ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vachula** Jméno: **Richard** Osobní číslo: **434738**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Systém pro podporu skladového hospodářství

Název diplomové práce anglicky:

Warehouse management system

Pokyny pro vypracování:

Cílem práce je vytvoření systému pro podporu skladového hospodářství pro malý podnik.
Seznamte se s dostupnými řešeními a proveďte jejich analýzu.
Na základě provedené analýzy specifikujte konkrétní funkční a nefunkční požadavky na navrhovaný systém s ohledem na požadavky daného podniku. Systém bude realizovat správu skladu a skladových zásob a automatizaci jejich naskladnění.
Na základě těchto požadavků navrhněte aplikaci. Pro analýzu a návrh použijte vhodné prostředky softwarového inženýrství.
Navrženou aplikaci implementujte a otestujte včetně testů použitelnosti.

Seznam doporučené literatury:

1. Ian Sommerville: Software Engineering, Global Edition, Pearson Higher Ed, 2016, ISBN1292096144
2. Arlow, J., Neustat, I.: UML 2 a unifikovaný proces vývoje aplikací. Computer Press, 2007

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Božena Mannová, Ph.D., kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **06.02.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Božena Mannová, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

České vysoké učení technické v Prahe
Fakulta elektrotechnická
Katedra počítačů

Diplomová práca

Systém pro podporu skladového hospodářství

Bc. Richard Vachula

Vedúci práce: Ing. Božena Mannová, Ph.D.

Študijný program: Otevřená informatika, magisterský

Odbor: Softwarové inženýrství

23. mája 2019

Poďakovanie

Ďakujem Ing. Božene Mannovej, Ph.D., za konzultácie a pravidelné dávky motivácie, ktoré mi pomohli pri vypracovaní mojej diplomovej práce.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe dňa 22. 5. 2019 .....

Abstract

Bc. VACHULA, Richard: Warehouse management system. [Master's Thesis] - Czech Technical University in Prague. Faculty of Electrical Engineering, Department of computers. Supervisor: Ing. Božena Mannová, Ph.D.

Aim of the thesis is to design and implement a web application as a warehouse management system according to the requirements of a company the system is built for. Besides providing greater clarity of used resources, system also simplifies the process of stocking by parsing contents of provided pdf invoices. Focus has also been placed on verification of the proper implementation of the system by means of unit and usability testing.

Keywords: warehouse, warehouse management, web application, OCR, testing

Abstrakt

Bc. VACHULA, Richard: Systém pro podporu skladového hospodářství. [Diplomová práca] - České vysoké učení technické v Praze. Fakulta elektrotechnická, Katedra počítačů. Vedúci: Ing. Božena Mannová, Ph.D.

Práca sa zaoberá návrhom a implementáciou na mieru šitého systému pre podporu správy skladového hospodárstva malého podniku s ohľadom na požiadavky konkrétneho zadávateľa. Systém zprehľadní správu skladových zásob a zjednoduší proces naskladnenia parsovaním dát z elektronickej podoby faktúr. Za cieľ si práca súčasne kľadie i overenie funkčnosti systému programovými testami i testami použiteľnosti.

Kľúčové slová: sklad, skladové hospodárstvo, webová aplikácia, OCR, testovanie

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

1.1 Motivácia práce

V roku 2016 sa autorovi naskytla príležitosť stať sa členom technického tímu, podieľajúceho sa na vývoji fungujúcej služby Uniqway. Chopením sa tejto príležitosti autor spoznal časť tímu zaoberajúcu sa tvorbou jej hardvérových riešení. Táto časť tímu je súčasne i menšou firmou so zakázkami v Českej republike i zahraničí.

Pre potreby evidencie skladových zásob a ich prehľadnejšieho spravovania využívajú vlastný systém, ktorý je však po stránke funkcionality značne obmedzený a s narastajúcimi požiadavkami už nedostačujúci. Taktiež spôsob implementácie robí tento systém náročný na jeho prípadné rozšírenie.

Správa pracovníkov, projektov a ich výstupov v podobe vyhotovených zariadení, prehľad pohybov v sklade, generovanie QR kódov pre urýchlenie vyskladnenia konkrétnych súčiastok či naskladnenie s využitím parsovania pdf faktúr sú jedny z mnohých funkcionalít, ktoré súčasný systém postráda, a ktorými má jeho nová podoba disponovať.

Vzhľadom na predchádzajúce skúsenosti s vývojom a naskytnutou možnosťou v podobe vyhotovenia nového systému v rámci svojej diplomovej práce, bol autor v tejto veci firmou oslovený a túto úlohu sa rozhodol prijať.

1.2 Cieľ práce

Cieľom tejto práce je návrh a implementácia na mieru zostaveného systému pre podporu skladového hospodárstva pre malý podnik s konkrétnym zadávateľom. Využité pri tom budú prostriedky softvérového inžinierstva. Výsledný systém bude realizovaný ako responzívna webová aplikácia. Súčasťou práce je taktiež overenie funkčnosti vhodným testovaním.

1.3 Popis kapitol

Práca je rozdelená do šiestich kapitol. V prvej sú zmienené motivácia práce a jej stanovený cieľ.

Obsahom druhej je stručná analýza zadávateľom aktuálne používaného riešenia, nasledovaná analýzou riešenia nového. Tá pozostáva z vydefinovania základných požiadaviek na systém, ktoré sú ďalej pretavené do aktérov systému a k nim prislúchajúcich prípadov použiteľnosti. Analýza obsahuje i Business Domain Model diagram entít, ktoré v systéme vystupujú spolu s ich vzťahmi.

Kapitola venovaná návrhu obsahuje popis zvolenej viacvrstvej architektúry systému. Súčasťou je i porovnanie relačných a nerelačných databáz, a popis zvoleného typu pre uvažovanú aplikáciu. Autor v rámci kapitoly pokračuje predstavením návrhu užívateľského rozhrania. Zmienené sú základné princípy jeho tvorby, farebná schéma či rozloženie základných grafických elementov.

V poradí ďalšej kapitole Implementácia autor popisuje dátový model a technológie použité pri vyhotovení výslednej aplikácie spolu s odôvodnením ich voľby. Podrobnejšie je rozobraný postup implementácie parsovania dát z pdf faktúr ako jedného z hlavných požiadavkov na systém. Kapitola je zavŕšená popisom implementácie navrhnutého užívateľského rozhrania.

Nasledujúcou je kapitola venovaná testovaniu. V rámci nej sú zmienené jednotkové a integračné typy testov použité pre overenie funkčnosti implementovaného systému, rovnako tak i testovanie použiteľnosti.

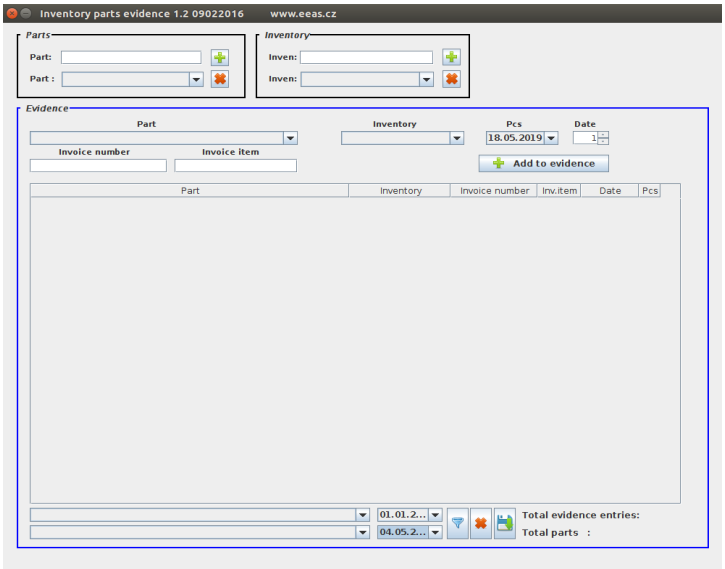
V závere sú obsiahnuté zhodnotenie práce spolu s možnosťami ďalšieho vývoja.

Kapitola 2

Analýza

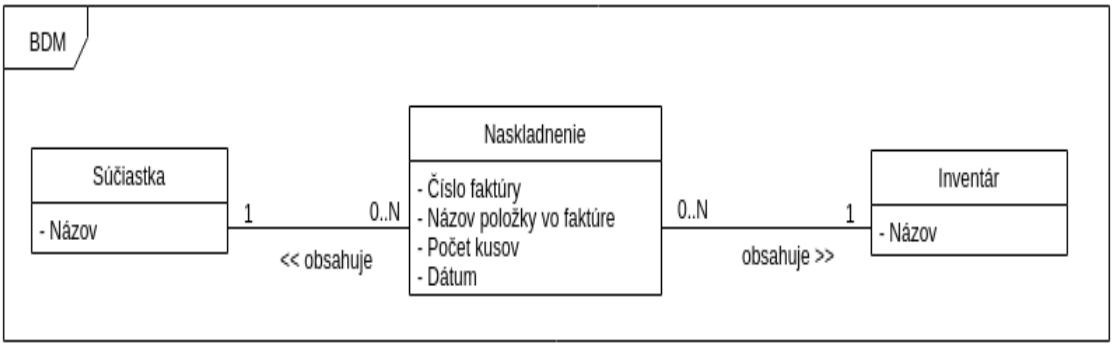
2.1 As-is

Jedným z prvých krokov pred vydefinovaním požiadavkov na finálny systém bolo nazretie na zadávateľom aktuálne používané riešenie. Je ním jednoduchá desktopová aplikácia s prívetivým a intuitívnym grafickým rozhraním implementovaná priamo zadávateľom, teda jej koncovým užívateľom. Nasadená je na jednom z interných počítačov a pre fungovanie nevyužíva pripojenie ku internetovej sieti. Po technologickej stránke je aplikácia implemnetovaná v jazyku Java. Grafické rozhranie bolo vytvorené použitím Swing a Abstract Window Toolkit (AWT) . V oboch prípadoch ide o toolkity pre tvorbu platformne nezávislých GUI práve v Jave. Pre perzistentné ukladanie dát je použitá SQLite databáza.



Obr. 2.1: Náhľad aktuálne používanej aplikácie

Hlavnými a súčasne jedinými entitami vystupujúcimi v systéme sú *Súčiastka*, *Inventár* a *Naskladnenie*. Súčiastka predstavuje naskladňovaný materiál a jej jediným atribútom je unikátny názov. Inventár je miestom pre fyzické umiestnenie naskladnených súčiastok. Jeho jediným atribútom je unikátny názov. Naskladnenie je záznamom o pridaní nového materiálu do skladu. Okrem voľby súčiastky a invetára umožňuje zadanie čísla faktúry, čísla položky na faktúre odpovedajúceho naskladňovanej súčiastky, jej počet a dátum. Vzťah týchto entít je zachytený Business Domain Model diagramom na obrázku 2.2.



Obr. 2.2: BDM model entít aktuálneho riešenia

Celková funkcionalita súčasného riešenia je značne obmedzená a zachytiť ju možno výčtom nasledujúcich požiadaviek:

- vytvorenie a zmazanie súčiastky,
- vytvorenie a zmazanie inventára,
- úprava počtu kusov súčiastky,
- zaevidovanie určitého počtu danej súčiastky do inventára,
- filtrovanie súčiastok zaevidovaných v určitom dátumovom rozmedzí,
- export výsledku filtrovania súčiastok do .csv súboru.

Tento zoznam poslúžil ako základ pre vydefinovanie požadovanej funkcionality nového systému, ktorému je venovaná časť 2.2.

Je treba zmieniť, že súčasná aplikácia - i keď funkčná, je svojou nie prílíš prehľadnou implementáciou a použitými technológiami náročná na doplnenie o novopožadované rozšírenia.

2.2 Nové riešenie

Nárast nových požiadaviek na zadávateľom používaný systém vyústil v zrod myšlienky pre vyhotovenie systému nového. Výstupom viacerých konzultácií práve so zadávateľom bol zoznam týchto požiadaviek s ich vecným popisom a prioritou.

Nasledujúce podsekcie zachytávajú analýzu nového riešenia počínajúc daným zoznamom pozostávajúcim z požiadaviek funkčných a nefunkčných. Z nich sú následne odvodení aktéri vystupujúci v systéme a k nim pridružené prípady užitia. V neposlednom rade je systém zachytený i z pohľadu Business Domain Model (BDM) diagramu, obsahujúcim jednotlivé entity spolu so vzťahmi medzi nimi, ktorý neskôr poslúžil pri návrhu databázového modelu.

2.2.1 Funkčné požiadavky

Požiadavky možno podľa Neudstata a Arlowa[27] definovať ako špecifikáciu toho, čo by malo byť implementované. Funkčné požiadavky pritom určujú, aké správanie bude systém poskytovať. Vzhľadom na skutočnosť, že Unified Modeling Language (UML) - jednotný jazyk pre tvorbu diagramov, nedefinuje jednoznačnosť v spôsobe značenia požiadaviek, tak ich značenie v našom prípade vychádza práve z [27]. Priradenie priority sa opiera o MoSCoW prioritizačnú metódu[24], kedy každá z požiadaviek je označená jedným zo začiatočných písmeniek anglických spojení Must have (M), Should have (S), Could have (C) a Won't have (W).

