        public void error(string message)
        {
            File.AppendAllText(LOGGER_FILE_PATH, DateTime.Now.ToString() + " " +
LEVEL_ERROR + ": " + message + Environment.NewLine);
        }

    }
```

4.1.4 **Objekty**

Při vývoji budeme používat objekty, které jsou definovány v databází. Proto se vytváří nová složka Entities, do které jsou zakládány třídy odpovídajících objektů a v každé z nich jsou definovány atributy tak, jak jsou uložené v databázi. Taková reflexe databázových objektů není ale dostačující. V případě, že je nějaká entita provázaná s jinou, jako cizí klíč je v databázi uložen pouze odkaz na druhou entitu (id). Proto pro přistoupení k nějakému objektu přes jiný objekt je potřeba dvakrát volat do databáze. Jako výhodnější řešení bývá použito to, že namísto odkazu na jiné objekty se ty objekty ukládají celé. Založíme si složku Dto (DTO – data transfer object), do které si vytvoříme "DTO podobu" všech tříd.

Zkratka DTO vznikla z termínu Data Transfer Object a značí strukturu v datové vrstvě, která stojí mezi aplikací samotnou a datovým úložištěm. Aplikační vrstva tedy nevykonává

operace přímo nad datovým úložištěm, ale nad DTO objekty, které jí poskytuje datová vrstva. Pro chod celého programu je to vhodnější: jsou jasně vymezeny třídy oprávněné k zásahům do datového podkladu, a zabrání se tak případné nekonzistenci dat.

4.1.5 DAO metody

Majoritní funkčnost systému je uložena v DAO (Data Access Object) metodách. Proto si pro každý objekt vytvoříme třídu do nové složky Entities.dao. Z důvodu omezeného počtu povolených stránek pro tuto práci, jsou dále popsané jen vybrané metody, které většinou platí pro osobu, tj. třída PersonDAO.

PersonDAO/create

Popis: Vytvoří nový objekt v tabulce person. Je volán pouze v rámci založení nového zákazníka nebo zaměstnance.
Vstup: objekt typu PersonDTO.
Výstup: vytvořený objekt typu PersonDTO.
Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří databázové spojení.
3. Vygeneruje unakátní id (pomocí metody nextId()).
4. Nastaví id objektu na hodnotu vygenerovaného id.
5. Nastaví ostatní atributy objektu na hodnoty, které obdržela na vstupu.
6. Vytvoří objekt personDTO.
7. Vyhledá vytvořený objekt v databázi (pomocí metody get(int id)) a vrátí.

Script vypadá následovně:

```
        public PersonDTO create(PersonDTO personDTO)
        {
            // Get database connection.
            NpgsqlConnection connection = DatabaseConnection.createConnection();
            NpgsqlCommand createCommand = connection.CreateCommand();
            String sql = "INSERT INTO person VALUES (@id, @address_id, @name,
@surname, @date_of_birth, @phone_number, @email_address)";
            createCommand.CommandText = sql;
            // Generete unique id.
            int id = SequenceDAO.nextId();

            createCommand.Parameters.AddWithValue("id", id);
            createCommand.Parameters.AddWithValue("address_id",
personDTO.Address.Id);
            createCommand.Parameters.AddWithValue("name", personDTO.Name);
            createCommand.Parameters.AddWithValue("surname", personDTO.Surname);
```

```
            createCommand.Parameters.AddWithValue("date_of_birth",
personDTO.Date_of_birth);
            createCommand.Parameters.AddWithValue("phone_number",
personDTO.Phone_number);
            createCommand.Parameters.AddWithValue("email_address",
personDTO.Email_address);
            createCommand.ExecuteNonQuery();

            // Check the person was created.
            PersonDTO createdPerson = get(id);

            return createdPerson;
        }
```

Jak můžeme vidět, po samotném vytváření objektu je rovnou zkontrolováno, zda se objekt opravdu uložil. Proto metoda nevrací objekt rovnou po jeho založení ale až po tom, co si ho získá z databáze. Provádí to pomocí další metody, a to get(int id).

PersonDAO/get

Popis: Vyhledá a vrátí objekt z tabulky person. Je volán v případě vyhledávání zákazníka nebo zaměstnance.

Vstup: id osoby.

Výstup: vytvořený objekt typu PersonDTO.

Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří databázové spojení.
3. V tabulce person vyhledá a vrátí objekt podle zadaného id.
4. Podle nalezeného objektu, vyhledá a vrátí objekt addressDTO s id příslušné adresy.
5. Konvertuje Person na PersonDTO (pomocí metody convertToDTO(person, addressDTO)).
6. Vrátí objekt typu PersonDTO.

PersonDAO/convertToDTO

Popis: Transformuje objekt typu Person na PersonDTO.

Vstup: objekt typu Person, objekt typu AddressDTO.

Výstup: objekt typu PersonDTO.

Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří instanci objektu PersonDTO.

3. Nastaví atributy objektu na hodnoty, které obdržela na vstupu.
4. Vrátí objekt typu PersonDTO.

PersonDAO/update

Popis: Aktualizuje objekt v tabulce person. Je volán pouze v rámci aktualizace zákazníka nebo zaměstnance.
Vstup: objekt typu PersonDTO.
Výstup: vytvořený objekt typu PersonDTO.
Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří databázové spojení.
3. Nastaví atributy objektu na hodnoty, které obdržela na vstupu.
4. Aktualizuje objekt.

PersonDAO/delete

Popis: Smaže objekt v tabulce person. Je volán pouze v rámci smazání zákazníka nebo zaměstnance.
Vstup: id objektu.
Výstup: žádný.
Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří databázové spojení.
3. Smaže objekt s id obdrženým na vstupu.

CustomerOrder/create

Popis: Vytvoří novou rezervaci (nový záznam v tabulce customerOrder).
Vstup: objekt typu CustomerOrderDTO.
Výstup: vytvořený objekt typu CustomerOrderDTO.
Happy

SequenceDAO/nextId

V jedné z výše zmíněných metod je zmíněná meto nextId. Pomocí této metody je zajištěno generování unikátního id v databázi. Script vypadá následovně:

```
public static int nextId()
        {
            // Get database connection.
            NpgsqlConnection connection = DatabaseConnection.createConnection();

            int id = connection.Query<int>("select nextval('common_seq')").First();
            return id;
         }
```

UserCredentialsDAO/verifyUser

Další zajímavou DAO metodou je metoda verifyUser, která je vytvořená v souvislosti s ověřením osoby, která se snaží o přihlášení do systému.

Vstup: uživatelské jméno a heslo.

Výstup: id role uživatele.

Happy day scenario:

1. Provede validaci vstupních dat.
2. Vytvoří databázové spojení.
3. Vyhledá zaměstnance s příslušným uživatelským jménem a heslem.
4. Vrátí zaměstnance, pokud byl nalezen (pomocí metody employeeDAO/get).
5. Vrátí id role příslušného zaměstnance.

Script vypadá následovně:

```
        public int verifyUser(String username, String password)
        {
            // Get database connection.
            NpgsqlConnection connection = DatabaseConnection.createConnection();
            // Create parameters and execute select query.
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("user_name", username);
            parameters.Add("user_password", password);
            User_credentials result = connection.Query<User_credentials>("SELECT *
FROM User_credentials WHERE user_name = @user_name AND user_password =
@user_password", parameters).FirstOrDefault();
            if (result != null)
            {
                EmployeeDTO employee = employeeDAO.get(result.id_employee);
                return employee.Id_roles;
            }
            else {
                return 0;
            }
        }
```

Seznam všech DAO metod:

AddressDAO/create
AddressDAO/get
AddressDAO/update
AddressDAO/delete
AddressDAO/convertToDTO

CustomerDAO/create
CustomerDAO/get
CustomerDAO/update
CustomerDAO/delete
CustomerDAO/convertToDTO
CustomerDAO/findAll()

EmployeeDAO/create
EmployeeDAO/get
EmployeeDAO/update
EmployeeDAO/delete
EmployeeDAO/convertToDTO

ItemDAO/create
ItemDAO/get
ItemDAO/update
ItemDAO/delete
ItemDAO/convertToDTO

CustomerOrderDAO/create
CustomerOrderDAO/get
CustomerOrderDAO/update
CustomerOrderDAO/delete
CustomerOrderDAO/convertToDTO
CustomerOrderDAO/getUnavailableDates

SequenceDAO/nextId

UserCredentialsDAO/verifyUser

4.1.6 **Rozhraní**

Dědičnost je jedna ze základních vlastností OOP, která slouží k tvoření nových datových struktur na základě starých. Nejprve a především, rozhraní (interface) v C# jsou prostředky, jak získat chybějící dědičnost v C#, což znamená, že nemůžete dědit z několika tříd, ale můžete implementovat více rozhraní. OOP se snaží podobat tomu, jak jsou definovány objekty v reálném životě, a rozhraní jsou velmi logickým způsobem seskupování objektů z hlediska chování. Rozhraní je jakási smlouva mezi sebou a jakoukoli třídou, která je implementuje. Tato smlouva stanoví, že každá třída, která implementuje rozhraní, implementuje také jeho metody, vlastnosti, indexátory a událostí. Rozhraní neobsahuje žádnou implementaci, pouze podpisy funkčnosti. Je možné si uvažovat o rozhraní jako o abstraktní třídě s odstraněnou implementací.

Rozhraní založené na konceptech návrhu poskytuje volné propojení, komponentově založené programování, snadnější údržbu, zvyšuje škálovatelnost kódové základny a zpřehledňuje kód, jelikož implementace je oddělena od rozhraní.[1]

V kódu naší aplikace lze najít rozhraní ve složce Services. Složka ServicesImpl dědí ze Services a je v ní uložena funkčnost daných rozhraní. Předpokládáme, že uživatelé prostřednictvím uživatelských prvků nebudou volat DAO metody přímo, ale přes třídy v ServicesImpl. Je to užitečné hlavně, když se nějaká metoda musí volat společně s jinou. Například při mazání zákazníka je vždy potřeba smazat i osobu, na kterou ukazuje její id, a adresu, která je k osobě vázaná. Proto CustomerServiceImpl/deleteCustomer vypadá následovně:

```
public void deleteCustomer(CustomerDTO customerDTO)
        {
            log.info("CustomerServiceImpl: customer deletion.");
            customerDAO.delete(customerDTO.Id);
            personService.deletePerson(customerDTO);
            log.info("Customer deleted. Id: " + customerDTO.Id);
        }
```

Zatímco CustomerServiceImpl/deletePerson:

```
public void deletePerson(PersonDTO personDTO)
        {
            log.info("PersonServiceImpl: deleting person.");
            int id = personDTO.Id;
            personDAO.delete(id);
            addressDAO.delete(personDTO.Address);
            log.info("Person deleted. Id: " + personDTO.Id);
```

[1] C# Interfaces, What Are They and Why Use Them - DZone Web Dev. DZone [online] [cit. 06.03.2019]. Dostupné z: https://dzone.com/articles/c-interfaces-what-are-they-and

```
}
```

4.2 Frontend

4.2.1 Formuláře

Velice častými operacemi v systému bude zadávání např. nových objednávek a zákazníků, méně časté bude zadávání nových položek či zaměstnanců. V každým případě, uživatel systému často přijde do styku s vyplňováním formulářů. Pro ukázku popíšeme formulář pro zadávání nového zákazníka. (Proces vytváření nového zákazníka na backendu je popsán v kapitole 3.9.)

V novém formu je pomocí ovládacích prvků vytvořen formulář tak, aby jeho následné vyplnění vyžadovalo od uživatele zadání všech informací potřebných pro úspěšné uložení zákazníka do databáze.

Stisknutím potvrzovacího tlačítka se vytvoří instance CustomerDTO, instance AddressDTO, kterým jsou nastavené atributy obdržené ze vstupních políček (textboxů). Dále se provádí validace vstupních dat, tj. kontrola, zda povinná pole nezůstala prázdná nebo nebyly zadány nepovolené znaky. Až v případě, že je formulář vyplněn správně, přiřadí se k instanci CustomerDTO instance AddressDTO a zavoláním customerService/createCustomer se vytvoří nový zákazník.

Formuláře na vytvoření nového zaměstnance či nové položky vypadají podobně.

4.2.2 Zobrazování tabulek z databáze

Pro výpis obsahu databazových tabulek je použit DataGridView. Logika zobrazování tabulky zákazníků uživatelům v kódu:

```
public void reloadDataTable()
        {
            NpgsqlConnection connection = DatabaseConnection.createConnection();
            string sql = "SELECT c.id as customerId, p.name as jméno, p.surname as
příjmení, p.date_of_birth as narozen, " +
                         "p.phone_number as telefon, p.email_address as email,
c.notification as notifikace, " +
                         "a.street as ulice, a.number as číslo, a.city as město,
a.postal_code as psč, a.id as addressId " +
                         "FROM Person p " +
                         "JOIN Customer c ON c.id = p.id " + "JOIN Address a ON
a.id = p.address_id";
            dataAdapter = new NpgsqlDataAdapter(@sql, connection);
            DatabaseConnection.closeConnection(connection);

            dataTable = new DataTable();
            dataAdapter.Fill(dataTable);
            dataGridView1.DataSource = dataTable;
```

