VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

REENGINEERING SKLADOVÉHO SYSTÉMU PRODEJCE SPORTOVNÍHO VYBAVENÍ

SPORTS EQUIPMENT RETAILER STORAGE SYSTEM REINGENEERING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. RADIM SVÁČEK**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/20218/2016/xsvace02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Sváček Radim, Bc.**
Obor: Management a informační technologie
Téma: **Reengineering skladového systému prodejce sportovního vybavení**
Sports Equipment Retailer Storage System Reeingeneering
Kategorie: Informační systémy

Pokyny:

1. Seznamte se stávajícím systémem managementu a s procesy, které jsou využívány ve firmě UniPortal s.r.o., zaměřte se na skladové hospodářství.
2. Po konzultaci s konzultantem firmy a s odpovědnými osobami za firemní procesy, specifikujte požadavky na procesy skladového hospodářství a skladové logistiky ve firmě.
3. Vytvořte dokumentaci pro návrh rozhraní skladového systému, které bude sloužit pro implementaci backend logiky.
4. Zvolte vhodné vývojové prostředí, vybrané po dohodě s konzultantem firmy a implementujte prototyp navrženého systému.
5. Na vzorku dat, vybraném po dohodě s vedoucí, ověřte správnost řešení. Na základě výsledků uživatelských testů a po konzultaci s konzultantem firmy, případně realizované řešení optimalizujte.
6. Zhodnoťte dosažené výsledky a diskutujte další možná vylepšení nástroje.

Literatura:

- BÖHMER, Marian. *Návrhové vzory v PHP:* 1. vyd. Brno Computer Press, 2012. ISBN 978-80-251-3338-5.
- ROSING, Mark Von. *The Complete Business Process Handbook*: 1st edition. Waltham Elsevier, 2014. ISBN 9780127999593.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kreslíková Jitka, doc. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je analyzovat procesy skladového hospodářství firmy, optimalizovat je a vytvořit backend skladového informačního systému. Aplikace slouží k zajištění příjmu, expedici a evidenci skladových zásob. Systém dále umožňuje komunikaci s dopravci skrze webové služby. Vyvíjen byl v jazyce PHP s využitím Nette, Slim a Doctrine. Aplikace byla úspěšně implementována a otestována.

Abstract

Goal of this thesis is to analyze processes in company's warehouse, optimize it and create backend of warehouse's information system. Application aim to evidence income goods, stock and expedition. System allows comunicate with web services of delivery companies. It was implemented in PHP with use of Nete, Slim and Doctrine. Application was successfuly implemented and tested.

Klíčová slova

reinženýring, skladový systém, business procesy, informační systém, php, nette, slim, doctrine, apiary, webové služby, ddd, cqrs, backend

Keywords

reengineering, warehouse system, business process, information system, php7, nette, slim, doctrine, apiary, web services, ddd, cqrs, backend

Citace

SVÁČEK, Radim. *Reengineering skladového systému prodejce sportovního vybavení.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kreslíková Jitka.

Reengineering skladového systému prodejce sportovního vybavení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní doc. RNDr. Jitka Kreslíková, CSc. Další informace mi poskytli pan Mgr. Zbyněk Kubičík a firma SportObchod.cz s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radim Sváček
20. května 2017

Poděkování

Rád bych na tomto místě poděkoval vedoucí mé diplomové práce paní doc. RNDr. Jitce Kreslíkové, CSc. za vedení mé práce, poskytnuté rady a čas, který mi věnovala. Dále bych rád poděkoval panu Mgr. Zbyňku Kubičíkovi za rady, náměty a konzultace při vedení práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvoření informačního systému pro skladové hospodářství firmy SportObchod.cz s.r.o. Jedná se o implementaci backend části webové aplikace, jež nabídne možnost evidence zboží od jeho příjmu, naskladnění, vyhledání až po expedici. Systém také umožňuje avizovat vybraným dopravcům odesílání balíků, což je implementováno pomocí napojení na webové služby jednotlivých dopravních společností.

Čtenář je v kapitole 2 seznámen s obecnou teorií podnikových procesů. Dále je tato kapitola věnována představení technologií, které byly použity při implementaci této aplikace. Následně jsou blíže popsány postupy a standardy použité při implementaci, jako je využití REST api či domain-driven design.

Kapitola 3 obsahuje specifikaci funkčních i nefunkčních požadavků na aplikaci. Důraz byl kladen především na identifikaci procesů probíhajících ve skladu při běžném provozu, přičemž právě tyto procesy se staly společně s identifikovanými aktéry a komponenty základem pro návrh informačního systému.

Právě návrhu aplikace se věnuje kapitola 4. Jelikož se jedná o implementaci backend aplikace s využitím REST api rozhraní, byla velká pozornost krom návrhu architektury věnována i návrhu kvalitního a srozumitelného rozhraní pro komunikaci.

Kapitola 5 se věnuje samotné implementaci, stejně jako řešení problémů, které při implementační části vznikaly. Blíže je popsána implementace jednotlivých komponent systému, stejně jako moduly pro komunikaci s webovými službami dopravců.

Předposlední kapitola 6 se věnuje testování vytvořeného informačního systému. Během vývoje bylo využito jednotkového a systémového testování, čímž se ověřila základní funkčnost aplikace. Uživatelským testováním poté byla dokázána správnost aplikace.

V závěrečné kapitole 7 jsou zhodnoceny dosažené výsledky, včetně informací z reálného provozu nasazené aplikace. Dále jsou zde rozebrány možnosti budoucího rozvoje informačního systému.

Tato diplomová práce navazuje na stejnojmenný semestrální projekt, který byl vypracován v rámci zimního semestru akademického roku 2016/2017. Z něj byla převzata kapitola 2, s částečnými úpravami pak i kapitoly 3 a 4.