FR ID	FR Popis	Priorita
001	Systém umožní správe skladu vytvorenie nového skladu.	M
002	Systém umožní správe skladu editáciu už vytvoreného skladu.	M
003	Systém umožní správe skladu zobrazenie zoznamu všetkých vytvorených skladov.	M
004	Systém umožní správe skladu detailnejší náhľad na konkétny sklad.	C
005	Systém umožní správe skladu export aktálneho stavu vo formáte .csv.	C
006	Systém umožní správe skladu export stavu skladu v konkrétnom dni vo formáte .csv.	C
007	Systém umožní správe skladu vytvorenie nového aliasu.	M
008	Systém umožní správe skladu úpravu už existujúceho aliasu.	M
009	Systém umožní správe skladu zobrazenie zoznamu všetkých aliasov.	M

010	Systém umožní správe skladu vytvorenie novej súčiastky.	M
011	Systém umožní správe skladu úpravu už existujúcej súčiastky.	M
012	Systém umožní správe skladu zobrazenie zoznamu všetkých súčiastok.	M
013	Systém umožní správe skladu vytvorenie nového kusovníka.	C
014	Systém umožní správe skladu import nového kusovníka.	C
015	Systém umožní správe skladu úpravu už existujúceho kusovníka.	C
016	Systém umožní správe skladu zobrazenie zoznamu všetkých kusovník v systéme.	M
017	Systém umožní správe skladu náhľad detailu konkrétneho kusovníka.	W
018	Systém umožní správe skladu zistenie celkovej hodnoty skladu ku danému dňu.	C
019	Systém umožní správe skladu naskladnenie súčiastok.	M
020	Systém umožní bežnému užívateľovi vyskladnenie súčiastok.	M
021	Systém automaticky vytvorí balíček pre každú položku príjemky.	M
022	Systém umožní správe skladu rozdelenie balíčkov.	W
023	Systém umožní bežnému užívateľovi vyskladnenie naskenovaním QR kódu balíčka.	S
024	Systém umožní správe skladu naskladnenie súčiastok importom pdf faktúry.	M
025	Systém umožní správe skladu vytvorenie faktúry.	M
026	Systém umožní správe skladu nahratie pdf súboru ku faktúre.	M
027	Systém umožní správe skladu úpravu už existujúcej faktúry.	M
028	Systém umožní správe užívateľov vytvorenie nového užívateľa.	M

029	Systém umožní správe užívateľov úpravu už existujúceho užívateľa.	M
030	Systém umožní správe užívateľov zmenu stavu užívateľa.	C
031	Systém umožní bežnému užívateľovi zmenu jeho hesla.	M
032	Systém umožní správe užívateľov zmenu užívateľských rolí.	M
033	Systém umožní správe projektov vytvorenie nového projektu.	M
034	Systém umožní správe projektov úpravu už existujúceho projektu.	S
035	Systém umožní správe projektov zmenu stavu projektu.	C
036	Systém umožní správe projektov zobrazenie všetkých projektov.	M
037	Systém umožní bežnému užívateľovi zobrazenie zoznamu všektých projektov, ktorých je členom.	M
038	Systém umožní správe projektov priradenie kusovníkov na projekt.	C
039	Systém umožní správe projektov priradenie súčiastok na projekt.	C
040	Systém umožní bežnému užívateľovi zobrazenie všetkých jeho vyskladnení.	S
041	Systém umožní správe projektov zmenu pracovníkov na projekte.	C

2.2.1.1 FR001

Každý sklad bude mať v rámci systému unikátny názov.

2.2.1.2 FR002

V rámci zobrazenia bude umožnené filtrovanie skladov podľa zatiaľ neurčených atribútov.

2.2.1.3 FR004

V rámci detailu budú zobrazené viaceré, neskôr špecifikované štatistiky týkajúce sa daného skladu.

2.2.1.4 FR005

Výsledný súbor bude obsahovať zoznam aktuálne naskladneného materiálu (názov súčiastky, počet kusov) a celkovú hodnotu tohto materiálu.

2.2.1.5 FR007

Každý alias bude mať v rámci systému unikátny názov. Ku aliasu bude možné priradiť iné aliasy ako jeho alternatívy.

2.2.1.6 FR010

Každá súčiastka bude mať v rámci systému unikátny názov a unikátne kódové označenie, ktoré bude slúžiť pre evidenciu čísla u obchodníka, u ktorého bola súčiastka zakúpená.

2.2.1.7 FR012

Zobrazené dáta jednotlivých súčiastkok budú špecifikované neskôr; spolu s nimi i atribúty, podľa ktorých sa bude dať v zozname filtrovať.

2.2.1.8 FR013

Každý kusovník bude mať v rámci systému unikátny názov. Pozostávať bude z konkrétnych súčiastok a ich počtov.

2.2.1.9 F014

Samotné vytvorenie kusovníka systém umožní aj import zo súboru. Jeho formát i štruktúra zodpovedá súboru *bom.xlsx* umiestnenom v prílohe.

2.2.1.10 FR015

Každá úprava bude znamenať vytvorenie nového kusovníka s iným číslom verzie pre prípad, že upravovaný kusovník už je použitý na niektorom z projektov.

2.2.1.11 FR016

Medzi zobrazovanými atribútmi budú názov, verzia a dátum vytvorenia. Zoznam bude filtrovateľný podľa názvu.

2.2.1.12 FR017

V rámci detailu budú zobrazené projekty, na ktorých je kusovník použitý. Ďalšie dáta budú špecifikované neskôr.

2.2.1.13 FR018

Celková hodnota súčiastok bude získaná z ceny určenej pri položkách príjemky pri naskladnení.

2.2.1.14 FR019

Naskladnenie bude realizované vytvorením príjemky spolu s jej položkami. Ku príjemke bude možné nahrať i elektronickú verziu faktúry, ku ktorej sa príjemka viaže.

2.2.1.15 FR020

Vyskladnenie bude realizované vytvorením výdajky. Jej položky budú obsahovať konkrétne balíčky, z ktorých bodú súčiastky odobraté. Každé vyskladnenie bude viazané na niektorý z projektov.

2.2.1.16 FR021

Pre každú položku na príjemke systém vytvorí balíček s počtom naskladnených kusov súčiastky odpovedajúcej danej položke.

2.2.1.17 FR022

Balíčky budú vytvárané podľa FR021 a nemusia teda odpovedať skutočnému rozdeleniu obdržaného materiálu od dodávateľa. Systém preto umožní toto dodatočné rozdelenie.

2.2.1.18 FR023

Pre každý jeden balíček bude umožnené vygenerovať QR kód s odkazom pre rýchlejšie vyskladnenie z daného balíčka.

2.2.1.19 FR024

Z nahranej faktúry v pdf formáte bude vyextrahovaných čo najviac dát pre urýchlenie vyskladnenia. Pôjde o informácie ako dátumy či číslo faktúry, rovnako tak jej jednotlivé položky. V prípade naskladňovania ešte neexistujúcej súčiastky systém umožní jej dodatočné vytvorenie v rámci procesu naskladňovania.

2.2.1.20 FR025

Každá faktúra v systéme bude obsahovať unikátne označenie.

2.2.1.21 FR028

Medzi povinné atribúty každého užívateľa budú patriť meno, unikátny email a heslo.

2.2.1.22 FR030

Množina stavov zatiaľ pozostáva zo stavu *aktívny* a stavu *neaktívny*, pričom neaktívnemu užívateľovi je zamedzený prístup do systému.

2.2.1.23 FR032

Umožnené bude pridávanie a odoberanie rolí jednotlivým užívateľom s výnimkou *admin* role. Tá bude od začiatku pridelená len jednému užívateľovi.

2.2.1.24 FR033

Každý projekt bude mať pridelenú jednu zodpovednú osobu, ktorú nebude možné meniť. Okrem toho môžu byť na projekt pridelené ďalšie osoby ako pracovníci.

2.2.1.25 FR035

Množina stavov zatiaľ pozostáva zo stavu *aktívny* a stavu *neaktívny*, pričom na neaktívny projekt nie je umožnené vyskladňovanie.

2.2.1.26 FR038, FR039

Priradenie kusovníkov a súčiastok na projekt bude slúžiť pre urýchlenie vyskladnenia z daného projektu a súčasne pre evidenciu materiálu nutného pre úspešné dokončenie projektu.

2.2.2 Nefunkčné (kvalitatívne) požiadavky

QR ID	QR Popis	Priorita
001	Systém bude výkonný a stabilný.	M
002	Systém bude multiplatformný.	M
003	Systém bude prehľadný.	M
004	Užívateľské kontá budú zabezpečené.	M

2.2.2.1 QR001

Od systému sa očakáva zvládnutie obsluhy desiatok užívateľov súčasne bez negatívneho vplyvu na dobu jeho odozvy. Systém bude taktiež pravidelne (časový interval zatiaľ nebol stanovený) vykonávať zálohu celej databázy.

2.2.2.2 QR002

Systém bude realizovaný ako webová aplikácia, vďaka čomu sa znížia hardvérové požiadavky na strane zadávateľa. Pre prístup do systému postačí počítač, smartfón či tablet s pripojením k internetu a webovým prehliadačom. Funkčnosť aplikácie bude zaručená pre prehliadače Internet Explorer vo verzii 9+, Google Chrome vo verzii 10+, Mozilla Firefox vo verzii 10+ a Opera vo verzii 12+. Odporúčané je však použtie najaktuálnejších verzií.

2.2.2.3 QR003

Systém bude vyhotovený s minimalistickým a intuitívnym rozhraním, pre čo najjednoduchšiu prácu s ním. Súčasne sa predpokladá i prispôsobenie rozhrania a jeho korektné zobrazenie na rôznych typoch zariadení, cez ktoré možno systém využívať.

2.2.2.4 QR004

Pre prístup užívateľa do systému bude vyžadované jeho heslo. Heslá budú v databáze uložené v nečitateľnej podobe zahešované pomocou bcrypt alebo Argon2 metód.

2.2.3 Prípady použitia

James Rumbaugh[29] definuje prípad použitia ako "špecifikáciu postupností činností, vrátane premenných a chybových postupností, ktoré systém, podsystém alebo trieda môžu vykonať prostredníctvom interakcie s externími aktérmi". Ide teda o zoznam akcií, ktoré je nutné vykonať pre dosiahnutie požadovaného cieľa.

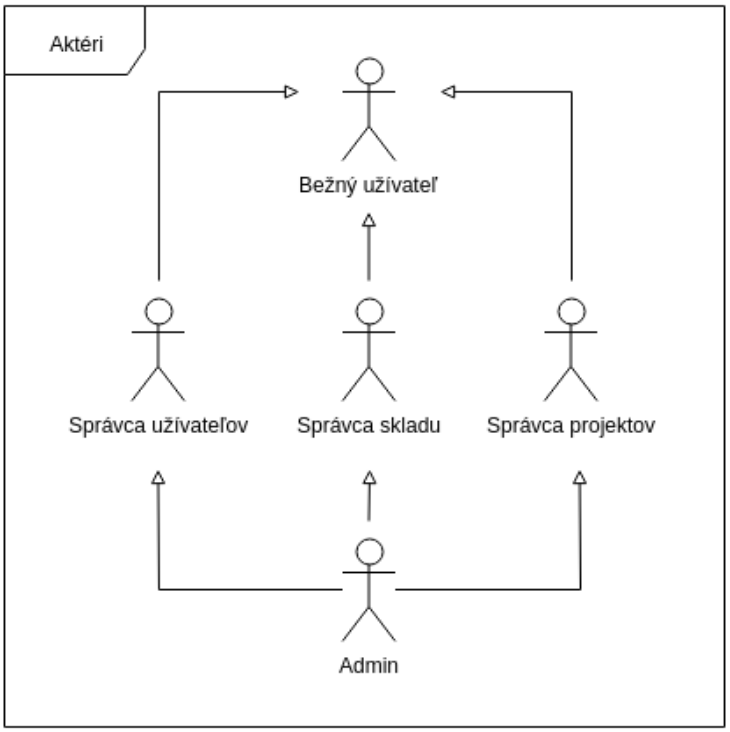
V rámci podsekcie je v podobe diagramov zaznačených niekoľko prípadov použitia pre jednotlivé role. Z nich bolo ďalej vybraných zopár, ktoré sú doplnené o scenár s postupnými krokmi. Všetky takto uvedené prípady použitia (ak nie je vyznačené inak) začínajú zobrazením domovskej stránky po prihlásení užívateľa.

2.2.3.1 Aktéri

Na základe zoznamu požiadaviek bolo vydefinovaných päť aktérov, ktoré odpovedajú rolám prideľovaným jednotlivým užívateľov v rámci systému. Jednotliví aktéri a ich vzťahy sú zachytení diagramom 2.3.

2.2.3.2 Bežný užívateľ

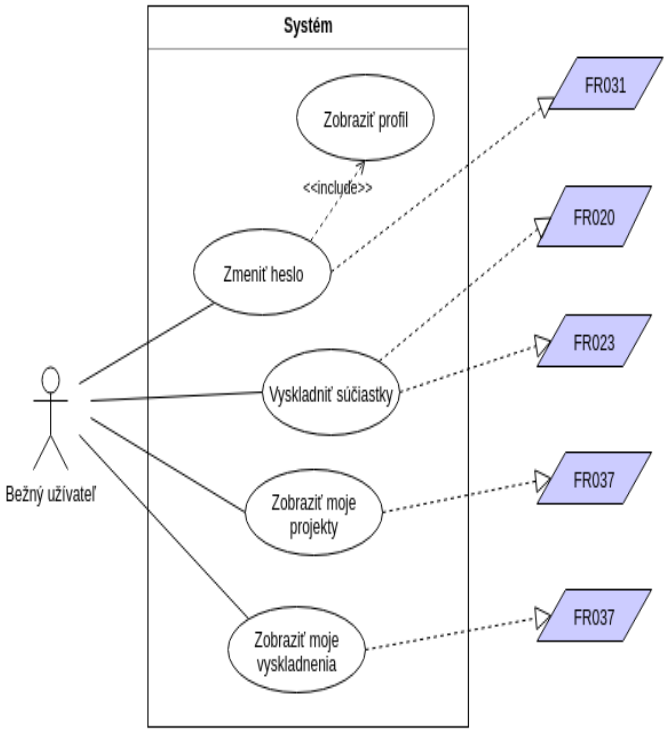
Rola *Bežný užívateľ* je implicitne priradená každému užívateľovi v systéme a nemôže mu byť odobraná. Prípady použita spolu s odpovedajúcimi požiadavkami sú zachytené na obrázku 2.4.



Obr. 2.3: Aktéri systému

Vyskladniť súčiastky

1. Užívateľ klikne na tlačidlo *Vyskladniť*
2. Užívateľ zvolí jeden z projektov, na ktorý bude vyskladnenie viazané
3. Užívateľ zvolí alias a jeho počet kusov na vyskladnenie v predvytvorenej sekcii pre výber aliasu
4. Ak užívateľ zvolil všetky potrebné aliasy, potom
   (a) Užívateľ klikne na tlačidlo *Uložiť*
   (b) Systém uloží informácie o vyskladnení
   (c) Systém užívateľa presmeruje na detail vyskladnenia
   (d) Systém užívateľa notifikuje správou na stránke o úspešnom vyskladnení
5. Alebo
   (a) Užívateľ klikne na tlačidlo *Pridaj alias*
   (b) Systém vytvorí novú sekciu pre výber aliasu
   (c) Užívateľ zvolí alias a jeho počet kusov na vyskladnenie v novej sekcii
   (d) Opakuj od 4.