```
            // Columns settings
            dataGridView1.Columns[ID_CUSTOMER_COLUMN_NAME].ReadOnly = true;
            dataGridView1.Columns[ID_ADDRESS_COLUMN_NAME].Visible = false;
        }
```

Proto, aby byla výsledná tabulka responzivní, je jeho vlastnost AutoSizeColumnsMode nastavena na hodnotu fill.

Dále je umožněno záznamy z tabulky upravovat nebo vymazávat (pro uživatelé, který má na to oprávnění). Kromě toho, pro lepší přehlednost a úsporu času je zde umožněno vyhledávání zákazníků podle Příjmení.

4.2.3 Proces vytváření rezervace

Celkem rutinní práci byla implementace vytváření záznamů o zákaznicích, položkách a zaměstnancích. Těžkým úkolem je zprovoznit ukládání záznamu o rezervací, jelikož se spolu s rezervací ukládá hodně atributů, a jsou k ní vázáné konkrétní zákazníci a položky, čí atributy nová rezervace ovlivní. Je jasné, že formulář pro vytvoření rezervaci se na jeden form nevejde, proto bude se rezervace vytvářet ve více krocích, a konkrétně ve čtyřech. Všechny kroky se budou zobrazovat ve stejným formu ale každý v jiném User Controlu.

V prvním kroku (grafické rozhraní pro který je zobrazeno na obrázku 12) musí uživatel vybrat zákazníka, ve druhém si uživatel vybírá položku, ve třetím zadává termíny od kdy do kdy bude položka rezervovaná. Ve čtvrtém kroku se provádí kalkulace celkové ceny. Je tady definovaná metoda total_price, která má na vstupu všechny parametry potřebné pro výpočet ceny k úhradě. Vypadá následovně:

```
        public double totalPrice(double itemPrice, double insurance, double
otherCosts, double advance, double discount)
        {
            double total = itemPrice + insurance + otherCosts - advance - discount;
            return total;
         }
```

ItemPrice je cena položky vybrané ve druhém kroku, ostatní parametry jsou zadané na vstupu uživatelem (viz obrázek 14).

V případě, že uživatel vyplnil všechny povinné atributy a ukládá rezervaci, vytváří se nový objekt typu customerOrder s tím, že jeho atribut Created (datum vytvoření) je nastaven na hodnotu `DateTime.Today`, hodnota Total_price je vypočítaná pomocí metody total_price a ostatní atributy jsou převzaté ze vstupu.

4.2.4 Zobrazování rezervačního kalendáře

Jedním z hlavních důvodu, proč potřebujeme informační systém pro půjčovnu oděvů je ten, že potřebujeme vědět, kdy je která položka dostupná. Proto při zobrazování seznamu položek (viz obrázek 9) se zobrazí také tlačítko, které spustí nové okno (form) s informacemi o vybrané položce. V danem formu je uživateli zobrazen kalendář dostupnosti položky, který se nachází v knihovně ovládacích prvků Windows Presentation Foundation (WPF), tj. musíme uvést `using System.Windows.Controls.` Dále pro daný kalendář je potřeba vytvořit tzv. host typu `ElementHost`, který se nachází v knihovně `System.Windows.Forms.Integration`, použití které, musíme samozřejmě také označit. Metoda pro vytvoření kalendáře vypadá následovně:

```
public void showAvailableDates(Dictionary<DateTime, DateTime> unavailableDates)
        {
            // Create host element for system.windows.controls calendar.
            ElementHost host = new ElementHost();
            host.Dock = DockStyle.Fill;

            Calendar calendar = new Calendar();
            foreach (KeyValuePair<DateTime, DateTime> entry in unavailableDates)
            {
                calendar.BlackoutDates.Add(
                    new CalendarDateRange(entry.Key, entry.Value));
            }
            host.Child = calendar;
            itemInfoForm.Controls.Add(host);
            host.Size = new Size(250, 500);
            host.AutoSize = true;
        }
```

Jak můžeme vidět, jako parametr metoda přijímá časový interval, který můžeme získat pomocí metody customerOrderDAO/getUnavailableDates.

CustomerOrderDAO/getUnavailableDates za použití NpgsqlDataReader vyhledává v tabulce customerOrder termíny začátku a konce rezervace položky, kterou přijala na vstupu jako parametr.

4.3 Testování

4.3.1 Unit testing

Dostáváme se k poslední fázi vývoje, kterou se v dané práci zabývám, a to k testování. Systém je testován prostřednictvím automatického testování – unit testingu. Víc informací o unit testech je uvedeno v kapitole 3.2.3.

Testování by nemělo ovlivnit běh programu, proto se unit testy vytváří do nového projektu. Po tom, co je Unit Test Project vytvořen, je k němu přidaná reference neboli odkaz na projekt RentApp. Po propojení je možné začít systém testovat.

Společně s každým z testů se budou používat dvě metody: `[TestInitialize] a [TestCleanup]`. Metoda `[TestInitialize]` slouží k přípravě prostředí před každým testem. To znamená, že se před každým spuštěním testu spustí tato metoda. Hlavním účelem použití této metody je docílit toho, aby se testy navzájem neovlivňovaly. Metoda `[TestCleanup]` se vola po provedení každého testu a slouží k vyčištění prostředí po předchozím testu.

Nyní si otestujeme, jak fungují DAO metod adres. V první řádě je přepsaná metoda Equals tak, aby srovnávala všechny atributy adres, které jsou porovnávané.

```
public override bool Equals(object obj)
        {
            if (obj == null)
            {
                return false;
            } else
            {
                AddressDTO addressToCompare = (AddressDTO) obj;
                if (this.City.Equals(addressToCompare.City) &&
                    this.Street.Equals(addressToCompare.Street) &&
                    this.Number.Equals(addressToCompare.Number) &&
                    this.Postal_code.Equals(addressToCompare.Postal_code))
                {
                    return true;
                } else
                {
                    return false;
                }
            }

        }
```

Teď za pomocí přepsané metody můžeme otestovat vytváření adresy.

```
public void CreateAddressTest()
        {
            AddressDTO addressDTO = new AddressDTO();
            addressDTO.Street = "Jílovská";
            addressDTO.Number = "231";
            addressDTO.City = "Praha";
            addressDTO.Postal_code = "142 00";
            AddressDTO createdAddress = addressDAO.create(addressDTO);