Kapitola 2

Teorie pro vývoj skladového informačního systému

V úvodu této části práce je popsáno modelování business procesů, základní teorie o tomto odvětví a představení používaných nástrojů. V další části textu je tato kapitola také věnována úvodu do informačních systémů, včetně popisu technologií, postupů a standardů pro efektivní návrh a implementaci informačního systému.

2.1 Podnikové procesy a jejich modelování

Na následujících řádcích je nejprve popsán pojem podnikový proces včetně jejich rozdělení. V druhé části je úvod do teorie modelování podnikových procesů a představení nejznámějších nástrojů pro modelování. U nástroje ARIS je navíc představeno několik modelů podrobněji.

2.1.1 Podnikový proces

V různých publikacích lze nalézt odlišné definice pojmu podnikový proces. Jedna z nich praví, že podnikový proces je chápán jako souhrn činností přeměňující za pomoci lidí a dalších nástrojů vstupy na určité výstupy, které mají hodnotu pro zákazníky nebo jiné procesy. Podnikový proces se provádí za účelem splnění určitého podnikového cíle (business goal). Jeho provádění je časově ohraničeno, přičemž pro jeho zahájení musí platit vstupní podmínky a jeho ukončení je charakterizováno dosažením určitých koncových podmínek [15]. Jako proces tak lze označit libovolnou operaci, která probíhá v rámci podniku.

Proces se skládá z uspořádaných činností (kroků) a lze jej dekomponovat na subprocesy a aktivity. Aktivity v rámci procesu můžeme klasifikovat do tří skupin, a to manuální aktivity, aktivity uživatelských interakcí a systémové aktivity. Manuální aktivity mohou být vykonány pouze lidmi, nikoliv informačním systémem. Kooperace lidské síly a informačního systému je zapotřebí při aktivitě uživatelských interakcí, zatímco systémové aktivity jsou zajištěny jen informačním systémem [15].

Automatizovaný proces můžeme označit jako workflow. Workflow je automatizace celého nebo části podnikového procesu, během kterého jsou předávány dokumenty, informace nebo úkoly od jednoho účastníka k druhému dle procedurálních pravidel [1].

Typy procesů

Podnikové procesy lze rozdělit do tří hlavních kategorií. Prvním typem jsou procesy hlavní, které tvoří jádro činnosti. S nimi pak souvisí procesy podpůrné a řídící.

Hlavní procesy Nejdůležitější procesy ve firmě, protože právě hlavní procesy přinášejí přidanou hodnotu a vytváří konečný produkt, který má pro zákazníka hodnotu. Hlavní procesy mají za úkol naplnění strategických cílů firmy. Obecně jsou tyto procesy navenek viditelné a snadno identifikovatelné, nicméně jsou většinou velmi komplikované.

Řídící procesy Představují procesy nutné pro chod firmy, přičemž samy nepřinášejí zisk. Jedná se o činností související s definováním strategie a cílů firmy a jejich dosažením. Patří sem tak především stanovení cílů, plánování či alokace zdrojů.

Podpůrné procesy Stejně jako procesy řídící, tak i podpůrné procesy negenerují firmě přímý zisk. Přesto jsou velmi důležité, neboť jsou na nich závislé procesy hlavní. Výstupem podpůrných procesů je totiž tvorba podmínek podporujících funkce hlavních procesů. Můžeme zde zařadit například kontrolu jakosti či nákup materiálu.

2.1.2 Modelování podnikových procesů

Výsledkem modelování podnikových procesů je jejich formalizovaný popis toho, co se ve firmě ve skutečnosti děje. Modelování dává ucelený přehled o struktuře a činnosti podniku a navíc dává prostor k optimalizaci činnosti. Výsledný model musí nutně odpovídat realitě.

K vytvoření takovéhoto modelu lze využít mnoho nástrojů. V této práci si blíže představíme tři z nich, a to UML, BPMN a ARIS.

UML

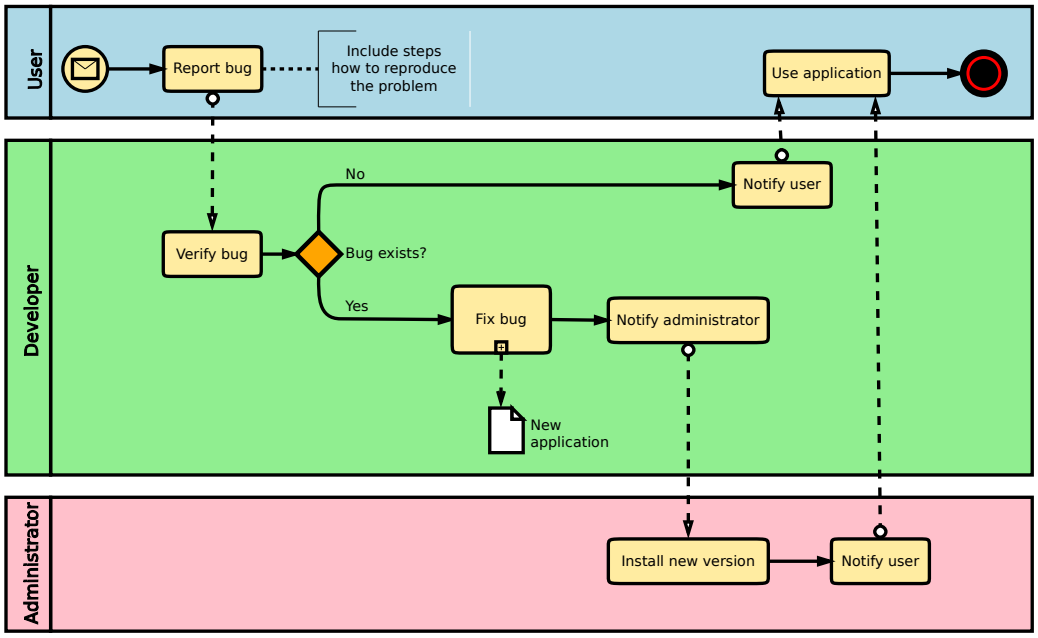
Hlavním posláním jazyka UML (Unified Modelling Language) byla především podpora vývoje objektově orientovaných systémů, nicméně dnes lze tento jazyk využít v mnohem širším spektru možností. UML se skládá z celé řady diagramů rozdělených do tří skupin - diagramy chování, diagramy interakce a diagramy struktury. Pro modelování procesů lze snadno využít diagram aktivit.