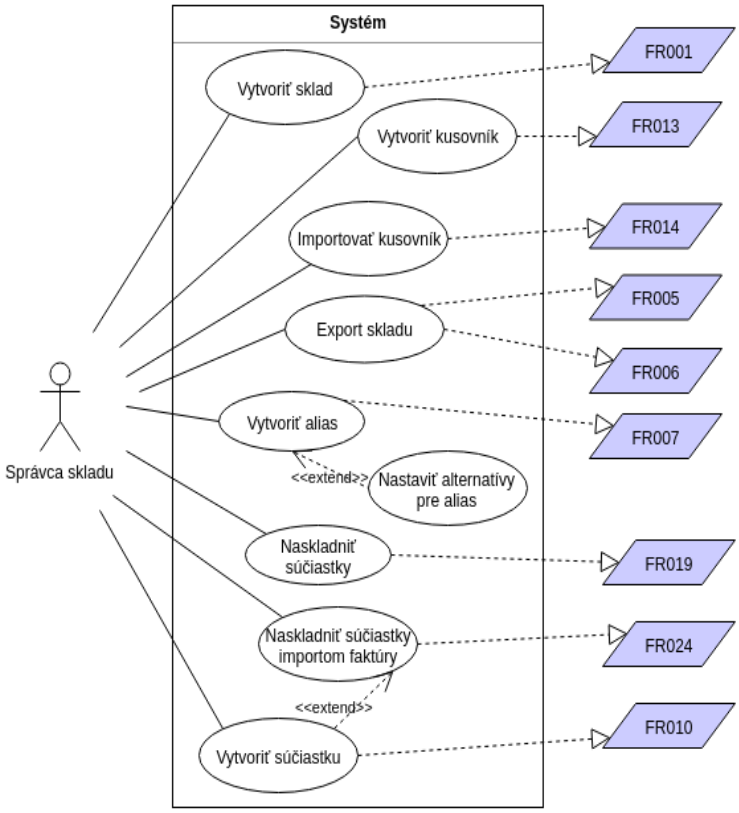


Obr. 2.4: Prípady použitia - Bežný užívateľ

2.2.3.3 Správca skladu

Naskladniť súčiastky

1. Užívateľ klikne na tlačidlo *Naskladniť*
2. Užívateľ zvolí jeden zo skladov, na ktorý bude naskladnenie viazané
3. Ak je faktúra pre dané naskladnenie už v systéme evidovaná, potom
   (a) Užívateľ vyberie v zozname faktúr požadovanú
4. Alebo
   (a) Užívateľ v zozname faktúr zvolí *Nová*
   (b) Systém zobrazí inputy pre vyplnenie ku faktúre
   (c) Užívateľ vyplní zobrazené inputy ku faktúre
5. Užívateľ zvolí súčiastku a zadá počet kusov a jednotkovú cenu
6. Ak užívateľ zvolil všetky potrebné súčiastky, potom
   (a) Užívateľ klikne na tlačidlo *Uložiť*
   (b) Systém uloží informácie o naskladnení



Obr. 2.5: Prípady použitia - Správca skladu

(c) Systém užívateľa presmeruje na zoznam všetkých naskladnení

(d) Systém užívateľa notifikuje správou na stránke o úspešnom naskladnení

7. Alebo

(a) Užívateľ klikne na tlačidlo *Pridaj súčiastku*

(b) Systém vytvorí novú sekciu pre výber súčiastky

(c) Užívateľ zvolí súčiastku a zadá počet kusov a jednotkovú cenu

(d) Opakuj od 6.

Naskladniť súčiastky importom faktúry

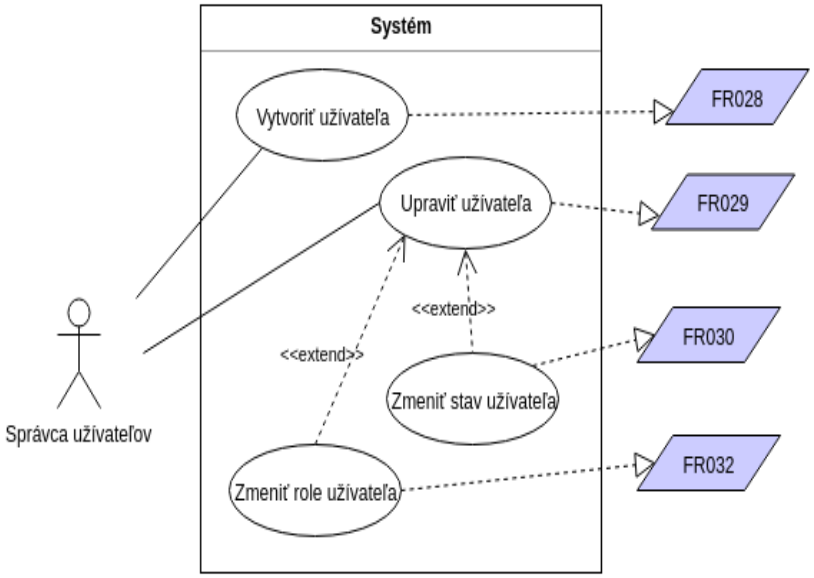
1. Užívateľ klikne na tlačidlo *Naskladnenia*
2. Užívateľ klikne na tlačidlo *Import* vpravo hore
3. Systém zobrazí modálové okno s formulárom
4. Užívateľ zvolí dodávateľa, nahrá faktúru v pdf formáte a zvolí uložiť

5. Systém vyextrahuje dáta z faktúry v závislosti od jej typu podľa dodávateľa
6. Systém užívateľa presmeruje na stránku naskladnenia s predvyplnenými hodnotami, ktoré sa podarilo vyextrahovať z faktúry (dátumy, položky, atď.)
7. Užívateľ zvolí jeden zo skladov, na ktorý bude naskladnenie viazané
8. Užívateľ doplní chýbajúce údaje v inputoch pre faktúru
9. Ak niektorá súčiastka z faktúry nebola v čase importu evidovaná v systéme, potom
   (a) Užívateľ klikne na tlačidlo *Vytvoriť* v sekcii súčiastky, ktorú chce vytvoriť
   (b) Systém zobrazí modálové okno s formulárom pre vytvorenie súčiastky
   (c) Užívateľ vyplní formulár a klikne na tlačidlo *Uložiť*
   (d) Systém uloží informácie o novej súčiastke
   (e) Systém zruší modálové okno a predvyplní input pre súčiastku novo vytvorenou
   (f) Opakuj od 9.
10. Alebo
    (a) Užívateľ klikne na tlačidlo *Uložiť*
    (b) Systém uloží informácie o naskladnení
    (c) Systém užívateľa presmeruje na zoznam všetkých naskladnení
    (d) Systém užívateľa notifikuje správou na stránke o úspešnom naskladnení

2.2.3.4 Správca užívateľov

Vytvoriť užívateľa

1. Užívateľ klikne na zoznam všetkých naskladnení
2. Užívateľ klikne na tlačidlo *Vytvoriť* vpravo hore
3. Systém zobrazí formulár pre vyplnenie
4. Užívateľ vyplní povinné inputy pozostávajúce z mena, emailu a hesla
5. Ak má mať nový užívateľ priradenú inú rolu ako *Bežný užívateľ*, potom
   (a) Užívateľ zvolí zo zoznamu rolí tie, ktoré budú pridelené
6. Alebo
   (a) Užívateľ ponechá zoznam rolí pre priradenie prázdny
7. Užívateľ klikne na tlačidlo *Uložiť*
8. Systém presmeruje užívateľa na zoznam všetkých užívateľov
9. Systém užívateľa notifikuje správou na stránke o úspešnom vytvorení nového užívateľa

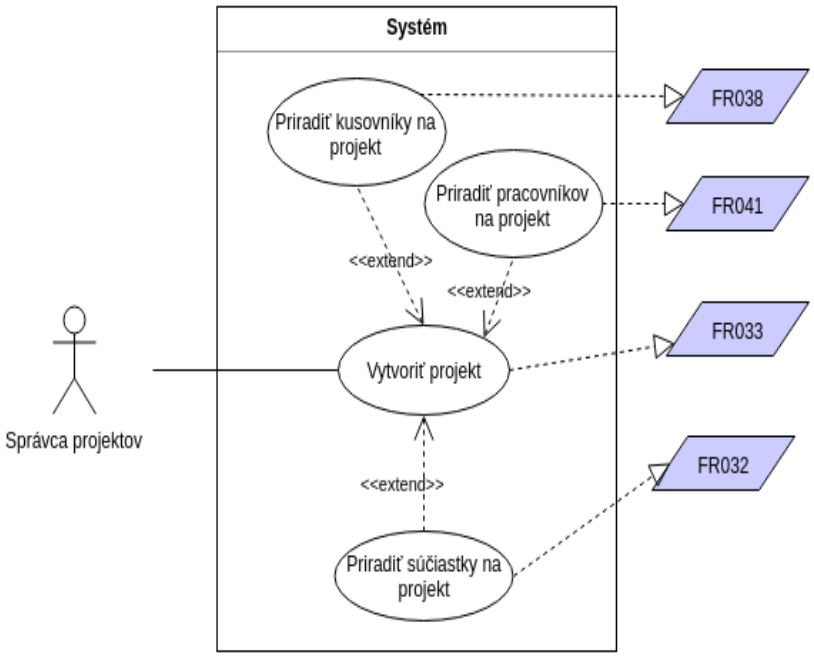


Obr. 2.6: Prípady použitia - Správca užívateľov

2.2.3.5 Správca projektov

Vytvoriť projekt

1. Užívateľ klikne na zoznam všetkých projektov
2. Užívateľ klikne na tlačidlo *Vytvoriť* vpravo hore
3. Systém zobrazí formulár pre vyplnenie
4. Užívateľ vyplní povinné inputy pozostávajúce z názvu a za projekt zodpovednej osoby
5. Užívateľ vyberie zo zoznamu ľudí tých, ktorí majú byť na projekt pridelení ako pracovníci
6. Užívateľ vyberie zo zoznamu potrebné kusovníky na projekt
7. Užívateľ vyberie zo zoznamu potrebné súčiastky na projekt
8. Užívateľ klikne na tlačidlo *Uložiť*
9. Systém presmeruje užívateľa na zoznam všetkých projektov
10. Systém užívateľa notifikuje správou na stránke o úspešnom vytvorení nového projektu



Obr. 2.7: Prípady použitia - Správca projektov

2.2.4 Business Domain Model

Na základe zozbieraných a zadefinovaných požiadaviek bolo možné určiť entity, ktoré budú v systéme vystupovať. Ich zoznam a väzby medzi nimi sú zachytené BDM diagramom 2.8.

Sklad z pohľadu systému predstavuje reálne miesto uskladnenia materiálu s unikátnym názvom a lokalitou. V rámci skladu sú umiestňované tzv. balíčky. Balíček možno chápať ako zoskupenie viacerých kusov materiálu rovnakého typu, teda súčiastok. Okrem počtu kusov v sebe nesie i jeho presnejšie umiestnenie v sklade dané slovným popisom či QR kód pre urýchlenie procesu vyskladnenia jeho naskenovaním. Do balíčkov je umiestňovaných niekoľko kusov jednej, konkrétnej súčiastky.

Súčiastka reprezentuje reálny materiál, ktorý sa v sklade nachádza. Jej unikátny názov je doplnený kódovým označením podľa jej dodávateľa. Dodávateľ v sebe nesie informáciu o názve a prípadnom odkaze na jeho webové stránky. Súčiastky, ktoré sú vo svojej podstate rovnaké a líšia sa len dodávateľom sú združené pod rovnakým aliasom. Z hľadiska návrhu by sa mohlo javiť lepším riešením evidencia iba jednej súčiastky, a jej rôzne kódové označenia presunúť ku väzbe na dodávateľa. Súčiastka sama o sebe by bola akýmsi aliasom. V tomto prípade by sme ale prišli o možnosť vyskladnenia zásob práve od konkrétneho dodávateľa. Ďalším z dôvodov môže byť zistenie vady u súčiastky len od jedného z viacerých dodávateľov. Takýmto spôsobom docielime možnosti zistenia na ktorých zariadeniach bola táto súčiastka použitá či v ktorých balíčkoch sa nachádza.

Jednou z požiadaviek bolo zohľadniť možnosť, že niektoré súčiastky, hoc navzájom rôzne, môžu byť v niektorých prípadoch použité bez rozdielu. Príkladom by mohlo byť použitie zlatej matice a striebornej matice. Práve tento požiadavok je riešený príznakom alternatívy pri aliase. Súčiastky v rámci aliasov s rovnakou alternatívou je možné medzi sebou navzájom zamieňať.

Viaceré súčiastky je možné ďalej spájať do tzv. kusovníkov. Ich primárnou úlohou je poslúžiť ako zoznam materiálu potrebného pre výrobu určitého zariadenia. Tie sú výstupom projektov, na ktorých zadávateľ pracuje. Projekt v sebe nesie okrem názvu i dátumy jeho vytvorenia či ukončenia. S projektom sú spojené už zmienené kusovníky; okrem nich však i jednotlivé súčiastky a to pre prípad, kedy je napr. nutná náhrada za kus poškodený pri zhotovovaní zariadenia.