            addressDTO.Equals(createdAddress);
        }
```

Podobným způsobem je testována aktualizace. Nejdřív je vytvořen jeden záznam adresy, zatím jinému záznamu je nastavené id prvního a také atributy, které jsou od atributů prvního záznamu odlišné. Následně je volána metoda AddressDTO/update s druhým

záznamem v parametru a výsledek je prostřednictvím metody Equals srovnám s prvním záznamem.

Zkusíme také otestovat, jestli se adresy mažou, tj. metodu AddressDTO/delete. Je to jednoduchý test. Nejdřív záznam o adrese vytvoříme, potom ho smažeme. Dále se pokusíme vyhledat danou adresu v tabulce a s její hodnotami vytvořit nový objekt typu AddressDTO. Zkontrolujeme, že vytvořený objekt typu AddressDTO je prázdný (pomocí Assert.IsNull()). Všechny testy skončily úspěšně.

4.3.1 Uživatelské testování

Jelikož je aplikace funkční, můžeme provést uživatelské testování. Toto testování spočívá v tom, že vybereme osobu či více osob z cílové skupiny uživatelů a pozorujeme jejich chování během toho, jak pracují s aplikaci. Daná metodika je výhodná v tom, že uživatel může odhalit chyby a nesrovnalosti, které unikli vývojářům. Další výhodou je to, že to, co se vývojářům zdá jasné a intuitivní nemusí být zřejmé i pro ostatní.

Na štěstí, mezi příslušníky mé rodiny se našli lidí, které můžeme považovat za typické uživatelé vytvářeného informačního systému. Proto jsem vytvořila seznam případů užití, které je potřeba otestovat a pozorovala uživatele, který je plnil. Bylo otestováno hlavně ukládání dat a vyhledávání informací, konkrétně:

- Ukládání, aktualizace, mazaní a vyhledávání zákazníka.
- Ukládání, aktualizace, mazaní a vyhledávání zaměstnance.
- Ukládání, aktualizace a vyhledávání položky a termínu její dostupnosti.
- Vytváření, aktualizace, mazaní a vyhledávání rezervace.

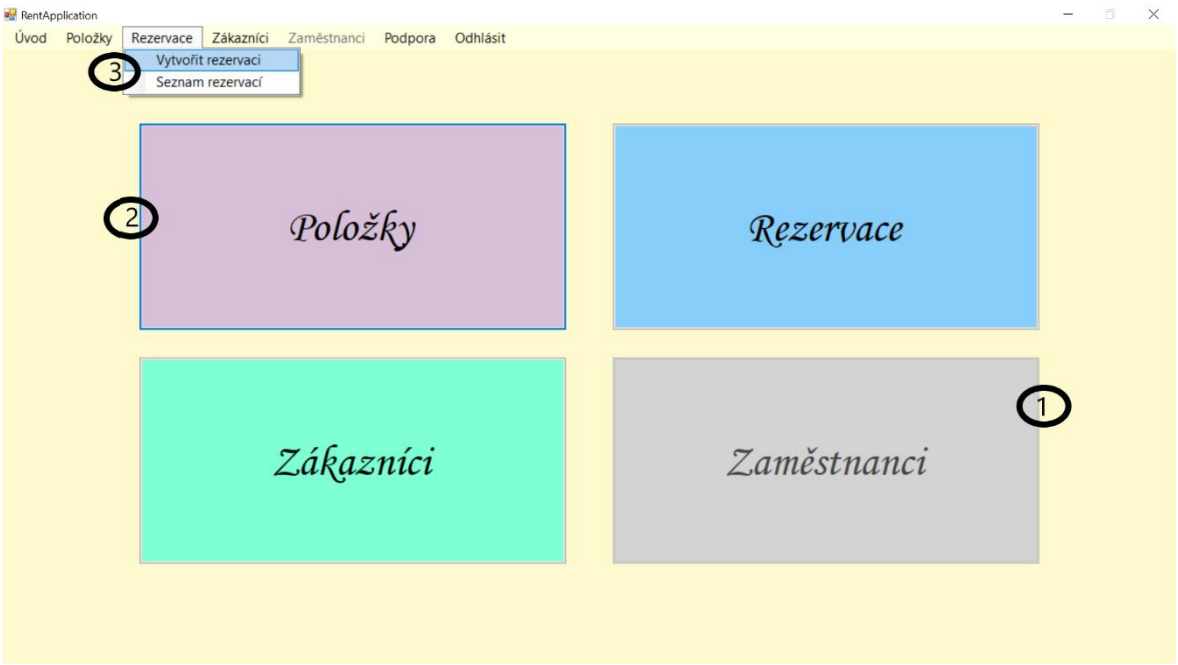
Testování proběhlo v pořádku bez větších problémů. Za menší nedostatek uživatel považuje to, že při zadávání nové objednávky je potřeba potvrzovat výběr zákazníka a položky. Po krátké debatě jsme se shodli na tom, že se na to v případě častějšího použití systému, dá zvyknout. Celkovým výsledkem daného testování bylo pozitivní hodnocení systému a závěr, že není potřeba aplikaci žádným způsobem předělávat.

4.4 Uživatelská příručka

Po spuštění aplikace se v první řádě zobrazí přihlašovací okno, kam uživatel zadá svoje uživatelské jméno a heslo. Po úspěšném přihlášení se načítá úvodní obrazovka s ovládacími prvky dostupnými podle role přihlášeného uživatele. Přihlásíme se pod jménem „editor“

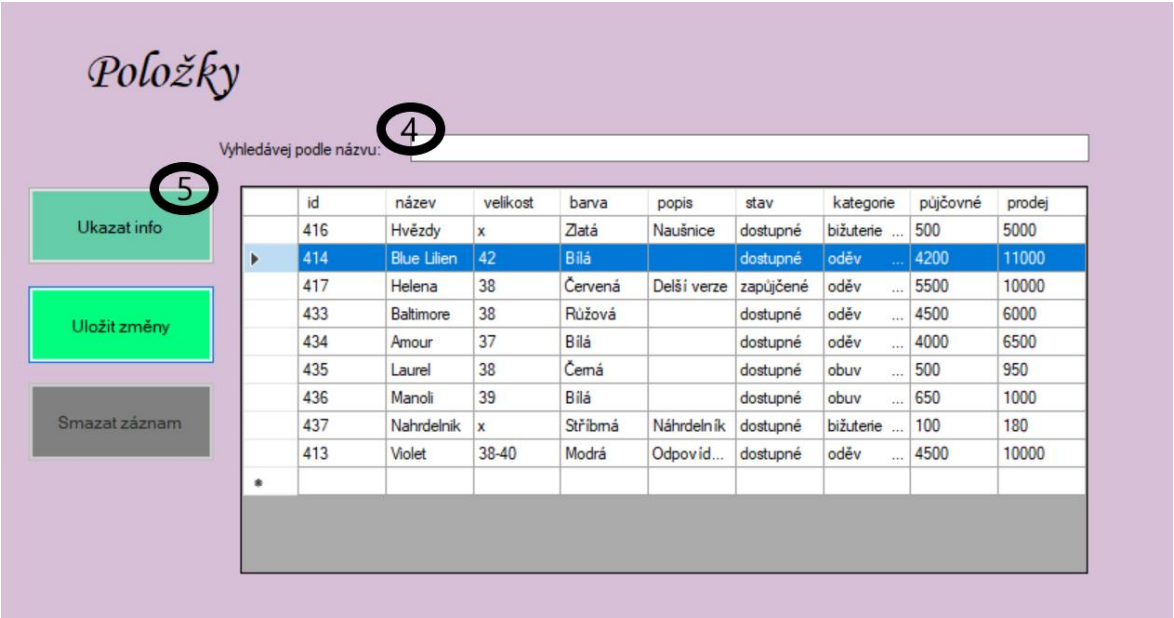
a heslem „editor“ a podíváme se na aplikaci z hlediska uživatele, který disponuje jen přístupovými právy role Editor.

Obrázek 8 Úvodní obrazovka, grafické rozhraní