Ten je založen na popisu toku činnosti pomocí aktivit a přechodů mezi nimi. Obsahuje startovací a ukončovací stav, přechody, rozhodovací bloky a aktivity.

BPMN

Business Process Model and Notation (BPMN) je soubor principů a pravidel pro modelování podnikových procesů. Hlavním cílem je poskytnutí takové notace pro popis procesu, která bude čitelná pro všechny účastníky procesu. Výchozím diagramem je Business Process Diagram (BPD), který vychází z vývojových diagramů a skládá se z několika částí.

Prvním jsou plovoucí objekty, které mohou být reprezentovány aktivitou, událostí nebo branou pro rozdělování či slučování toku. Dále lze využít propojovací objekty několika druhů. Třetí částí jsou takzvané dráhy (swimlanes), které vizuálně oddělují odpovědnosti za aktivity. Poslední části jsou pak artefakty pro rozšíření základních elementů. Příklad takového diagramu lze vidět na obrázku 2.1.



Obrázek 2.1: Příklad Business Process Diagram (převzato z [12])

ARIS

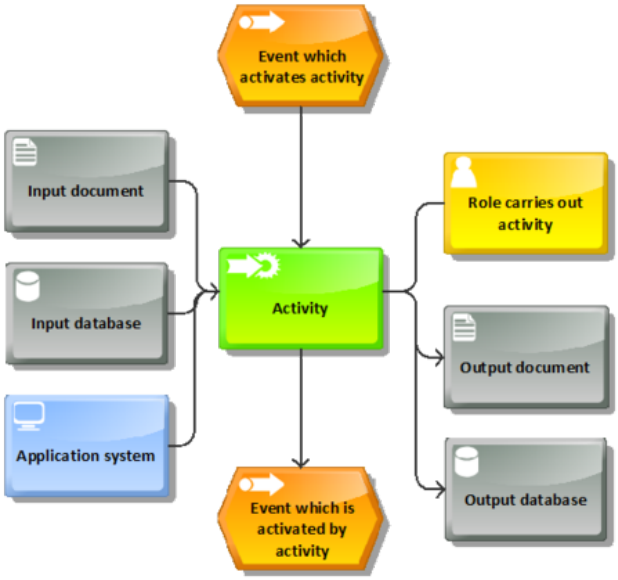
Architecture of Integrated Information Systems (ARIS) je koncept pro modelování podnikových procesů, jejich zdokumentování a optimalizaci. Nabízí čtyři pohledy na modelovanou společnost - organizační, datový, funkcionální a procesní.

Organizační pohled nabízí statický model struktury firmy. Obsahuje lidi, role, funkce apod. Datový pohled ukazuje statické modely podnikových informací a zahrnuje datové modely, soustavu vědomostí či nosiče informací. Funkcionální pohled zobrazuje statické modely úkonů procesů a zahrnuje hierarchii funkcí, podnikové cíle, podpůrné systémy a softwarové aplikace.

Procesní pohled ukazuje chování procesů a to, jak souvisí se zdroji, daty a funkcemi podnikového prostředí. Lze jej považovat za nejdůležitější z nabízených pohledů. Při jeho tvorbě nabízí ARIS širokou škálu diagramů.

EPC Procesní řetězec řízený událostmi (Event-Driven Process Chains) představuje jednu z nejpoužívanějších metod modelování podnikových procesů. Podstata tohoto modelu tkví v řetězení aktivit a událostí do jediné posloupnosti, jejímž cílem je realizovaný výstup procesu. Navíc lze využít i logické spojky ke slučování či rozdělování toku. Příklad takového diagramu lze vidět na 2.2.

Model tvorby přidané hodnoty Je určen pro zobrazení přehledu procesů a jejich návaznosti. Lze ho také využít jako model reprezentující hierarchii procesů nebo k vytvoření mapy přehledu všech procesů.



Obrázek 2.2: Příklad diagramu EPC (převzato z [7])

2.2 Implementační standardy a postupy

Následující sekce je věnována představení některých postupů a standardů, které budou využity při implementaci skladového informačního systému. Pro návrh bylo využito paradigma DDD (Domain Driven Design), při implementaci pak byl využit vzor CQRS (Command Query Responsibility Segregation). S okolím pak backend systém komunikuje skrze REST api (Representational State Transfer).

Na začátek je třeba vyjasnit dvě základní části webového informačního systémů, a to backend a frontend. Pojem frontend slouží k označení části webové aplikace, která je viditelná běžným návštěvníkům. Jedná se tedy o grafický návrh webové stránky včetně implementace pro zobrazení dat v internetovém prohlížeči. Naproti tomu backend pro běžného návštěvníka viditelný není. Běžně je backend aplikace provozována na serveru spolu s databází. Backend poskytuje data a logiku aplikace, díky čemuž může být výsledná webová stránka dynamická, zatímco frontend takto získaná data prezentuje, z čehož plyne, že mezi sebou tyto dvě na sobě nezávislé části komunikují [10].

2.2.1 DDD

Domain Driven Design (DDD) je filozofie návrhu software navržena k zvládnutí vytvoření aplikací pro komplexní doménu. Jedná se o kolekci návrhů, principů a praktik, které můžou být využity při návrhu SW [5]. Poprvé byl tento pojem zaveden Ericem Evansem v [2].

V rámci fáze vývoje software je hlavním cílem DDD část analýzy a návrhu, nicméně částečně zasahuje i do fáze implementace. Pro efektivní vývojový proces je nutné propojení těchto fází, jelikož v každé z nich lze odhalit jiné detaily o doméně při pohledu z jiné perspektivy.