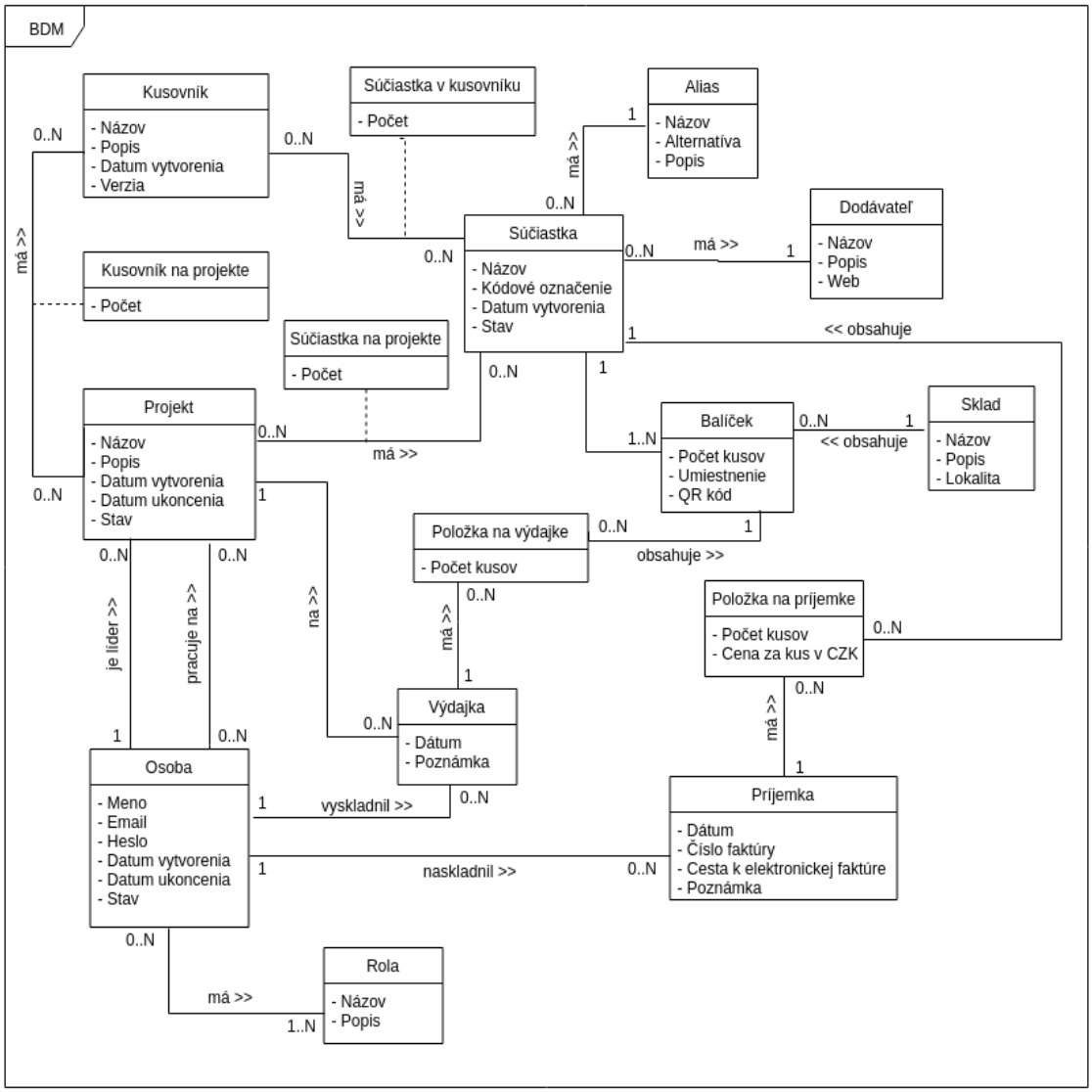
Vyskladňovanie zásob je možné len pre účely použitia na projektoch. Každý projekt musí mať pridelenú zodpovednú osobu. Medzi základné atribúty evidované pri osobách v systéme patria meno, email, heslo či stav, ktorý hovorí o tom, či je danému užívateľovi povolený prístup do systému alebo nie. Vykonávanie určitých činností osobou v systéme je podmienené rolou, ktorú má priradenú. Rola je určená názvom.

Jednou z hlavných funkcií systému je možnosť evidencie naskladnenia. V BMD je reprezentovaná entitou Príjemka. Tá k sebe viaže osobu, ktorá naskladnenie vykonala a súčasne jednotlivé položky na príjemka nesúce informáciu o naskladnenom počte konkrétnej súčiastky.

Funkcionalita vyskladnenia materiálu je zachytená entitou Výdajka. Podobne ako u príjemky, i tu je držaná väzba na osobu zodpovednú za vyskladnenie. Odlišnou je však väzba jednotlivých položiek. Tie nie sú previazané s konkrétnou súčiastkou, ale s balíčkom a celkovým počtom kusov odoberaných z daného balíčka.

2.2.5 Existujúce riešenia

Vzhľadom na prílišnú špecifickosť bolo od podrobnejšej analýzi už existujúcich riešení upustené. Z mnoha dostupných systémov pre podporu správy skladového hospodárstva boli preskúmané napr. Pohoda[14], Money[13] , Byznys[11] či Karat[12]. Každý jeden prekypoval funkcionalitou, ktorú nami navrhované riešenie postráda; naopak pre náš systém dôležité prvky ako tvorba kusovníkov, správa projektov či extrahovanie dát z pdf súborov faktúr neboli v čase vyhotovenia práce dostupné ani u jedného z nich.



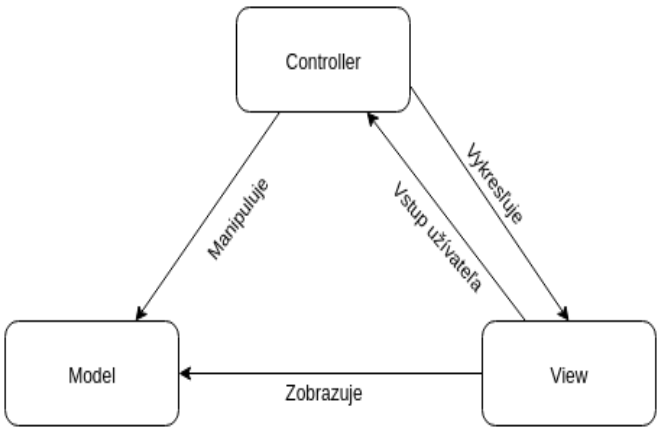
Obr. 2.8: Business Domain Model

Kapitola 3

Návrh systému

3.1 Architektúra systému

Dôležitým krokom v rámci tvorby aplikácie je správna voľba architektúry. Tá predstavuje abstraktný pohľad na systém, na systémové komponenty a ich vzťahy. Pre nami navrhovanú aplikáciu bola zvolená architektúra rozšírená o vrstvu pre správu obchodnej logiky. V dôsledku jej použitia dochádza k rozdeleniu systému na tri navzájom prepojené časti. *Model* je reprezentáciou dát, s ktorými aplikácia pracuje. *View* je zobrazením modelu a ďalších prvkov užívateľského rozhrania, skrz ktoré užívateľ interaguje s aplikáciou. Úlohou *Controller* komponenty je spracovanie užívateľského vstupu, a následné vykonanie príslušných akcií a zmien v modeli. Stará sa teda o celkové previazanie funkčnosti aplikácie.

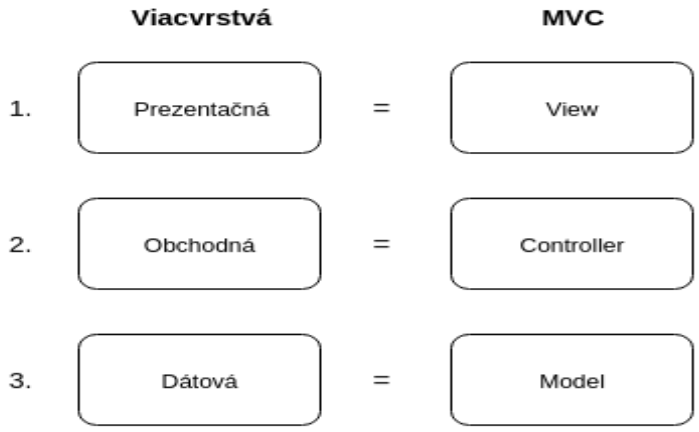


Obr. 3.1: MVC architektúra

Charakteristikou takto navrhnutej architektúry je takzvaný low coupling, teda nízka miera vzájomného prepojenia komponent[22]. Táto vlastnosť so sebou prináša zvýšenú prehľadnosť kódu a jeho ľahšiu udržateľnosť. Rozčlenenie systému podľa MVC taktiež umožňuje tímom paralelnú prácu vývojárov na jednotlivých častiach a teda celkovo rýchlejší vývoj finálneho produktu.

Nevýhodu môže predstavovať komplexnosť zahrnutá v pochopení nových vrstiev abstrakcie a v samotnej nutnosti dekompozície systému na zmienené časti MVC.

V ponímaní MVC z pohľadu viacvrstvej architektúry[7] možno jej časti pripodobniť ku prezentačnej, obchodnej a dátovej vrstve ako je uvedené na obr. 3.2



Obr. 3.2: MVC ako viacvrstvá architektúra

Riešenie obchodnej logiky je teda doménou vstvy Controller. Obchodná logika v sebe zahŕňa dátovú validáciu požiadaviek z prezentačnej (View) vrstvy a súčasne logiku rozhodovaciu, výpočty, agregáciu informácii z viacerých zdrojov apod.

Pre dosiahnutie ešte väčšej prehľadnosti, udržateľnosti, znovupoužiteľnosti a ľahšej testovateľnosti sme navrhovanú MVC architektúru rozšírili o tzv. *servisnú vrstvu*, do ktorej bola odčlenená zložitejšia obchodná logika.

3.2 Databáza

Úloha databázy je v našom systéme nepostradateľnou. Každá interakcia s ním vyžaduje prácu s dátami, ktoré je potrebné vhodným spôsobom evidovať, pričom dôraz je nutné klásť i na ich dlhodobé uchovanie. Finálnej voľbe predchádzalo zoznámenie sa s používanými databázovými technológiami.

Medzi najpopulárnejšie databázové technológie patria databázy relačné a nerelačné[3].

3.2.1 Relačné databázy

Organizácia dát databáz tohto typu spočíva v relačnom modeli, navrhnutom E. F. Coddom v roku 1970 [19]. Dáta sú organizované do tabuliek (relácií) reprezentujúcich entitné typy. Atribúty danej entity sú definované stĺpcami a riadky predstavujú jej konkrétne inštancie, pričom každý z nich musí byť v rámci danej tabuľky unikátne adresovateľný.

Ešte pred tým, ako je možné ukladať samotné dáta je nutné definovať databázovú schému. Tá je tvorená práve tabuľkami spolu so vzťahmi medzi nimi vyjadrenými prostredníctvom primárnych a cudzích kľúčov.

Pre zabezpečenie dátovej integrity sa transakcie relačných databáz vyznačujú štyrmi vlastnosťami - atomicitou, konzistenciou, izoláciou a trvácnosťou, v angličtine známych pod akronymom ACID.

3.2.2 Nerelačné databázy

Zásadný rozdiel v porovnaní s relačnými databázami leží v štruktúre ukladania dát. Práve podľa nej delíme nerelačné (označované ako NoSQL) databázy na dokumentové, stĺpcové, grafové či key-value.

Využitie nachádzajú najmä v oblasti Big Data či IoT (akronym anglického Internet of things) technológii pri práci s neštrukturovanými dátami, kedy schému na úrovni databázi nemožno vopred definovať.

Rozdiel možno nájsť taktiež v prístupe ku transakčnému spracovaniu. Na úkor atomicity, konzistencie a trvácnosti dát naprieč systémom stavia NoSQL na ich dostupnosti a škálovateľnosti.

3.2.3 Voľba

Vopred známa schéma a nutnosť zaistenia konzistencie a integrity dát pri práci s nimi v značnej miere zavážili pri výslednom rozhodnutí, ktoré padlo na relačný typ databáz. Istou mierou zavážila i predchádzajúca skúsenosť práce práve s týmto typom databáz, ktorý pre projekty podobného či väčšieho rozsahu bol vždy postačujúci. Medzi výhody možno zaradiť i použitie štandardizovaného jazyka SQL.

Dátový model a konkrétny zvolený databázový systém sú zmienené v kapitole 4.

3.3 Užívateľské rozhranie

Pre aplikáciu je okrem toho, aby bola dobre naprogramovaná, či už z hľadiska efektivity fungovania alebo vnútornej architektúry, dôležitá i jej dobrá použiteľnosť. V snahe o jej zabezpečenie sme sa pri návrhu užívateľského rozhrania opierali o základné a všeobecné princípy popísané v podobe pravidiel v [30].

Jednou z vlastností užívateľského rozhrania, o ktorú bolo autorom usilované je konzistencia. Konzistencia v používanej terminológii či v spôsobe zobrazovanie prvkov

rovnakého typu rovnakým spôsobom, podobných podobne. S tým je spojená taktiež predvídateľnosť a postupné vytváranie stereotypov v používaní.

Cieľovou skupinou nebol návrh rozhrania systému v žiadnom smere obmedzovaný. Jeho obsluhu budú vykonávať bežní užívatelia s dlhoročnou praxou v používaní počítačových i mobilných zariadení, pre ktoré je nový systém určený.

Dôraz bol pri návrhu taktiež kladený na poskytovanie spätnej väzby užívateľovi, aby bol informovaný o výsledkoch jeho práce s aplikáciou - či už pôjde o akcie úspešné alebo nie.

Požiadavkou zo strany zadávateľa bola realizácia nového systému ako webová aplikácia s možnosťou použitia na mobilných zariadeniach. Návrh rozhrania bol preto vedený princípom Mobile-First, ktorý spočíva v prednostnom vývoji rozhrania na zobrazovacích plochách s menším rozlíšením a uhlopriečkou. Snaha je kladená na sprostredkovanie kvalitného zážitku z použitia aplikácie práve užívateľom s displejmi menších rozmerov. Z pohľadu dôležitosti teda stavia mobilné zariadenia minimálne na úroveň tradičných počítačov s veľkými obrazovkami. Zobrazovaný obsah sa však môže v závislosti od obrazovky líšiť, kedy väčšie displeje môžu byť po stránke zobrazených prvkov bohatšie.

Pre dosiahnutie minimalistického užívateľského rozhrania bol pri jeho návrhu použitý *flat design*. Ide o návrhový štýl, ktorého hlavnou zásadou je minimálne použitie štylistických elementov pôsobiacich ako 3D objekty. Pozornosť je upriamená taktiež na minimalistické použitie typografie a farieb.