Velká tlačítka – Položky, Objednávky, Zákazníci, složí pro přesměrování uživatele na seznam příslušných entit. Tlačítko Zaměstnanci, označené číslem 1, je aktivní pouze pro uživatele s oprávněním spravovat zaměstnance, tj. pro roli authority. Stejné platí i pro tlačítko Zaměstanci v horním menu. Podíváme se podrobněji na seznam položek, ke kterému se lze dostat buď pomocí velkého tlačítka Položky, označeného číslem 2, nebo přes horní menu Položky / Seznam položek.

Obrázek 9 Seznam položek, grafické rozhraní

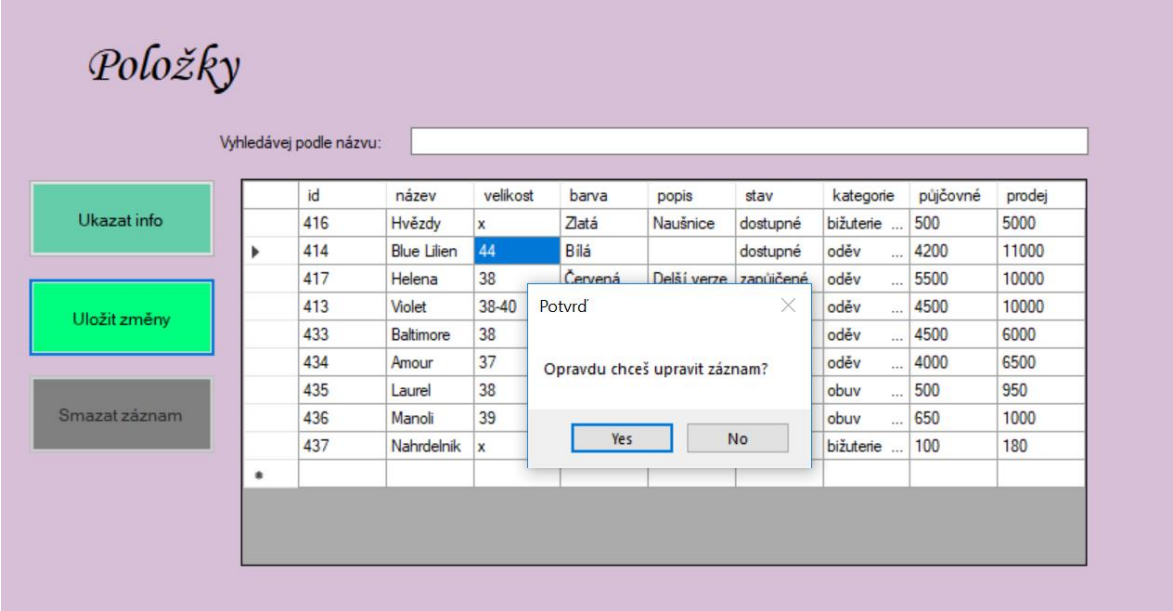


V novém okně se načte obrazovka se seznamem položek. Tabulka se naplní záznamy z databáze pomocí dataAdapteru (podrobně v kapitole 3.12). Vzhledem k tomu, že by

v budoucnu mohla půjčovna disponovat velkým množstvím položek, výhodou je umožnění vyhledávání položky podle názvu. Pro tento účet slouží políčko označené číslem 4.

V případě, kdyby byla potřeba nějaký záznam upravit, je ho možné upravovat přímo v tabulce. Aktualizace je potvrzena kliknutím na tlačítko Uložit změny a následným potvrzením v dalším okně.

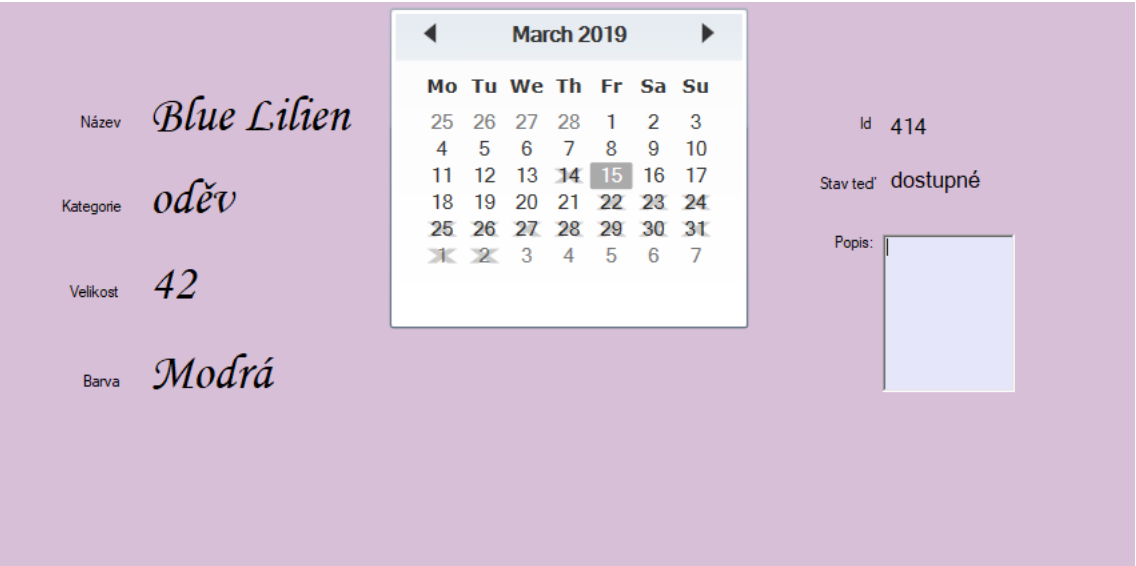
Obrázek 10 Potvrzovací okno, grafické rozhraní



Je možné editovat hodnoty ve všech sloupcích kromě id. Podobné potvrzovací okno se objeví i v případě mazání záznamu pomocí tlačítka Smazat záznam, které pro uživatele s roli editor není aktivní.

Důležitou funkčností je zobrazování termínů dostupnosti položky. Pro přehled informaci o položce, a hlavně pro zjištění všech termínů, kdy není zapůjčena, složí tlačítko Ukazat info, označené číslem 5. Informace o vybrané položce se zobrazí v novém okně.

Obrázek 11 Informace o položce, grafické rozhraní

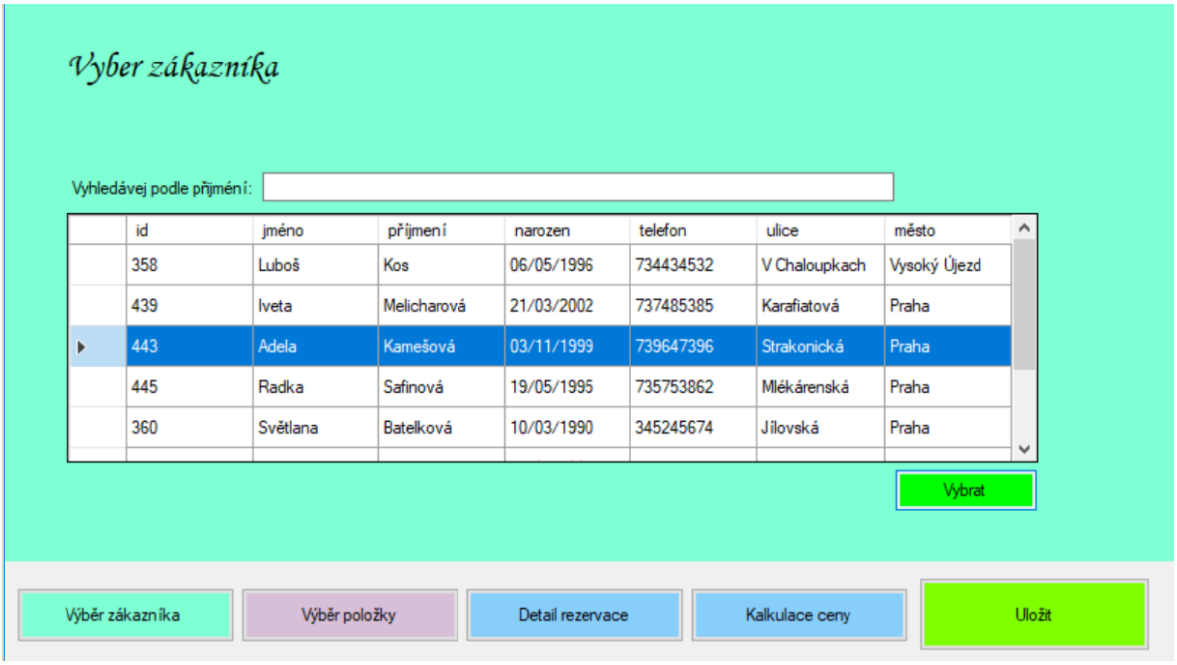


Každá objednávka je provázaná s položkou, takže jakmile je nějaká položka rezervovaná, je daný termín položky označen jako nedostupný. To ale neznamená, že ve dnech, které nejsou

přeškrtnuté, je položka k zapůjčení. Je potřeba počítat s dalšími stavy, např. položka může být po vrácení ještě několik dni v čistírně.

Zajímavé je zadávání rezervací, které se spouští tlačítkem v menu – Vytvořit rezervaci, které je označené číslem 2.

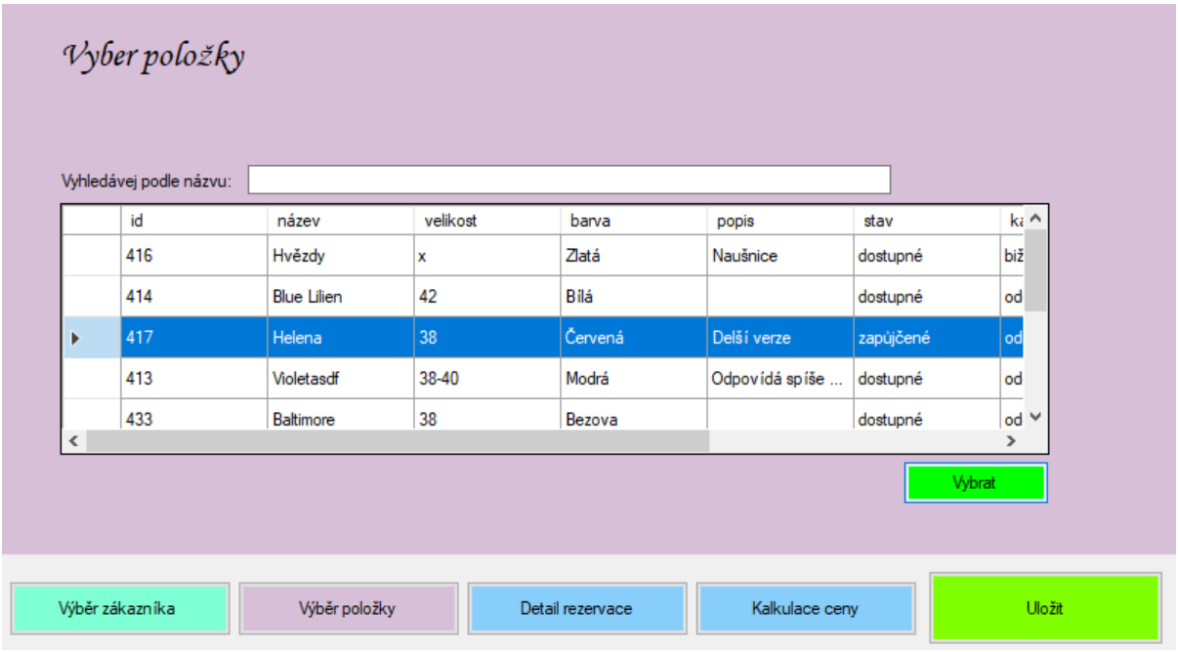
Obrázek 12 Rezervace, výběr zákazníka, grafické rozhraní



Vytvoření rezervace se skládá ze čtyř kroků, každý z kterých odpovídá jednomu ze čtyř tlačítek v dolní části obrazovky. Prvním krokem je výběr zákazníka, pro kterého je objednávka vytvářena. Důležité je označit vybraného zákazníka a potvrdit výběr tlačítkem Vybrat, které se nachází v pravé dolní části obrazovky. Je zde také umožněno vyhledávat

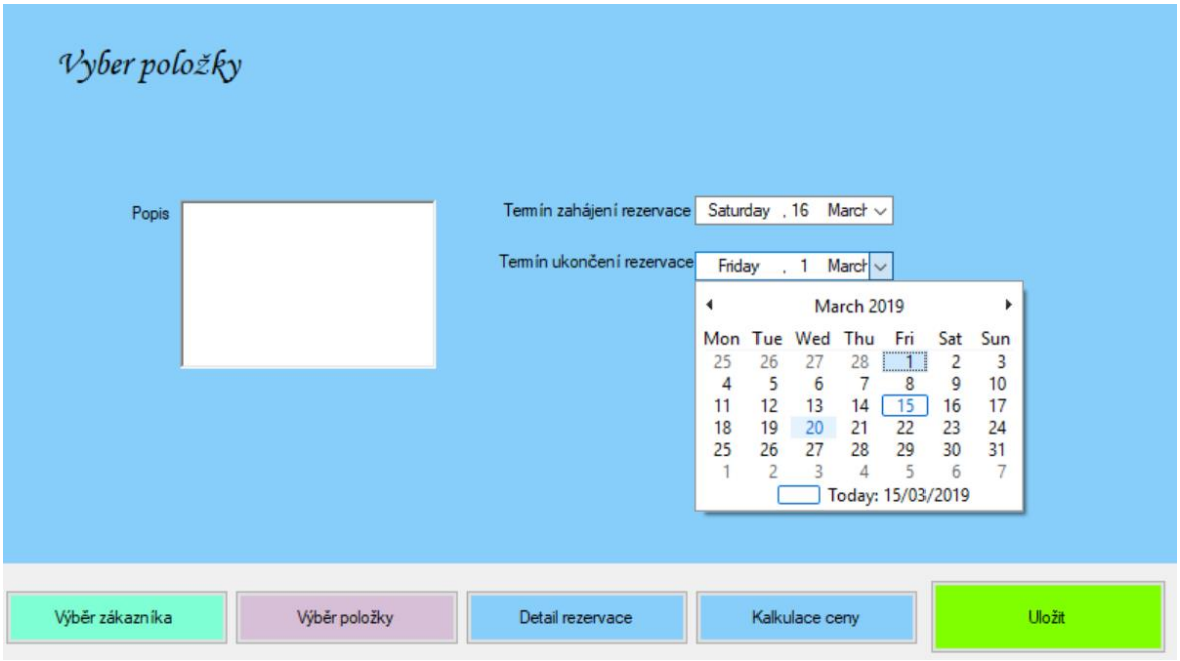
zákazníka podle příjmení. Druhým krokem je výběr položky, který funguje na stejným principu s tím rozdílem, že se položky vyhledávají podle názvu.