Pod pojmem doména se skrývá oblast, kterou má vyvíjená aplikace řešit. Tím pádem musí být aplikace obrazem domény, proto musí být postaven na hlavních principech, konceptech a prvcích domény a jejich vztazích. V tu chvíli přichází řada na vytvoření modelu domény, tedy abstrakci reality.

Hlavními předpoklady doménou řízeného návrhu je primární zaměření na jádro domény a její logiky, vytvoření komplexního návrhu na základě modelu domény a spolupráce mezi programátory a doménovými experty při iterativním upřesňování modelu [2].

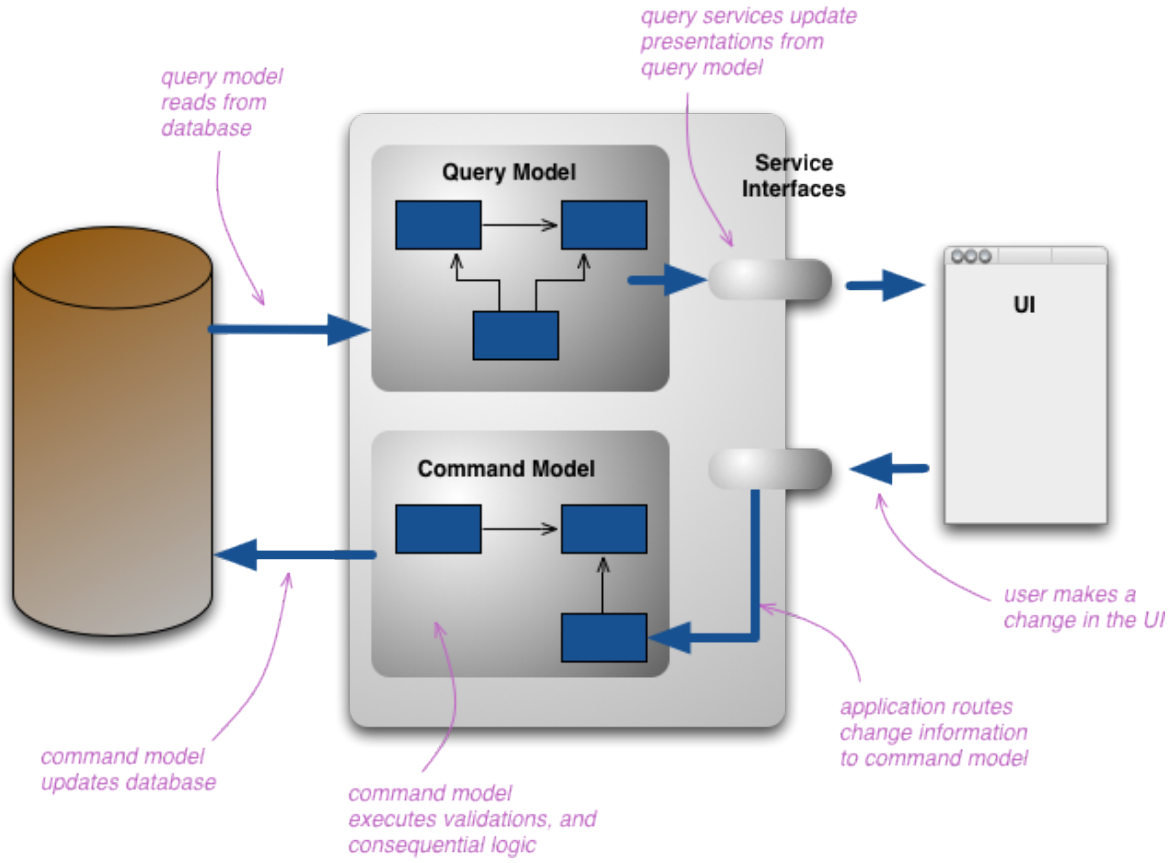
2.2.2 CQRS

Command Query Responsibility Segregation v praxi znamená rozdělení doménového modelu na dva modely, a to model čtecí a model zápisový. CQRS využívá Princip jedné odpovědnosti (single responsibility principle) na úrovni modelu a redukuje komplikovanost systému [5].

CQRS lze s výhodou využít pro systémy nad komplexními doménami [3]. Backend systému komunikuje s webovým UI přes definované rozhraní, přičemž jednotlivé požadavky obslouží Query nebo Command.

Query Query je HTTP GET požadavek, který UI zašle, kdykoliv potřebuje číst informace. Query nikdy nesmí změnit stav systému, jen vrátit data. Příkladem Query může být získání seznamu uživatelů.

Command Command je HTTP POST, PUT či DELETE požadavek, který se pokouší změnit stav systému. Příkladem může být Command pro vytvoření nového uživatele, jeho editaci či smazání. Command vrací návratový kód, dále může obsahovat hlavičku s lokací nové entity a tělo s jeho identifikátorem.



Obrázek 2.3: Command Query Responsibility Segregation (Převzato z [3])

2.2.3 REST

V roce 2002 REST (Representational State Transfer) poprvé popsal Roy Fielding, jeden z autorů specifikace HTTP, ve své disertační práci. Jedná se o architekturu rozhraní pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být například data, přičemž každý zdroj má svůj vlastní identifikátor URI (Uniform Resource Identifier). REST definuje čtyři HTTP metody pro práci se zdroji - GET, POST, PUT a DELETE [9].

Obecně platí, že REST má menší nároky na software na straně klienta než jiné přístupy, protože standardní webový prohlížeč stačí k přístupu k libovolné aplikaci a zdroji dat [11].

GET Jedná se o metodu pro získání zdroje. Jedná se o klasický HTTP požadavek na stránku, který je běžně využíván. Hlavička přitom může navíc obsahovat informace o požadované formě odpovědi. Nikdy ovšem metoda GET nesmí být implementována s vedlejšími účinky (side-effects), tedy se musí jednat jen o operaci čtení (read-only) [9].