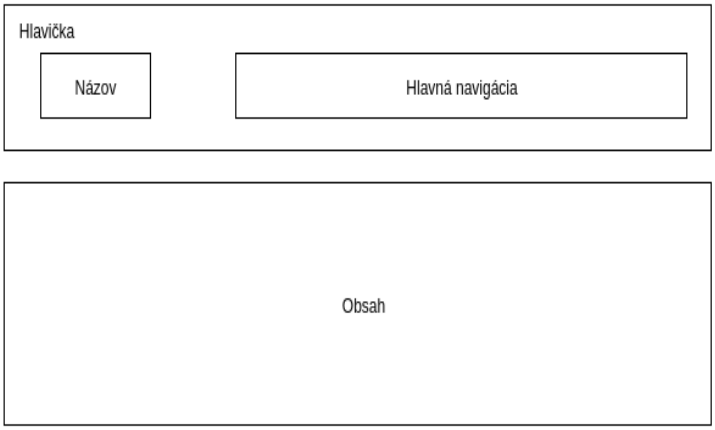
3.3.1 Farebná schéma

Obr. 3.3: Farebná schéma

Za dominantnú farbu použitú pri zafarbení väčšinovej časti poziadia bola použitá fialová (#9179BA). Tmavý odtieň fialovej farby (#3D3847) bol použitý pre podfarbenie navigácie a textu na svetlom pozadí. Pri elementoch hlavného obsahu pozostávajucich najmä z tabuliek a formulárov bola pre ich zvýraznenie použitá na pozadí biela farba (#FFFFFF). Pre tlačidlá či aktívne odkazy nachádzajúce sa v omrvinkovej navigácii bol použitý svetlý odtieň fialovej farby (#F2F0F4).

3.3.2 Rozloženie aplikácie

Základnými dvoma prvkami grafického rozloženia aplikácie zobrazenými na obr. 3.4 sú hlavička a obsah.



Obr. 3.4: Základné prvky užívateľského rozhrania

3.3.2.1 Hlavička

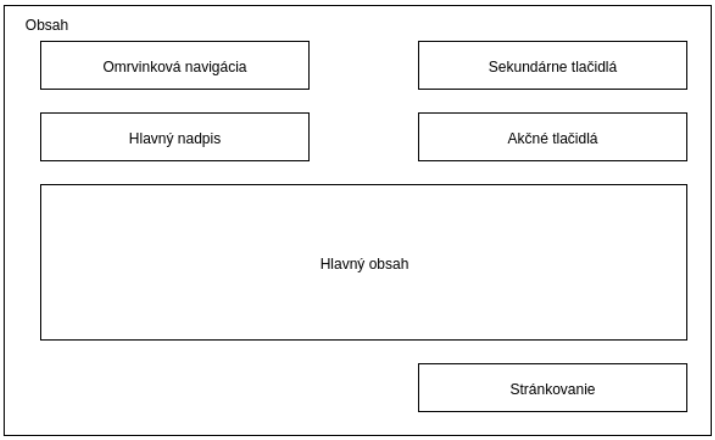
Hlavička je vždy najvrhnejšie zobrazovaným elementom na stránke. Hlavičku samotnú možno rozdeliť na pravú a ľavú časť.

V ľavej časti hlavičky je zobrazený názov aplikácie. Názov býva zväčša substituovaný logom, ktoré však momentálne nie je k dispozícii, no môže zaň byť názov časom nahradený.

Pravá časť je tvorená prvkami hlavnej navigácie, ktorými sú odkazy smerujúce do jednotlivých častí aplikácie. Pri zobrazení aplikácie neprihláseným užívateľom pozostáva hlavička iba z jedného odkazu a to na stránku prihlásenia do systému. Po prihlásení bude tento odkaz nahradený "klikateľným"menom užívateľa. Po kliknutí naň sa zobrazí ponuka s odkazom smerujúcim na stránku pre zmenu hesla a odkazom pre odlhásenie užívateľa. Okrem toho bude zoznam odkazov rozšírený o ďalšie, v závislosti od užívateľovi priradených rolí. Správca užívateľov bude mať k dispozícii odkaz smerujúci na stránku so zoznamom všetkých užívateľov v systéme. Pre správcu projektu pribudne odkaz na stránku so zoznamom všetkých projektov. Ponuka prihláseného správcu skladu bude najrozsiahlejšia vzhľadom na množstvo jeho zodpovedností. Súčasťou jemu zobrazenej ponuky budú odkazy smerujúce na zoznamy všetkých naskladnení, vyskladnení, faktúr či zoznamy súčiastok, balíčkov, aliasov, zariadení, dodávateľov a v neposlednom rade skladov. Pre urýchlenie činností pribudnú do hlavnej navigácie i odkazy napr. priamo na stránku s formulárom naskladnenia. Pri zmenách stránky zostáva pre užívateľa hlavička stále rovnaká.

3.3.2.2 Obsah

Sekcia obsahu pod hlavičkou pozostáva z niekoľkých častí, čo je znázornené na obr.3.5



Obr. 3.5: Obsah pod hlavičkou

Jedným z elementov umiestnených priamo pod hlavičkou je omrvinková navigácia. Vďaka nej má užívateľ spôsob, ako si udržať prehľad o polohe práve navštívenej stránky. Umožňuje mu navigáciu po prejdených stránach až po aktuálne zobrazenú, resp. na druhej strane po počiatočnú, teda hlavnú. Omrvinková navigácia zachytáva užívateľom prejdenú cestu začínajúc od východiskového bodu. Príkladom v našej aplikácii môže byť úprava faktúry s kódovým označením 123. Preň by navigácia bola nasledovná: *Domovská stránka - Faktúry - 123 - Úprava.*

Na pravej strane na rovnakej úrovni ako omrvinková navigácia môžu byť umiestnené sekundárne tlačidlá. Tie budú obsahovať odkazy na s aktuálne zobrazeným obsahom súvisiace stránky.

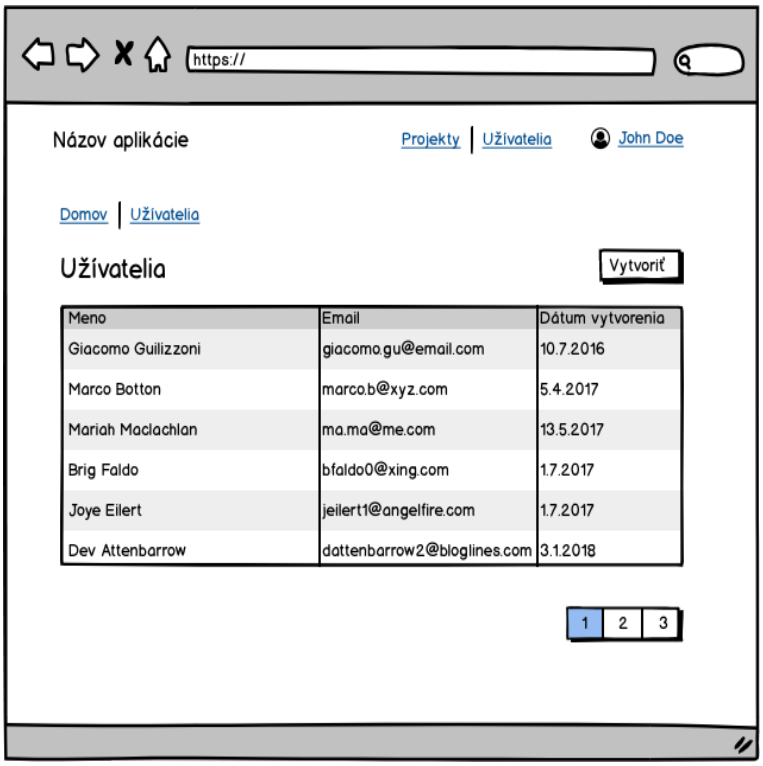
Pod omrvinkovou navigáciou je miesto určené pre hlavný nadpis.

Napravo od hlavného nadpisu budú umiestnené akčné tlačidlá pre vykonanie určitej akcie týkajúce sa správy zobrazeného obsahu. Okrem umiestnenia budú tieto tlačidlá od sekundárnych odlíšené i veľkosťou a sfarbením. Príkladom môže byť v prípade zobrazého zoznamu súčiastok tlačidlo s nápisom *Vytvoriť* pre vytvorenie novej súčiastky.

V časti hlavného obsahu bude zachytená hlavná informácia, ktorá má byť v rámci stránky užívateľovi vyobrazená. Môže ňou byť napríklad tabuľka pre zobrazenie záznamov konkrétnej entity či formulár pre vytvorenie alebo úpravu konkrétneho záznamu.

Pri zobrazení väčšieho množstva záznamov niektorej entity v rámci hlavného obsahu bude tento zoznam takzvane stránkovaný. V rámci jednej stránky bude zobrazené len obmedzené množstvo záznamov a zobrazenie zvyšných bude docieliteľné na samostatných stránkach. Odkazy na tieto stránky budú číslované a umiestnené pod hlavným obsahom.

Pre lepšiu predstavu toho, ako budú jednotlivé elementy na stránke vyzerať bol vytvorený wireframe 3.6 , zachytávajúci zobrazenie stránkovaného zoznamu užívateľov.



Obr. 3.6: Wireframe - Stránka užívateľov

Kapitola 4

Implementácia

4.1 Databáza

4.1.1 Použitá databáza

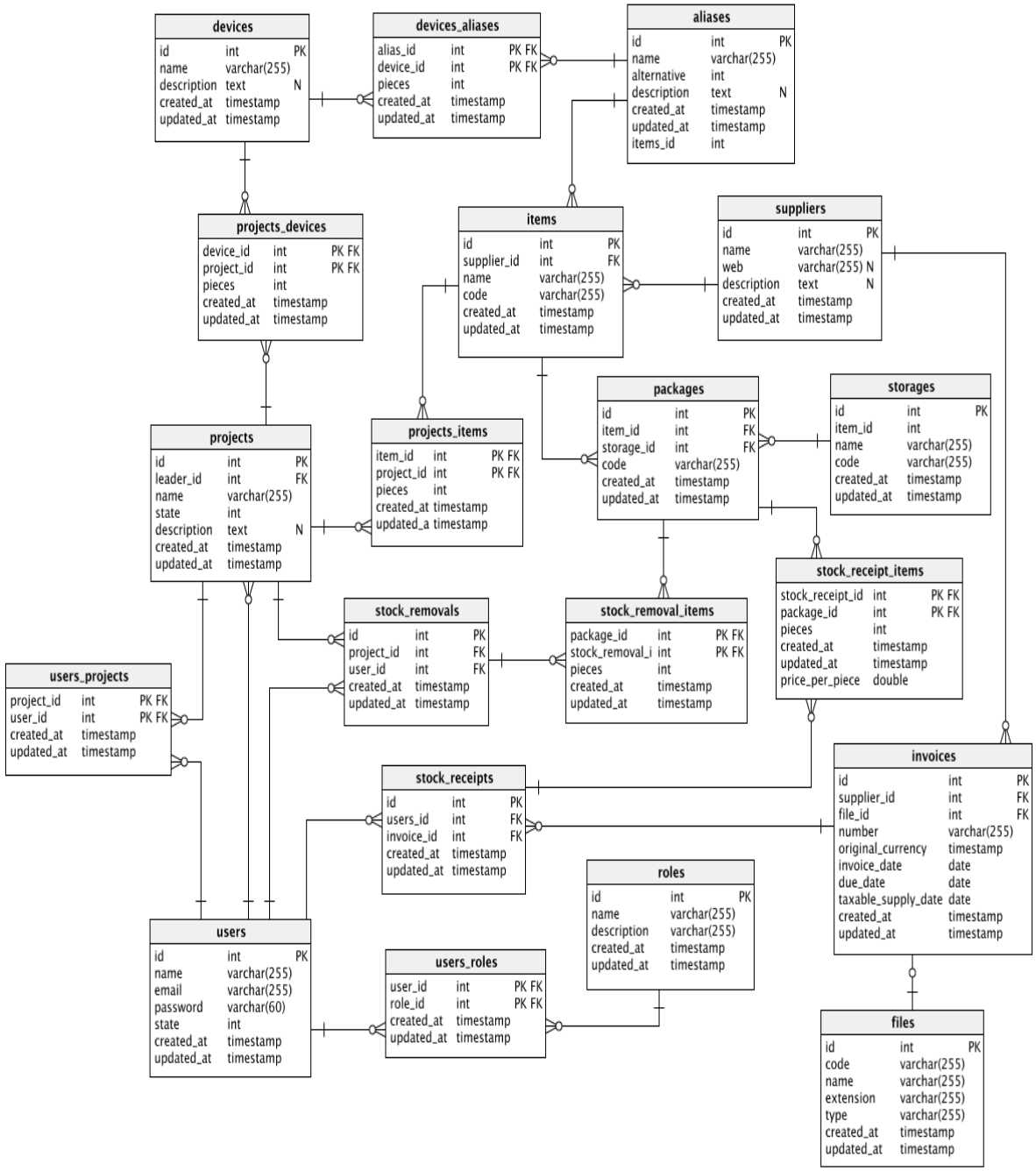
Pre implementáciu bol zvolený databázový systém PostgreSQL[10]. Ide o öpen-source"riešenie s aktívnou komunitou a širokou podporou v rámci operačných systémov. Dodržuje SQL štandard oveľa striktnejšie ako MySQL - s PostgreSQL často porovnávaný, taktiež öpen-source"databázový systém. Medzi výhody zvoleného systému nutno zaradiť i podporu mnohých rozšírení [20].

4.1.2 Dátový model

V predchádzajúcej kapitole autor zmienil voľbu technológie relačných databáz ako perzistentné úložisko dát aplikácie. Jej samotnému vytvoreniu predchádzalo definovanie dátového modelu. Ten východza z analýzy a svojím prevedením môže pripomínať nami zadefinovaný BDM model, keďže je podrobnejšou dátovou reprezentáciou v ňom vystupujúcich entít.

Za použitý dátový model bol zvolený takzvaný entitne vzťahový model (z anglického entity-relationship model), ktorý je široko používaný pri navrhovaní databáz a systémových analýzach[32].

Diagram pre uvažovanú aplikáciu je zachytený na obr.4.1. Jednotlivé tabuľky odpovedajúce entitám obsahujú základné atribúty dané názvom, dátovým typom a špecifikátorom, ktorý je voliteľný. Špecifikátor nadobúda hodnoty N (hodnota, ktorej zadanie nie je povinné), PK (primárny kľúč tabuľky) a FK (cudzí kľuč tabuľky referencujúci primárny kľúč inej tabuľky pre vyjadrenie ich vzťahu). Práve pre zaznačenie vzťahov bola použité tzv. Crow's foot značenie[2].



Obr. 4.1: Dátový model

Každá z tabuliek obsahuje dva atribúty pre zaznačenie časových značiek vytvorenia nového záznamu v tabuľke (atribút created_at) a úpravy záznamu (atribút updated_at).