Obrázek 13 Rezervace, výběr položky, grafické rozhraní



Ve třetím kroku se zadávají detaily rezervace.

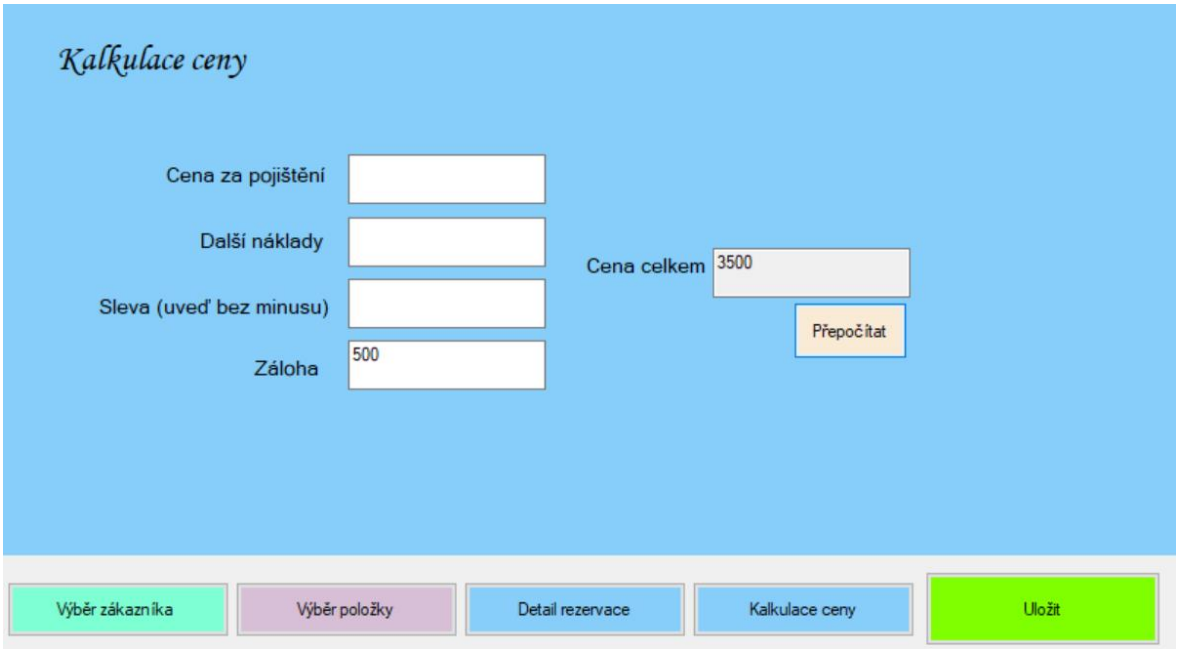
Obrázek 14 Rezervace, detail, grafické rozhraní



Tady je důležité správně zadat termíny zahájení a ukončení rezervace, jelikož se v těchto termínech bude vybraná položka zobrazená jako nedostupná. Popis rezervace, tj. jakékoliv doplňující informace, samozřejmě není nutné vyplňovat.

Posledním krokem je kalkulace ceny.

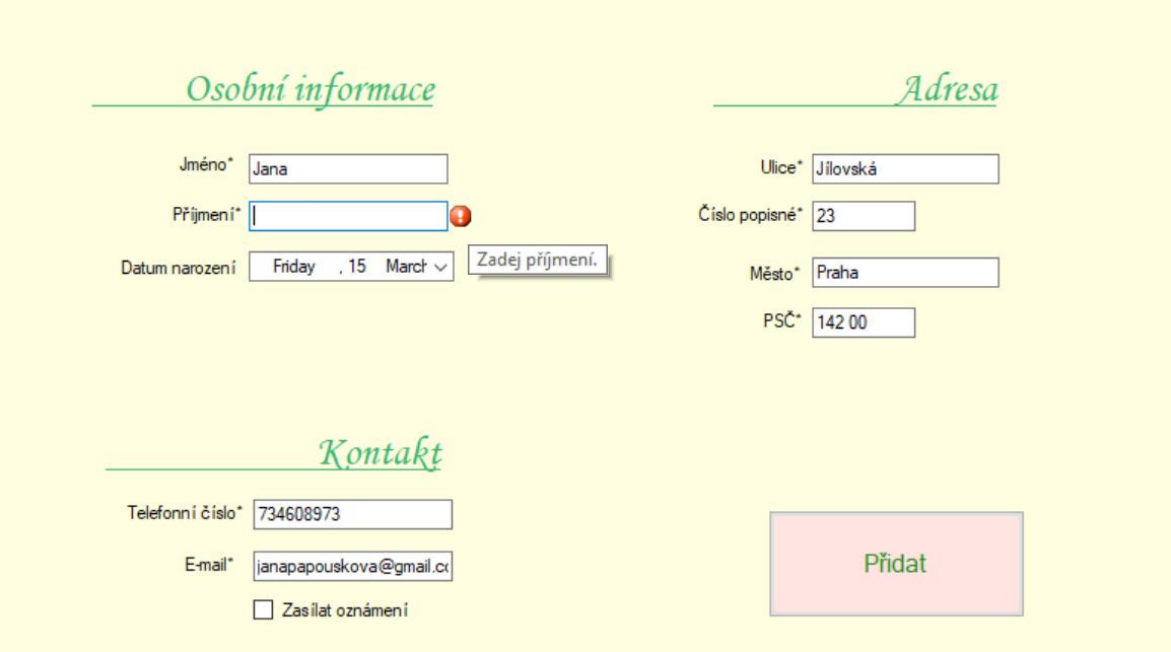
Obrázek 15 Rezervace, kalkulace ceny, grafické rozhraní



V daném kroku se zadávají všechny náklady a slevy související s rezervací. Zmačknutím tlačítka Přepočítat dojde k automatickému vyhodnocení výsledné ceny. Zadávání žádné z položek není povinné. V případě, že všechna políčka zůstanou prázdná, cena celkem se bude pochopitelně rovnat ceně rezervace položky vybrané v kroku 2 – Výběr položky.

Nakonec se podíváme na vyplňování formuláře při vytváření záznamu o novém zákazníkovi, který spustíme pomocí tlačítka v menu Zákazníci/Přidat zákazníka a který vypadá následovně:

Obrázek 16 Formulář na vyplnění info o zákazníkovi



Povinná políčka jsou označená *. V případě, že nějaké z povinných políček zůstane nevyplněné, při stisknutí tlačítka Přidat, se nový záznam nevytvoří, ale objeví se varování jako na obrázku u políčka Příjmení*.

5. Výsledky a diskuse

Hlavním cílem mé bakalářské práce bylo navrhnout, implementovat a otestovat podnikový informační systém pro půjčovnu oděvů. Výsledkem mého snažení se stal funkční software a práce, která slouží jako jeho dokumentace a průvodce vytváření informačního systému. Bohužel ne všechny funkčností, které vyplynuly z analýzy požadavků byly implementovány. Hlavním důvodem je to, že podobné informační systémy jsou vyvíjené v týmech, jelikož jsou náročnější než například webové stránky, na kterých pracuje většinou jeden člověk, proto během vývoje byl pociťován nedostatek zdrojů a nebyla implementovaná veškerá funkčnost.

Kdyby půjčovna oděvů, pro kterou byl systém vyvíjen, se v brzké době otevřela, v první řádě by se změnilo to, že se data neposílají na server, ale ukládají se na počítač (databáze je uložena na localhostu). Jelikož takové ukládání dat není bezpečné, data by se po nasazení aplikace posílala na server, což by umožnilo běh daného systému na více počítačích.

Další nevýhodou je to, že jsem umožnila uživatelům mazat rezervace. Původní návrh byl, v případě, že se rezervace zruší, měnit stav dané rezervace a uvolňovat položku vázánou na ni v daném termínu. Místo této funkčnosti byla implementována funkčnost jednodušší, kdy se při zrušení rezervace celá rezervace maže a položka se tak automaticky uvolňuje.

Za úspěch považuji hlavně to, že i přes četné množství chyb spojených s nejsložitější fázi – implementací, podařilo se mi jich spoustu vyřešit a vytvořit funkční informační systém.