POST Metoda pro vytvoření dat, která je známá z použití v HTML formulářích. K vytvoření dat je potřeba zavolat zdroj s URI pomocí HTTP metody POST, která ve svém těle obsahuje data nového zápisu. Po odeslání by měl server vrátit patřičný návratový kód – HTTP má k tomu účelu stavový kód 201 – Created, v němž lze předat URI nově vytvořeného zdroje. Pokud došlo k chybě, měl by server vrátit chybový kód.

PUT Slouží pro editaci zdroje, kdy je vlastnost zdroje nahrazena tou, kterou klient zaslal v těle zprávy. Ve chvíli volání metody PUT je již znám identifikátor zdroje. Neexistuje-li zápis s takovým identifikátorem, je tento zápis vytvořen podobně jako u metody POST [9].

DELETE Zdroj lze pomocí volání URI HTTP metodou DELETE smazat. Nic více není pro smazání vykonat.

2.3 Použité technologie a nástroje

V této části práce jsou blíže představeny technologie a nástroje využité k implementaci skladového informačního systému. Základem je jazyk PHP, přičemž bylo využito hned několik aplikačních rámců (framework), jako jsou například Nette, Slim či Doctrine. Pro vytvoření dokumentace rozhraní systému byl použit nástroj Apiary.

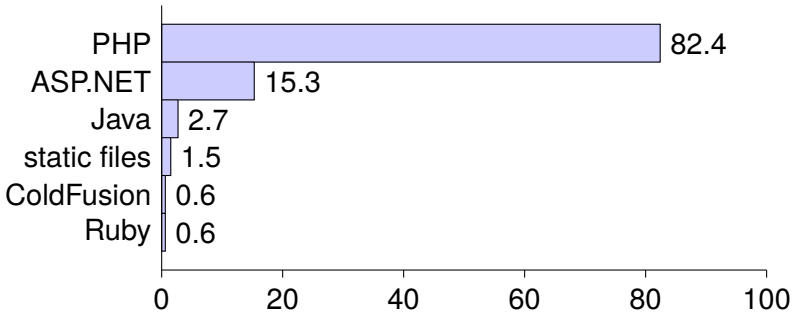
2.3.1 PHP

Skriptovací programovací jazyk určený především pro implementaci dynamických webových stránek. V práci byl použit ve stabilní verzi 7.1.0, jež vyšla 1. prosince 2016. Autorem jazyka PHP[1] (PHP: Hypertext Preprocessor, dříve Personal Home Page) je Rasmus Lerdorf. V prosinci 2016 mělo PHP podíl 82.4% mezi jazyky používanými pro tvorbu webu [13].

2.3.2 Nette

Jedná se o aplikační rámec s otevřeným kódem pro tvorbu webových aplikací v jazyce PHP. Původním autorem je David Grudl, nicméně o jeho rozvoj se stará Nette Foundation [6]. Eli-

[1] Oficiální stránky: https://php.net/



Obrázek 2.4: Podíl server-side programovacích jazyků (Převzato z [13])

minuje výskyt bezpečnostních děr a disponuje ladícími nástroji. Framework Nette[2] využívá konceptu Model-View-Controller.

2.3.3 Slim

Slim[3] je mikro framework pro jazyk PHP. Je využíván pro vývoj API a služeb vyhovujících pravidlům REST. Slim přichází s takovými schopnostmi, jako jsou směrování URL, HTTP cache na straně klienta, šifrování relace cookie, a podporuje také rychlé notifikace (flash messages) přes požadavky HTTP [4].

2.3.4 Doctrine 2

Doctrine 2[4] je ORM (Object-Relational Mapping) framework pro jazyk PHP. Jeho úkolem je zajistit mapování objektů na relační databázi. Díky tomu je možné pracovat s daty z databáze jako s tradičními objekty. Důležitou vlastností aplikačního rámce Doctrine je psaní databázových dotazů použitím Doctrine Query Language (DQL), který vychází z ORM frameworku Hibernate.

2.3.5 Apiary.io

Apiary.io[5] je nástroj pro zdokumentování rozhraní. Apiary pomáhá především ve třech oblastech: tvorbě dokumentace, vytvoření prototypu API a testování API. Pro popis rozhraní využívá API Blueprint. Po vytvoření dokumentace je k dispozici prototyp API, který bude na HTTP požadavky odpovídat podle pravidel definovaných v dokumentaci.

2.3.6 EAN

Kód EAN (European Article Number) je jednotná mezinárodní číselná identifikace zboží, jež se tiskne na štítky zboží ve formě čárového kódu. Nejpoužívanějším typem, jež se využívá i ve skladě prodejce, je typ EAN-13. První dvě číslice označují zemi, po pěti číslicích připadá na výrobce a interní označení produktu výrobcem, poslední číslice je pak kontrolní [14].

[2] Oficiální stránky: https://nette.org/
[3] Oficiální stránky: https://www.slimframework.com/
[4] Oficiální stránky: http://www.doctrine-project.org/
[5] Oficiální stránky: https://apiary.io/

2.4 Aktuální stav skladového systému

V době návrhu a implementace funguje ve skladu informační systém pokrývající základní požadavky vedení. Nicméně z důvodu rozšíření požadavků na systém a optimalizaci systému byla zvolena varianta vývoje nového IS zcela od základů. Motivací pro nový systém je i jeho napojení na webové služby jednotlivých dopravců pro zrychlení expedice balíků ze skladu.

Hlavním kamenem úrazu stávajícího systému je velmi pomalá odezva na požadavky uživatele, kdy především při procesu příjmu rychlost systému způsobuje problémy. Problém je zároveň s procesem vyhledávání objednávek, ve kterém nelze vyhledávat více objednávek zároveň či některé objednávky upřednostňovat.