V rámci požiadaviek a rozsahu práce nebolo implementované mazanie záznamov. Ide však o bežnú operáciu nad dátami, na realizáciu ktorej bolo pomyslené. Tabuľky, ktorých záznamy bude možné mazať budú rozšírené o časovú značku zmazania (atribút deleted_at). Pri odstránení záznamu z rozhrania aplikácie bude jej hodnota nastavená na čas vykonania tejto akcie. Aplikácia bude vždy pracovať len so záznamamy, ktoré daný atribút nemajú nastavený, teda ešte neboli zmazané. Užívateľovi sa preto bude javiť, že dáta boli odstránené, no kvôli uchovaniu historických dát nebudú z databázy vyňaté.

4.2 Použitie aplikačného rámca

4.2.1 Jazyk PHP

Výberu konkrétneho aplikačného rámca predchádzal výber programovacieho jazyka. Pre realizáciu implementácie bol zvolený jazyk PHP vo verzii 7.1.

PHP je populárnym jazykom[8] pre vývoj klient-server dynamických webových aplikácii. Podľa [9] je PHP najpoužívanejším jazykom v stredne veľkých webových aplikáciach. Je jazykom skriptovacím a nesie prívlastky ako öpen-source"či multiplatformný.

Výhodou je okrem predchádzajúcej skúsenosti autora s týmto jazykom i široká komunita ľudí, ktorá v danom jazyku vyvíja. S väčšou komunitou sa spája i väčšia pravdepodobnosť toho, že problém, na ktorý človek natrafí už bol niekým riešený.

Práve softvérové riešenia konkrétnych problémov bývajú distribuované formou takzvaných softvérových balíčkov, ktoré svojím použitím rozširujú aplikáciu o požadovanú funkcionalitu. Pre ich správu je nutné použiť správcu balíčkov, ktorý zabezpečuje ich správnu inštaláciu či verzovanie v rámci projektov. Konkrétnym správcom PHP balíčkov je Composer, ktorý bol využitý aj v rámci práce. Tento spôsob zdieľania funkcionality je taktiež využívaný aj v iných jazykoch.

4.2.2 Laravel

Dnes pri vývoji zložitejších aplikácii už nebýva zvykom použitie len samotného programovacieho jazyka a jeho konštrukcií, ale i takzvaného aplikačného rámcu (z anglického framework). Aplikačný rámec je softvérová štruktúra, sada knižníc, ktorej cieľom je uľahčenie vývoja prevzatím typických problémov danej oblasti, teda sprístupnenie množstva funkcií, ktoré sám vývojár už nemusí implementovať [5]. Od neho sa potom očakáva doplnenie kódu špecifického pre ním riešenú úlohu.

Pre náš systém zvolený jazyk PHP sa pýši veľkým množstvom aplikačných rámcov, ktoré sú nad ním vystavané. Nami zvoleným aplikačným rámcom sa stal Laravel, ktorý je v súčasnosti najpopulárnejším öpen-source"riešením[6]. Okrem predchádzajúcich skúseností autora pri výbere zavážila i prepracovanosť dokumentácie, veľkosť komunity a množstvo integrovaných súčastí samotného aplikačného rámca.

Práve vďaka komunite býva odhaľovanie chýb a ich následné odstraňovanie oveľa rýchlejšie. V tom tkvie výhoda použita bezpečnostných prvkov ponúkaných aplikačným rámcom.

Laravel vo svojom jadre stavia na základoch aplikáčného rámca Symfony, využívajúc mnohé z jeho komponent. Modularita je jednou z dominant Laravelu, pričom na správu balíčkov využíva dedikovaného, už zmieneného správcu Composer.

Pri použití Laravel aplikačného rámca je aplikácia po stránke štruktúry implicitne rozdelená na dve vrstvy - Router a Controller, ktoré boli v rámci práce rozšírené o servisnú vrstvu, z dôvodu ľahšej udržateľnosti a väčšej prehľadnosti. Každá vrstva zodpovedá za inú časť aplikačnej logiky a je reprezentovaná iným typom programovej komponenty.

4.2.2.1 Router

Router je komponentou, ktorá prijatú požiadavku spracováva ako prvá. Jednotlivé prístupové body aplikácie, ktoré je možné adresovať, bolo nutné definovať v na to určených konfiguračných súboroch. Na základe typu HTTP metódy a URL prijatej požiadavky sú tieto body mapované na príslušné metódy tried v Controller vrstve, ktoré sú Routerom vyvolané.

4.2.2.2 Controller

Triedy tejto vrstvy sú zodpovedné za dátovú validáciu spracovávaných požiadaviek a za realizáciu jednoduchšej biznis logiky apikácie. Ich úlohou je taktiež zaslanie patričnej odpovedi užívateľovi.

Contoller trieda obsahuje metódy - v názvosloví Laravelu takzvané akcie, pričom každá z nich je dedikovaná pre spracovanie konkrétnej požiadavky.

Jedným z argumentov týchto metód je objekt reprezentujúci požiadavku, obsahujúc jej dáta, typ, ip adresu odosielateľa apod. Možnosťou objektov tohto typu je taktiež nadefinovanie validačných pravidiel, ktoré by inak boli umiestnené do metód Controller triedy. Takýmto odtienením validácie docielime čistejšieho a prehľadnejšieho kódu, a súčasne si môžme byť istý, že v momente začiatku vykonávania kódu Controller metódy pracujeme s už valídnymi dátami.

Ďalším, často využívaným argumentom bývajú premenné z dynamických URL. Premennou v príkladovej URL */users/123* je fragment 123, ktorý odpovedá hodnote identifikátoru konkrétneho užívateľa. V metóde potom možno tento identifikátor použiť pre získanie dát o danom užívateľovi z databáze a následné vykonanie práce s nimi. Zmena hodnoty uvažovaného framgmentu by mala za následok prácu s iným užívateľom.

Metódy v rámci jednej Controller triedy sú zodpovedné za úkony nad jednou entitou. Ako príklad možno uviesť StorageController, metódy ktorého sú zodpovedné za správu entity sklad.

Základné metódy každej Controller triedy sú zachytené v nasledujúcej tabuľke nižšie. Osvedčeným postupom je snaha zachovávať túto štruktúru.

Potreba pridania ďalšej metódy väčšinou značí možnosť vytvorenia novej Controller triedy a presun tejto metódy tam.

Metóda	Popis
index	zobrazenie všetkých záznamov entity
show	zobrazenie konkrétneho záznamu entity
create	zobrazenie formulára pre vytvorenie záznamu entity
store	vytvorenie záznamu entity
edit	zobrazenie formulára pre úprava záznamu entity
update	uloženie úprav záznamu entity
destroy	zmazanie záznamu entity

4.2.3 Service

Pre vykonanie zložitejšej biznis logiky a validácie sú v rámci Controller vrstvy využívané servisné triedy, ktoré boli zmienené v kapitole 3.1 Architektúra systému.

Neúspešná validácia je v mnohých prípadoch riešená "vyhodením"príslušnej výnimky, ktorá je následne užívateľovi vhodnou formou prezentovaná.

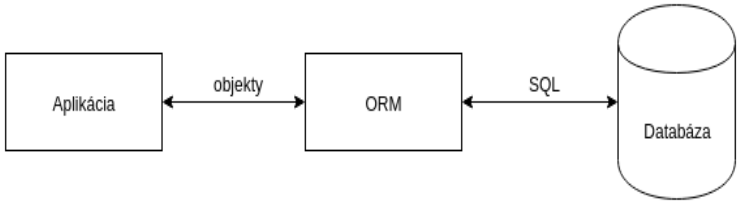
Azda najväčšou výhodou umiestnenia funkcionality do servisných tried je ich znovupoužiteľnosť. Triedy v rámci Controller vrstvy nie sú vo svojej podstate určené pre vzájomné volania, čo by mohlo mať za následok výskyt duplicít kódu na mnohých miestach. Práve odstránenie opakovanej implementácie robí kód prehľadnejším a udržateľnejším.

V rámci práce bol vytvorený i takzvaný trait s názvom BaseService. Trait v poníman jazyka PHP predstavuje spôsob zdieľania rovnakej, už implementovanej funkcionality medzi rôznymi triedami bez nutnosti využitia dedičnosti. BaseService obsahuje metódy pre základné CRUD (akronym pre anglické *Create Read Update Delete*) operácie nad inštanciami entity, pre ktorú je konkrétna servisná trieda určená.

4.2.4 Objektovo relačné mapovanie

Pre zjednodušenie práce s databázou v zdrojovom kóde aplikácie bola použitá technika objektovo relačného mapovania (z anglického object-relational mapping).

Ide o programovaciu techniku zaisťujúcu automatickú konverziu dát medzi relačnou databázou a objektami využívanými v aplikácii. Laravel využíva ORM návrhový vzor Active Record pre jeho vlastnú implementáciu s názvom Eloquent. Pri použití Eloquent sú jednotlivé tabuľky relačnej databázy reprezentované prislúchajúcimi triedami, označovanými ako modely. Objekty týchto tried odpovedajú zasa záznamom v riadkoch tabuliek. Rozhranie objektov je rozšírené o metódy, ktorých volanie odtieňuje základné databázové CRUD operácie nad odpovedajúcim záznamom, a teda čiastočne i nutnosť znalosti dopytovacieho jazyka databázy [23].



Obr. 4.2: Schéma využitia ORM v aplikácii

Výhodou použitia ORM je taktiež možnosť práce s viacerými typmi databáz abstrahovanej pod jedným uniformným rozhraním, ktoré poskytujú práve spomínané objekty.

4.2.5 Migrácie

Rovnako ako zmienené ORM odľahčuje vývojára od použitia dopytovacieho jazyka databázy pri práci so záznamami v nej, uľahčujú migrácie vytvorenie a úpravy samotnej databázovej schémy.

Jednotlivé zmeny, ako napríklad vytvorenie novej tabuľky či jej rozšírenie o ďalšie atribúty, sú reprezentované súbormi - v jazyku Laravelu nazývanými práve migrácie, obsahujúcimi ich popis.

Výhodou je okrem abstrakcie použitej databázy i jednoduché zdieľanie jej schémy medzi ľudmi podieľajúcimi sa na vývoji aplikácie, realizované jedným z bežných spôsobov zdieľania kódu ako je napríklad verzovací nástroj Git.

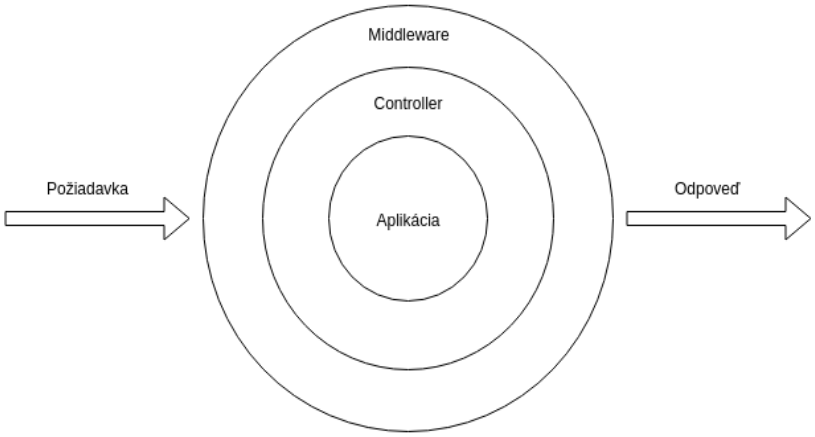
Na samotné migrácie môže byť nazerané ako na verzovací nástroj schémy a teda, že by nemalo dochádzať k úprave historických zmien. V štádiu vývoja aplikácie, kedy nebolo ešte vykonané jej nasadenie do produkčného prostredia, a tým pádom neriskujeme možnú stratu či poškodenie v nej uložených užívateľských dát, je možné upravovať aplikované migrácie a následne schému vytvoriť nanovo. Medzi výhody tohto prístupu udržovania menšieho počtu migrácií patrí napríklad rýchlejší beh testov pracujúcich s databázou či vyššia prehľadnosť jednotlivých úprav [1].

V rámci práce boli vytvorené migrácie pre dosiahnutie podoby schémy, ktorá odpovedá nami uvažovanému dátovému modelu 4.1.

4.2.6 Middleware

Medzi najdôležitejšie integrované súčasti aplikačného rámca Laravel nutno zaradiť i takzvaný middleware. Ide o mechanizmus určený pre filtrovanie HTTP požiadaviek vstupujúcich do aplikácie. Pred tým, ako je požiadavka spracovaná metódou niektorej z Controller tried, prechádza sériou preddefinovaných a nami vytvorených vrstiev, ktoré ju napríklad po dátovej stránke rozširujú alebo na základe nesplnenia určitej podmienky zabránia jej ďalšiemu priechodu aplikáciou.

Príkladom v rámci práce použitej middleware vrstvy, ktorú ponúka sám Laravel, je ochrana pred CSRF útokom. Tá pri spracovaní požiadavky overuje prítomnosť



Obr. 4.3: Zasadenie middleware vrstiev v Laraveli

unikátneho tokenu pre zabránenie odosielania dát z iných stránok ako tých, ktoré boli vytvorené samotnou aplikáciou.

Autor využil tento mechanizmus pre implementáciu viacerých middleware, autorizujúcich požiadavky prihláseného užívateľa na základe jemu priradených rolí.

4.2.7 Lokalizácia

Za jazyk použitý pre texty aplikácie zobrazované užívateľovi bola zvolená angličtina. Súčasťou Laravelu je však jednoduchá možnosť tvorby jazykových mutácií, a to umiestnením textov do určených súborov a ich následné adresovanie v zdrojovom kóde. Pre jednoduchšie rozšírenie v budúcnosti bol tento spôsob použitia textov zavedený autorom už od začiatku vývoja.