6. Závěr

Jako téma své bakalářské práce jsem si zvolila celý proces vývoje informačního systému pro půjčovnu oděvů. V průběhu práce byla realizovaná analýza požadavků na informační systém, dále byl navržen datový model, pak následovala nejnáročnější a nejdelší fáze – implementace. Po tom, co byl systém funkční, byl otestován a následně byla vytvořená uživatelská příručka k systému.

Hlavním důvodem volby daného tématu se stal fakt, že zrovna informační systém pro půjčovnu oděvů bych mohla potřebovat pro svůj budoucí podnik. I v případě, že si zakládání firmy rozmyslím, tato práce se stala pro mě přínosnou v mnoha ohledech. Především mě přiměla prohloubit své znalosti světa informačních technologií, zejména v oblasti softwarového inženýrství, čímž mě obohatila cennými zkušenostmi, které mi poslouží v mém profesním životě.

Seznam použitých zdrojů

1. Database Models in DBMS | Studytonight. Studytonight – Best place to Learn Coding Online [online]. © 2019.
   Dostupné z: https://www.studytonight.com/dbms/database-model.php
2. Disciplína sběr a analýza požadavků [online]. © [cit. 2019-03-06].
   Dostupné z: http://lucie.zolta.cz/index.php/softwarove-inzenyrstvi/150-disciplina-sber-a-analyza-pozadavku
3. Enterprise Resource Planning (ERP) Definition. Investopedia - Sharper Insight. Smarter Investing. [online]. © [cit. 2019-03-06].
   Dostupné z: https://www.investopedia.com/terms/e/erp.asp
4. Funkční a nefunkční testy | Testování softwaru. Testování softwaru [online]. © [cit. 2019-03-06]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/funkcni-a-nefunkcni-testy/
5. Integrační Testování | Testování softwaru. Testování softwaru [online]. © [cit. 2019-03-06]. Dostupné z: http://testovanisoftwaru.cz/tag/integracni-testovani/
6. Introduction to the C# Language and the .NET Framework | Microsoft Docs. [online]. © [cit. 2019-03-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework
7. Metody vývoje aplikací. Waterfall, V-model, Inkrementální model. iQuest blog [online]. © [cit. 2019-03-06].
   Dostupné z: http://blog.iquest.cz/2017/07/metody-vyvoje-aplikaci-waterfall-v.html
8. NAKOV, Svetlin et col. Fundamentals of computer programming with C#: the Bulgarian C# Programming Book. Sofia, 2013. ISBN 978-954-400-773-7
9. Podnikový proces (Business process) - ManagementMania.com. [online]. © 2011.
   Dostupné z: https://managementmania.com/cs/business-process-podnikovy-proces
10. Regression Testing Complete Guide: Tools, Method, and Example. Software Testing Help – A Must Visit Software Testing Portal[online]. © 2006.
    Dostupné z: https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/