Stávající systém je implementován v jazyce PHP, přičemž UI je vytvořeno pomocí jednoduchých HTML stránek s formuláři pro zadávání vstupů. Některé ze vstupů lze zadávat přímo čtečkou, přičemž jsou tato data automaticky vloženy do vstupních polí.

Kapitola 3

Specifikace požadavků

Cílem této kapitoly je specifikace požadavků na skladový systém prodejce sportovního vybavení. Nejdříve je neformálně popsán systém a požadavky na něj. Poté jsou detailně analyzovány procesy probíhající v prostředí reálného skladu. V další části jsou pak identifikováni aktéři, kteří budou se systémem pracovat, následně jsou popsány jednotlivé komponenty systému. Nakonec je stručný popis nefunkčních požadavků na systém.

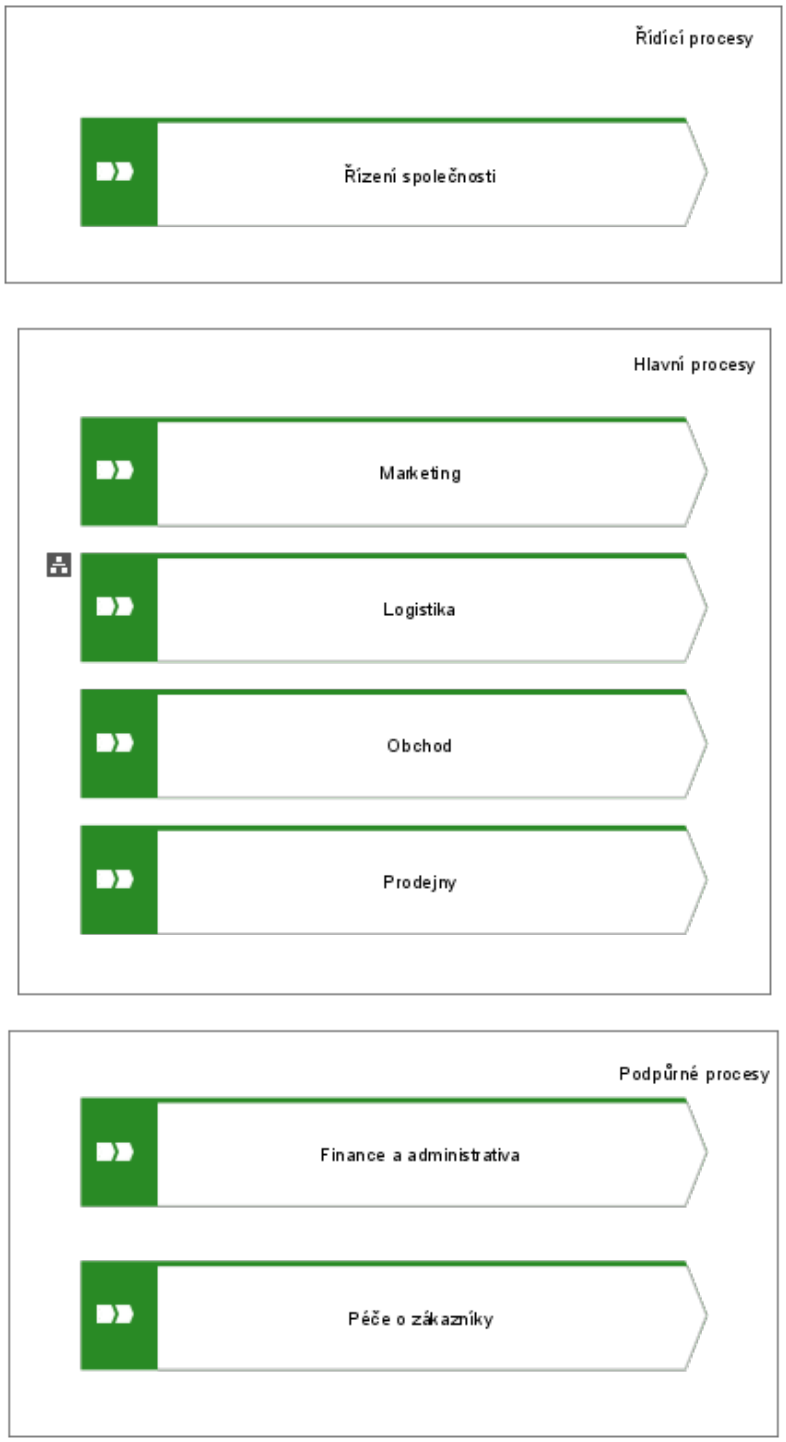
3.1 Neformální specifikace

Účelem této práce je analyzovat, navrhnout, implementovat a nasadit skladový systém pro firmu zabývající se prodejem sportovního vybavení, přesněji backend tohoto systému. Jedná se tedy o informační systém, jež poběží na privátním serveru.