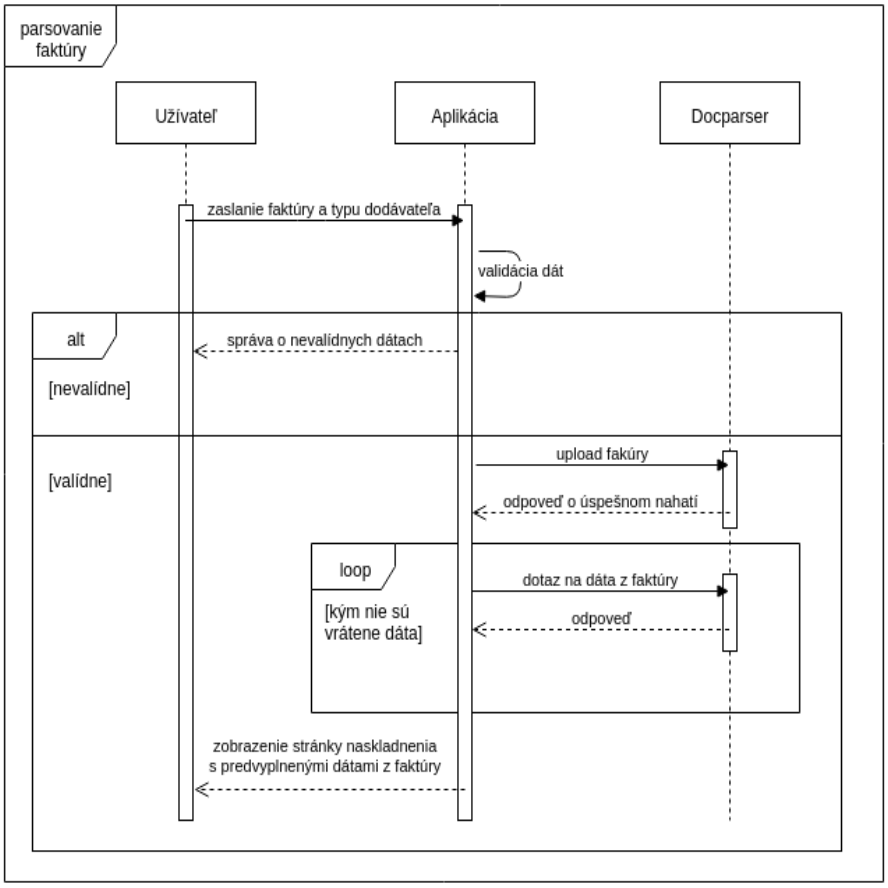
4.2.8 Parsovanie faktúr

Jednou z požiadaviek na uvažovanú aplikácu bola možnosť naskladnenia súčiastok do skladu, a to s využitím pdf faktúry poskytnutej dodávateľom (požiadavka FR024).

Na základe dohody so zadávateľom boli určení štyria dodávatelia, ktorých faktúry bude možné takýmto spôsobom využiť. Pre každého dodávateľa zadávateľ poskytol niekoľko vzorových faktúr pre určenie ich opakujúcich sa spoločných znakov.

Implementácia parsovania pdf faktúr za účelom extrahovať z nich potrebné dáta bola realizovaná použitím online nástroja tretej strany Docparser[4]. Tento nástroj poskytuje užívateľsky prívetíve rozhranie pre vytvorenie parserov a nastavenie sekcií pdf dokumentu, v ktorých má požadované dáta, ako napríklad položky faktúry, vyhľadávať a pomocou OCR technológie vyčítať. Prostredníctvom REST prístupových bodov je potom možné nahrávať nové súbory a dotazovať sa na dáta z nich v JSON podobe.

Diagram 4.4 zachýtava priebeh komunikácie počínajúc nahratím faktúry užívateľom až po zobrazenie požadovanej stránky naskladnenia s predvyplnením získanými dátami.

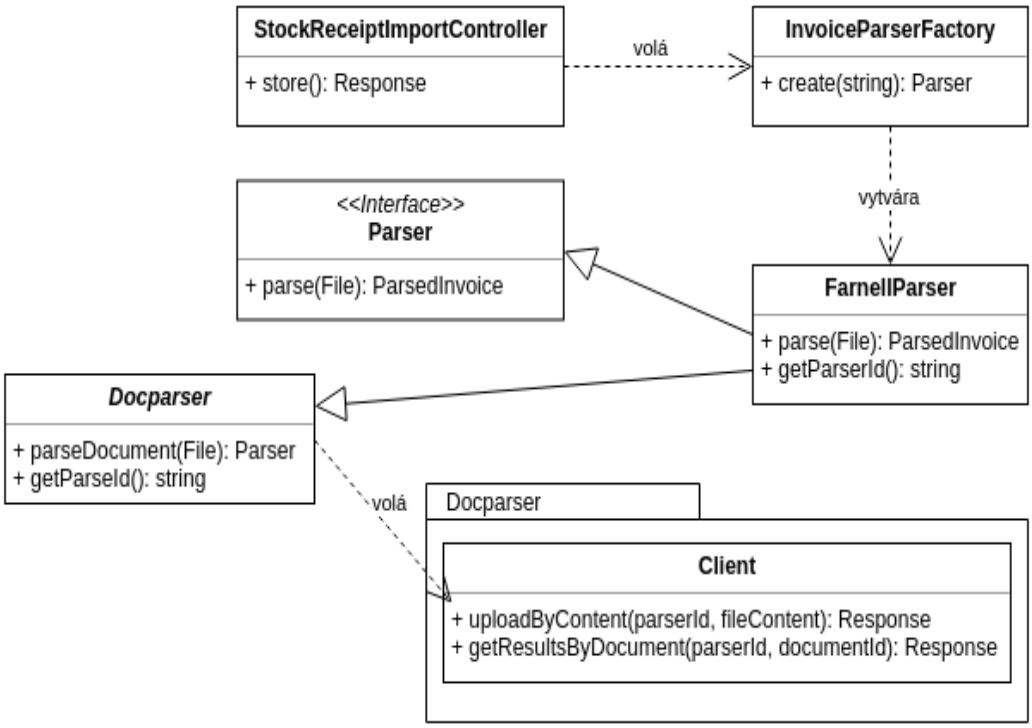


Obr. 4.4: Diagram interakcie parsovania faktúry

Na strane aplikácie bola autorom implementovaná trieda *InvoiceParserFactory*, ktorá na základe užívateľom poskytnutého typu dodávateľa vráti inštanciu triedy určenej pre parsovanie daného súboru.

Spolu boli vytvorené štyri triedy určené pre parsovanie faktúr - každá pre jeden typ dodávateľa. Všetky implementujú spoločné rozhranie *InvoiceParser* s jedinou metódou *parse*, ktorá ako argument prijíma súbor a vracia objekt nami definovaného typu zaobalujúci dáta z faktúry. Vďaka tomuto rozhraniu je možné používať parser vratený našou factory triedou bez nutnej znalosti jeho konkrétneho typu.

Triedy parserov rozširujú abstraktnú triedu *Docparser*, ktorá abstrahuje komunikáciu s klientom volajúcim externé API tretej strany. Tento klient bol taktiež vytvorený v rámci práce. Vzťah medzi zmienenými triedami je znázornený na diagrame 4.5. Pre názornosť je v ňom uvedený len jeden typ konkrétneho parseru - FarnellParser.



Obr. 4.5: Diagram tried pri parsovaní faktúry

4.3 Užívateľské rozhranie

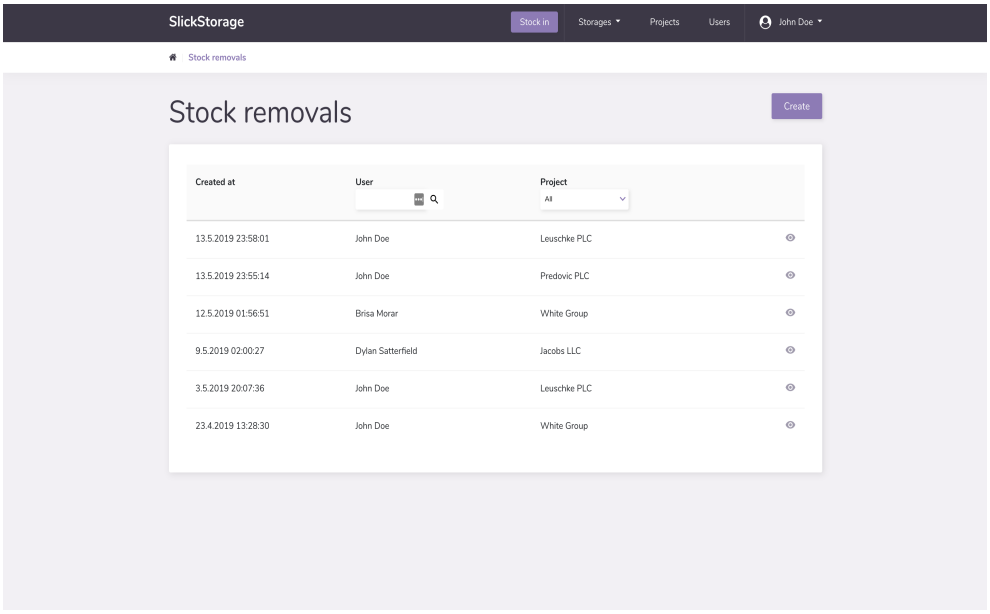
4.3.1 Views

Pre zobrazovanie obsahu užívateľovi v prostredí prehliadača bola použitá technika jeho renderovania na strane serveru. To znamená, že dopytované stránky sú vytvorené serverom a ich celý obsah vrátený v odpovedi prehliadaču.

V použitom aplikačnom rámci Laravel sú tieto stránky nazývané views. Views môžu okrem prehliadačom spracovateľného obsahu pozostávajúceho z HTML, javascriptu či kaskádových štýlov i PHP kód, ktorý server pred odoslaním interpretuje. Pre uľahčenie zápisu niektorých PHP konštruktov ponúka Laravel šablónovací jazyk Blade.

4.3.2 Vzhľad

Na tvorbu responzívnej prezentačnej stránky aplikácie boli použité takzvané kaskádové štýly (Cascade Style Sheets) , ktoré sú jednoduchým mechanizmom na vizuálne formátovanie obsahu stránok na internete.

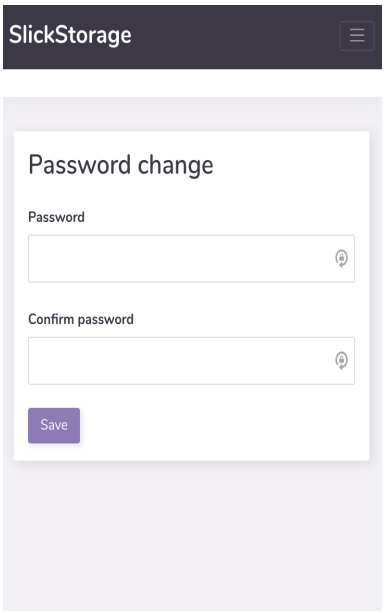


Obr. 4.6: Zobrazenie tabuľky vyskladnení

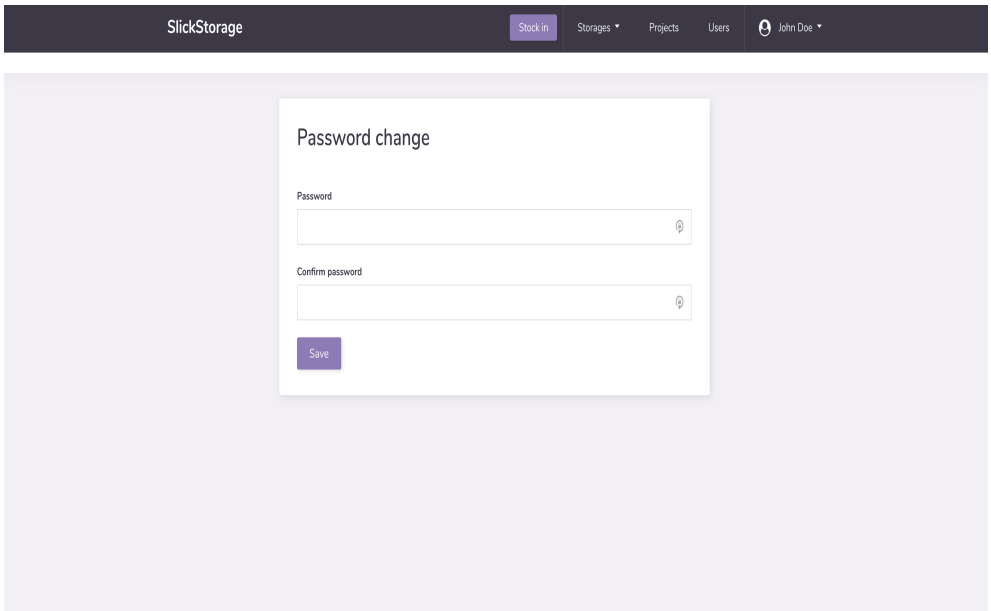
Pre uľahčenie implementácie užívateľského rozhrania bol pri tvorbe kaskádových štýlov použitý preprocesor Sass. Jeho hlavnou výhodou je rozšírenie CSS syntaxe o pokročilejšie konštrukty zjednodušujúce ich zápis a udržateľnosť. Kód napísaný pomocou Sass je potom kompilovaný do prehliadačom interpretovateľného CSS.

Pre kompiláciu bol zvolený nástroj webpack[16]. Úlohou tohto nástroja je súčasne i minifikácia výsledného kódu. Tá pozostáva z dvoch krokov. Ako prvé sú jednotlivé CSS súbory zlúčené do jedného. Následne dochádza ku odstráneniu formátovania, ktoré uľahčuje čitateľnosť kódu pre človeka, avšak je nepotrebná pre správnu interpretáciu prehliadačom. Výsledkom tohto druhu optimalizácie je zníženie počtu dotazov prehliadača na server, čo má za následok skrátenie doby jej načítania a teda zníženie rizika možného užívateľského diskomfortu.

Implementáciu niektorých častí rozhrania odľahčilo použitie voľne dostupných knižníc. Pre aplikáciu najpodstatnejšou bola jQuery - knižnica s bohatým API, ktorá predstavuje vrstvu abstrakcie pri interakcii Javascriptu a HTML štruktúry stránok.



Obr. 4.7: Zobrazenie nastavenia hesla na mobilnom zariadení



Obr. 4.8: Zobrazenie nastavenia hesla na monitore počítača

Kapitola 5

Testovanie

Neodmysliteľnou súčasťou vývoja aplikácie je overovanie jej správnej funkčnosti testovaním. Existuje mnoho rôznych druhov testov, líšiac sa svojou komplexitou, dobou trvania či štádiom projektu, v ktorom sa vykonávajú. Príkladom môžu byť testy jednotkové či integračné testy, testy užívateľského rozhrania, záťažové testy a ďalšie.

Okrem typov samotných testov rozlišujeme i rôzne postupy použité pri ich tvorbe. Práve spôsob zakomponovania procesu testovania do vývoja zohráva dôležitú úlohu najmä z dlhodobého hľadiska. Bez ohľadu na to, ako flexibilná je architektúra či skvele je navrhnuté rozdelenie systému, bez testov sa človek zdráha vykonať zmeny v kóde kvôli strachu zo zanesenia neodhalených chýb[25].