11. Rizeni-kvality-softwaru. soom.cz. [online]. © 2019. Dostupné z: https://www.soom.cz/clanky/1176--PTWA-Rizeni-kvality-softwaru

12. SDLC (Software Development Life Cycle) Tutorial: What is, Phases, Model. [online]. © [cit. 12.03.2019]. Dostupné z: https://www.guru99.com/software-development-life-cycle-tutorial.html
13. Software Testing Help - A Must Visit Software Testing Portal [online]. © [cit. 2019-03-06]. Dostupné z: https://www.softwaretestinghelp.com/system-testing
14. SQLCourse - Lesson 1: What is SQL?. SQLCourse - Interactive Online SQL Training for Beginners [online]. Dostupné z: http://www.sqlcourse.com/intro.html
15. Systém | Andromedia.cz. Andromedia.cz | Sdílením informací ke konkurenční výhodě [online]. © [cit. 2019-03-06].
Dostupné z: http://www.andromedia.cz/andragogicky-slovnik/system
16. TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy
17. Types of Software Testing: Different Testing Types with Details. Software Testing Help – A Must Visit Software Testing Portal [online]. © 2006.
Dostupné z: https://www.softwaretestinghelp.com/types-of-software-testing/
18. Types of Software Testing: Different Testing Types with Details. Software Testing Help – A Must Visit Software Testing Portal[online]. © 2006.
Dostupné z: https://www.softwaretestinghelp.com/types-of-software-testing/
19. What is Business Process Modeling Notation | Lucidchart. Online Diagram Software & Visual Solution | Lucidchart [online]. © [cit. 11.03.2019].
Dostupné z: https://www.lucidchart.com/pages/bpmn
20. What is Integration Testing (Tutorial with Integration Testing Example). Software Testing Help – A Must Visit Software Testing Portal [online]. © 2006.
Dostupné z: https://www.softwaretestinghelp.com/what-is-integration-testing/
21. What is Integration Testing (Tutorial with Integration Testing Example). Software Testing Help – A Must Visit Software Testing Portal [online]. © 2006.
Dostupné z: https://www.softwaretestinghelp.com/what-is-integration-testing/
22. What is Microsoft Windows? definition and meaning - BusinessDictionary.com. Online Business Dictionary - BusinessDictionary.com [online]. © 2019.
Dostupné z: http://www.businessdictionary.com/definition/Microsoft-Windows.html

23. What is MySQL? What is a Database? What is SQL? (thesitewizard.com). thesitewizard.com: Website design, promotion, CGI, PHP, JavaScript scripting, and revenue earning. [online]. © 2010. Dostupné z: https://www.thesitewizard.com/faqs/what-is-mysql-database.shtml
24. What is Unified Modeling Language (UML)?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration [online]. © [cit. 2019-03-06]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
25. Windows Forms Overview | Microsoft Docs. [online]. © [cit. 2019-03-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview

7. Přílohy

Příloha 1 Procesy v podniku