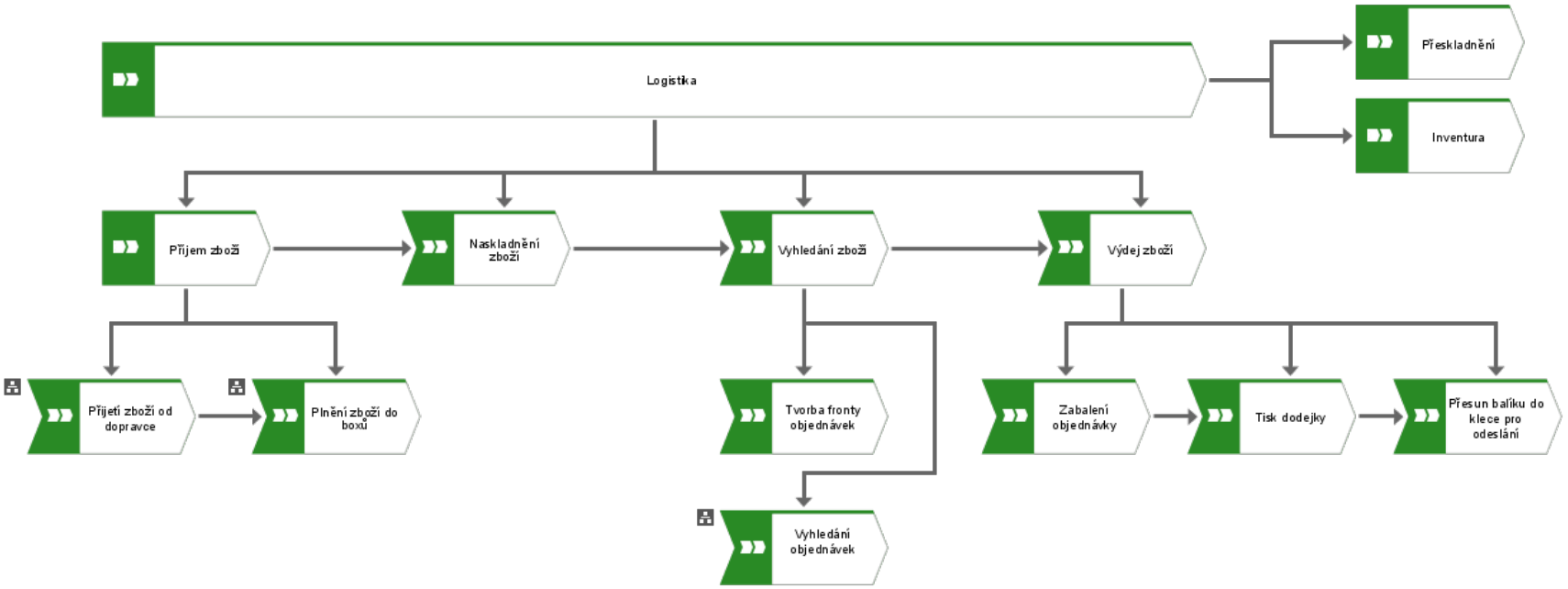
Hlavním úkolem tohoto systému je usnadnění, zrychlení a zaznamenání pohybu zboží ve skladu, a to od jeho přijetí od dodavatele, přes naskladnění, vyhledání, zabalení až po expedici k zákazníkovi. Důležitou součástí systému bude i možnost inventarizace skladu či ocenění skladu.

Životní cyklus jednoho produktu ve skladu začíná jeho naskladněním, kdy je načten jeho EAN kód, který jej identifikuje. Poté je produkt umístěn do jednoho z úložných boxů. Jakmile vznikne v systému elektronického obchodu objednávka, ve které se může tento produkt objevit, je tato objednávka propagována do skladového systému. Poté je produkt nalezen pracovníkem vyhledávání a přemístěn na pracoviště expedice, kde jej další pracovník společně s ostatními produkty dané objednávky zabalí, načež je zboží expedováno k zákazníkovi.

Se systémem budou pracovat zaměstnanci skladu, kteří ke komunikaci se ním budou využívat mobilní zařízení a čtečky kódů EAN. Uživatelé budou rozděleni do několika rolí, které jim umožní provádět různé operace nad systémem. Nejvíce možností bude mít vedoucí skladu. Jednotliví podřízení pak mají přístup ke specifickým operacím, které souvisí s výkonem jejich práce. Například pracovník expedice, která má na starost balení objednávek a jejich předání dopravci, může v systému vytvářet nové balíky či označit objednávku jako odeslanou.



Obrázek 3.1: Přehled procesů ve firmě [vlastní]



Obrázek 3.2: Procesy skladového hospodářství [vlastní]

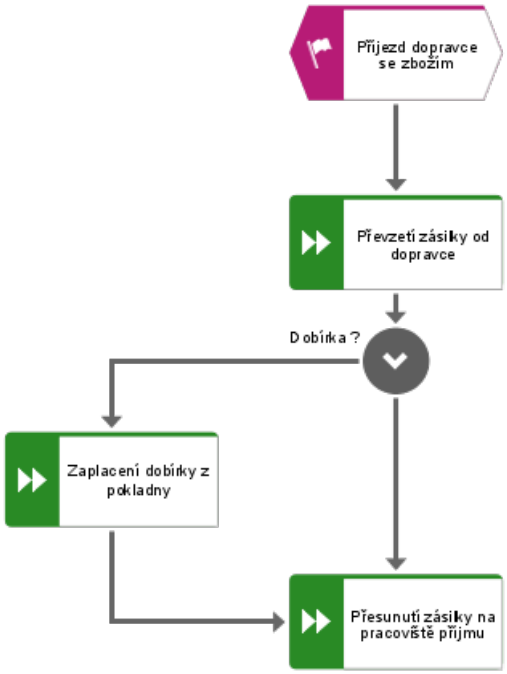
3.2 Identifikace procesů

Základní přehled procesů firmy lze vidět na obrázku 3.1. Krom logistiky, která je v následující části této práce podrobně analyzována, patří mezi další hlavní procesy například marketing či obchodní činnost. Samostatnou částí jsou pak podpůrné procesy a řídící proces společnosti.

Procesy probíhající v reálném skladu jsou základním kamenem pro návrh užitečného systému, proto je této části textu věnována velká pozornost. Na diagramu 3.2 můžete vidět základní identifikované procesy logistiky včetně vybraných podprocesů. Následující řádky jsou věnovány bližší analýze čtyř základních procesů, tedy příjmu, naskladnění, vyhledání a expedici zboží.

3.2.1 Příjem zboží

Samotný proces příjmu zboží, jak je vidět na obrázku 3.2, se skládá ze dvou podprocesů, kdy je nejprve zboží přebráno od dopravce, načež jsou jednotlivé produkty umísťovány do úložných boxů.