Jedným z takýchto spôsobov je takzvaný testovaním riadený vývoj (z aglického Test Driven Development). Založené je na malých, opakujúcich sa krokoch, počnúc definíciou funkcionality, vytvorením testu pre danú funkcionalitu a až potom implementáciou samotnej funkcionality. Tento postup si autor vyskúšal v rámci uvažovanej aplikácie pri zapracovávaní funkcionalít ako sú prihlásenie do systému, zmena hesla či CRUD operácie nad niektorými z entít modelu.

Aplikácia zhotovená v rámci práce bola testovaná viacerými spôsobmi, ktorým sú venované nasledujúce sekcie.

5.1 Jednotkové testy

Jednotkový test (z anglického *unit test*) je časťou kódu overujúcou funkčnosť jednej izolovanej časti aplikácie (väčšinou jednu metódu). Overenie je vykonané jej spustením a následnou kontrolou splnenia predom definovaných predpokladov. Pri ich nesplnení je test považovaný za neúspešný.

Medzi základné charakteristiky jednotových testov patrí ich nezávislosť na iných testoch či na poradí ich spustenia. Dôležitá je ich opakovateľnosť v zmysle nezávislosti na prostredí, v ktorom sú spustené a v poskytnutí deterministického výsledku. Beh jednotkových testov by mal byť rýchly. Ich spúštanie by malo byť časté, aby sa čím skôr dokázalo objaviť prípadné chyby.

Pre testovanie bol použitý aplikačný rámec PHPUnit. Test predstavuje jednu metódu testovacej triedy. Každá z týchto metód obsahuje aspoň jedno volanie takzvanej *assert* metódy, ktorá vyhodnocuje splnenie definovaného predpokladu.

Jednotkovím testovaním bola v rámci práce overená funkčnosť úrovne servisných tried. V rámci testov bola využitá i testovacia inštancia databázy pre kontrolu správnosti vykonania CRUD operácii nad objektami modelov. Pre uľahčenie generovania dát boli implementované takzvané *factories* - triedy zodpovedné za tvorbu inštancii konkrétneho modelu prednaplnené náhodnými dátami.

Uvedený je príklad testu vytvorenia nového užívateľa prostredníctvom triedy UserService.

```
/** @test */
public function it_can_create_user()
{
   $user = UserService::create([
      'name'     => 'John Doe',
      'email'    => 'john.doe@email.com',
      'password' => 'password'
   ]);

   $this->assertEquals('John Doe', $user->name);
   $this->assertEquals('john.doe@email.com', $user->email);
   $this->assertTrue(Hash::check('password', $user->password));
}
```

5.2 Integračné testy

Podstata integračných testov spočíva v overení funkčnosti dvoch alebo viacerých navzájom nezávislých jednotiek aplikácie ako celok[28]. Jednotky v našom ponímaní predstavujú jednotlivé vrstvy aplikácie.

Typom integračných testov implementovaných v rámci práce sú takzvané *feature testy*. Pri ich spustení dochádza k simulovaniu zasielania požiadaviek prehliadača na prístupové body aplikácie, čím je napodobnená interakcia samotným užívateľom.

Pri tomto druhu testovania bol okrem PHPUnit použitý i aplikačný rámec Mockery. Slúži pre vytváranie takzvaných *mockov* - objektov, ktorých správanie je možné upraviť pre potreby testu a nahradiť nimi závislosti testovaných objektov. Tvorba mockov našla využitie napríklad pri testovaní parsovania pdf faktúr, teda pri práci s API tretej strany.

Ukážkovým feature testom je test úspešného prihlásenia užívateľa do systému.

```
/** @test */
public function user_can_login_with_correct_credentials()
{
   $user = factory(User::class)->create([
      'password' => Hash::make($password = 'password')
   ]);

   $response = $this->post(route('login'), [
      'email'    => $user->email,
      'password' => $password
   ]);

   $response->assertRedirect(route('home'));
   $this->assertAuthenticatedAs($user);
}
```

5.3 Testovanie použiteľnosti

Cieľom tohto testovania je odkryť najzávažnejšie problémy, na ktoré môžu užívatelia pravdepodobne naraziť[21]. Testovanie môže prebiehať kedykoľvek počas procesu samotného vývoja, ale rovnako tak i v dobe, kedy je už aplikácia funkčná a prispieť tým k jej možnému redizajnu.

Jednou z najdôležitejších a najčastejšie používaných metód testovania použiteľnosti je užívateľské testovanie. Jeho princíp spočíva v pozorovaní a zaznamenávaní interakcie užívateľov cieľovej skupiny s danou aplikáciou, v snahe odhaliť chyby znižujúce kvalitu užívateľského zážitku. Úkony užívateľa bývajú riadené scenárom vytvoreným na základe prípadov použitia.

Pri vývoji nami uvažovanej aplikácie boli organizované stretnutia so zadávateľom, na ktorých mu boli predstavované zmeny implementované od posledného stretnutia. V rámci toho prebiehalo manuálne testovanie vedené autorom v spoluúčasti zadávateľa pre demonštráciu postupov vykonania niektorých požadovaných funkcionalít v systéme. Takto odhalené nedostatky boli následné zaznačené a riešené.

Kapitola 6

Záver

6.1 Zhodnotenie práce

Cieľom práce bolo analyzovať, navrhnúť a implementovať na mieru zostavený softvér pre podporu skladového hospodárstva pre malý podnik s konkrétnym zadávateľom, spolu s jeho vhodným spôsobom otestovania. Úvodná kapitola je venovaná stručnej analýze zadávateľom aktuálne využívaného riešenia pre správu skladu. Na základe tejto analýzy a viacerých stretnutí so zadávateľom bol vydefinovaný zoznam požiadaviek na nový systém, ktorý mal byť realizovaný ako webová aplikácia. Z nich boli následne určené parametre pre správny naväzujúci návrh aplikácie. Medzi ne patria aktéri systému a s nimi spojené prípady použitia, rovnako taktiež doménový model pre zachytenie základných entít vystupujúcich v systéme.

V rámci návrhu bola určená základná architektúra systému a použitá databázová technológia ako spôsob perzistentného uloženia dát. Navrhnuté bolo i užívateľské rozhranie odpovedajúce určeným požiadavkam a spĺňajúce moderné štandardy a trendy.

Výsledná aplikácia bola implementovaná použitím aplikačného rámca Laravel vo verzii 5.8. Ide o rámec jazyka PHP, so širokou komunitou, množstvom poskytovanej funkcionality a bohatým ekosystémom. Vytvorený bol i komplexný dátový model relačnej databázy, na ktorom aplikácia stavia. Navrhnuté a implementované bolo taktiež užívateľské rozhranie, vyhovujúce základným požiadavkám a spĺňajúce moderné štandardy.

Funkčnosť implementovanej aplikácie bola počas vývoja testovaná jednotkovými a integračnými testami, ktoré poslúžia i pre kontrolu spätnej kompatibility v rámci ďalšieho vývoja. Testovanie za účasti zadávateľa prebiehalo na stretnutiach formou spoločného, manuálneho otestovania nových funkcionalít počas ich demonštrácie.

Pevné základy vzdelania v oblasti softvérového vývoja, nadobudnuté počas štúdia na ČVUT v Prahe poslúžili autorovi pri realizácii tejto práce, ktorá vo všetkých svojich fázach obohatila jeho skúsenosti v oblasti vývoja.

6.2 Možnosti ďalšieho vývoja

Ako jeden z ďalších krokov nasadenie aktuálnej verzie aplikácie pre možnosť jej použitia zadávateľom. Následné úpravy v podobe implementácie zvyšných požiadaviek budú inkrementálne zasadené do už bežiacej aplikácie.

Plánované je i nahradenie systému tretej strany využívaného pre parsovanie faktúr. Po funkčnej stránke je dostačujúci, avšak pri dlhodobejšom využívaní je tento systém spoplatnený. Pri nenájdení vyhovujúceho, voľne dostupného riešenia, ktorého hľadanie bolo doposiaľ neúspešné, zamýšla autor implementáciu vlastného parsera pomocou aplikačného rámca Tesseract[15].

V rámci ďalšieho vývoja plánuje autor taktiež implementáciu funkcionality pre zisťovanie najvýhodnejších cien požadovaných súčiastok naprieč online verziami obchodov určených dodávateľov.

Zoznam bibliografických odkazov

[1] *How migrations might be slowing down your Laravel tests* [online]. Alex Vanderbist. Dostupné z: `https://www.postgresql.org/`. [vid. 5.5.2019].

[2] *Crow's Foot Notations*, 2019 [online]. Dostupné z: `http://www2.cs.uregina.ca/~bernatja/crowsfoot.html`, . stav z 5.5.2019.

[3] DB-Engines Ranking. In: *DB Engines* [online]. solid IT GmbH 2019. Dostupné z: `https://db-engines.com/en/ranking`, . [vid. 3.5.2019].

[4] *Docparser* [online]. SureSwift Capital, Inc. Dostupné z: `https://docparser.com/`, . [vid. 6.5.2019].

[5] *Co je to framework?* [online]. Peter Láng. Dostupné z: http://langi.cz/webarna/co-je-to-framework, .

[6] Open-source frameworks. In: *github* [online]. Dostupné z: `https://github.com/topics/framework`, . [vid. 6.5.2019].

[7] *N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE* [online] Guru99. Dostupné z: `https://www.guru99.com/n-tier-architecture-system-concepts-tips.html`, . [vid. 3.5.2019].

[8] TIOBE Index for May 2019. In: *TIOBE Index for May 2019* [online]. TIOBE Software BV 2019. Dostupné z: `https://www.tiobe.com/tiobe-index/`, . [vid. 6.5.2019].

[9] PHP Market Position. In: *Usage statistics and market share of PHP for websites* [online]. W3Techs 2019. `https://w3techs.com/technologies/details/pl-php/all/all`, . [vid. 6.5.2019].

[10] *PostgreSQL* [online]. The PostgreSQL Global Development Group. Dostupné z: `https://www.postgresql.org/`, . [vid. 5.5.2019].

[11] *Byznys* [online]. Byznys software, s.r.o. Dostupné z: `https://www.byznys.eu/cs/evidence-br-skladoveho-hospodarstvi`, . [vid. 1.5.2019].

[12] *Karat* [online]. KARAT Software a. s. Dostupné z: `https://www.karatsoftware.cz/erp-karat/skladovy-system`, . [vid. 1.5.2019].

[13] *Money* [online]. Solitea Česká republika, a.s. Dostupné z: `https://money.cz/`, . [vid. 1.5.2019].

[14] *Pohoda* [online]. FIRMADAT s.r.o. Dostupné z: `http://www.ucetnictvi-ekonomicky-system.cz/o-ucetnictvi-pohoda/skladove-hospodarstvi`, . [vid. 1.5.2019].

[15] *Tesseract* [online]. Google LLC. Dostupné z: `https://opensource.google.com/projects/tesseract`, . [vid. 9.5.2019].

[16] *Webpack*, 2019 [online]. Dostupné z: `https://webpack.js.org/`, . [vid. 7.5.2019].

[17] AMBLER, S. W. Mapping objects to relational databases. *An AmbySoft Inc. White Paper*. 1997. Dostupné z: `http://jeffsutherland.com/objwld98/mappingobjects.pdf`.

[18] BECK, K. *Test-driven development: by example*. Addison-Wesley Professional, 2003. ISBN 978-0321146533.

[19] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*. 1970, 13, 6, s. 377–387.

[20] CONRAD, T. PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need. 05 2004.

[21] DUMAS, J. S. – DUMAS, J. S. – REDISH, J. *A practical guide to usability testing*. Intellect books, 1999. ISBN 978-1841500201.

[22] FOWLER, M. Reducing coupling. *IEEE Software*. 2001, , 4, s. 102–104.

[23] FOWLER, M. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 978-0321127426.

[24] HATTON, S. Choosing the right prioritisation method. In *19th Australian Conference on Software Engineering (aswec 2008)*, s. 517–526. IEEE, 2008.

[25] MARTIN, R. C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009. ISBN 978-0132350884.

[26] MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Ö'Reilly Media, Inc.", 2011. ISBN 978-1449310509.

[27] NEUSTADT, I. – ARLOW, J. *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, Albatros Media as, 2016. ISBN 978-802511503.

[28] OSHEROVE, R. *The Art of Unit Testing: With Examples in. Net*. Manning Publications Co., 2009. ISBN 978-1617290893.

[29] RUMBAUGH, J. – JACOBSON, I. – BOOCH, G. *The unified modeling language reference manual*. Pearson Higher Education, 2004. ISBN 978-0201309980.

[30] SHNEIDERMAN, B. et al. *Designing the user interface: strategies for effective human-computer interaction.* Pearson, 2016. ISBN 978-0321537355.

[31] SOMMERVILLE, I. *Software engineering, Global Edition.* Pearson, 2016. ISBN 978-1292096131.

[32] SONG, I.-Y. – FROEHLICH, K. Entity-relationship modeling. *Potentials, IEEE.* 02 1995, 13, s. 29 – 34. doi: 10.1109/45.464652.

[33] WATSON, R. T. *Data management, databases and organizations.* John Wiley & Sons, 2008. ISBN 978-1943153039.

Dodatok A

Zoznam použitých skratiek

API Application Programming Interface

AWT Abstract Window Toolkit

CRUD Create Read Update Delete

CSS Cascade Style Sheets

HTML Hypertext Markup Language

MVC Model-view-controller

UML Unified Modeling Language

Dodatok B

Obsah priloženého CD

```
/implementacia
        /app
        /artisan
        /bootstrap
        /composer.json
        /composer.lock
        /config
        /database
        /.editorconfig
        /.env.example
        /.gitignore
        /package.json
        /phpunit.xml
        /public
        /resources
        /routes
        /server.php
        /storage
        /tests
        /vendor
        /webpack.mix.js
        /yarn.lock

/text
        /analyza.tex
        /csplainnat.bst
        /figures
        /implementacia.tex
        /k336_thesis_macros.sty
        /Makefile
        /navrh.tex
        /obsah-cd.tex
```

```
/README.md
/reference.bib
/testovanie.tex
/uvod.tex
/vachuric-thesis-2019.nls
/vachuric-thesis-2019.pdf
/vachuric-thesis-2019.synctex.gz
/vachuric-thesis-2019.tex
/zaver.tex
```