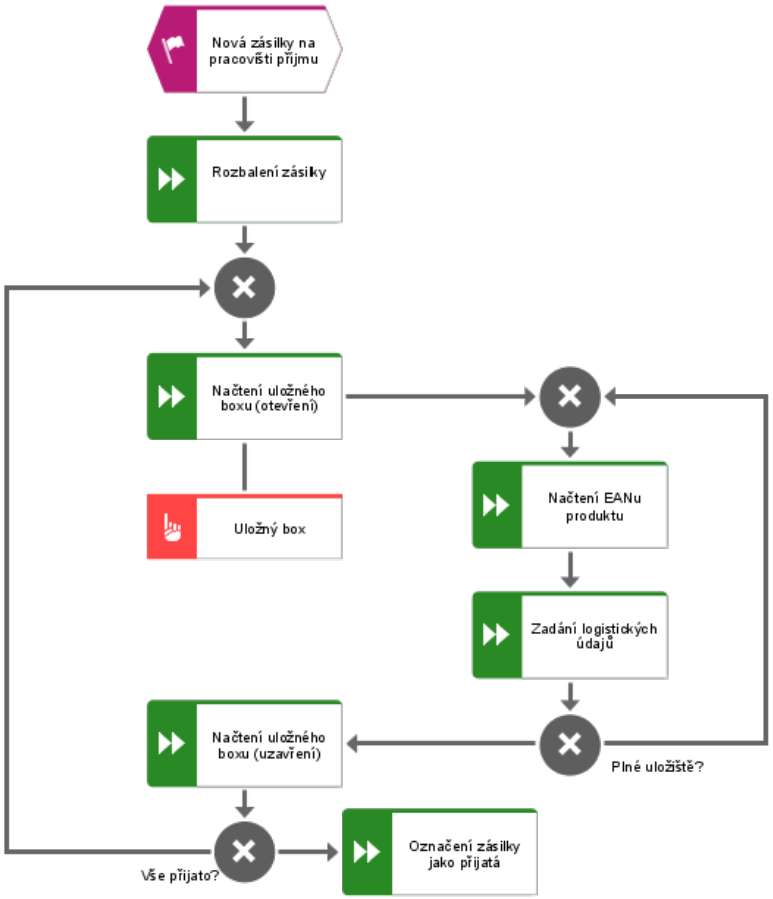
Obrázek 3.3: Proces přijetí zboží [vlastní]

Prvním podprocesem je přijetí zboží od dopravce. Proces je vyvolán událostí doručení zásilky se zbožím vybraným dopravcem. V některých výjimečných případech může být zboží zasláno na dobírku, tudíž je nutné dopravci zaplatit hotově částku za přijaté zboží. Poté je již zboží, ať už ve formě palety, balíku či jiného balení, přesunuto z nákladové rampy skladu na pracoviště příjmu. Tento proces je znázorněn na diagramu 3.3.

Přímo na pracovišti příjmu probíhá druhý podproces příjmu zboží, a to plnění zboží do úložišť, který je zobrazen na diagramu 3.4. Ten je vyvolán událostí přesunutí zásilky na pracoviště příjmu. Poté pracovník nejprve rozbalí zásilku, načte první z úložišť a postupně

vkládá do tohoto boxu zboží. Vkládání zboží znamená nejprve načtení kódu EAN daného produktu a volitelně také zadání logistických údajů, nejsou-li prozatím v systému zadány.

Do úložiště pracovník vkládá zboží, dokud není naplněn, poté musí znovu načíst kód úložiště a označit jej jako naplněné. Takové úložiště je připraveno k naskladnění. Jakmile je veškeré zboží z dané zásilky rozřazeno do úložišť, je tato zásilka označena za přijatou.



Obrázek 3.4: Proces plnění zboží do úložných boxů [vlastní]

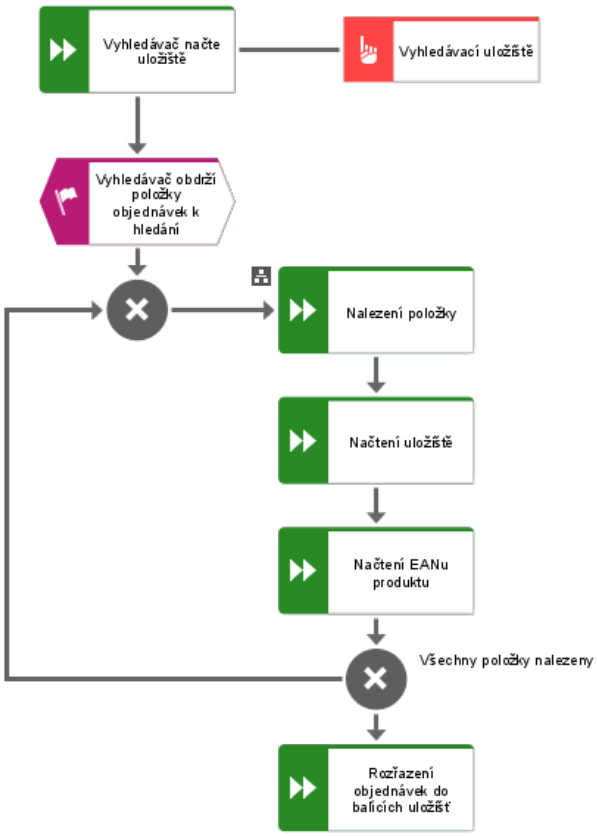
3.2.2 Naskladnění zboží

Druhý hlavní proces logistiky je tím nejjednodušším. Poté, co je přijata zásilka a jsou naplněny úložiště, jsou tato úložiště označena jako připravena k naskladnění. V tu chvíli může pracovník skladu tato úložiště umístit do libovolné pozice ve skladu, která je označena jako prázdná. Zařazení úložiště do pozice probíhá tak, že pracovník nejprve načte kód úložiště, zvolí možnost zařadit do pozice, načež načte kód pozice. Tato posloupnost úkonů má za výsledek zařazení úložiště do dané pozice. V tuto chvíli je tak již zboží, které je umístěno v zařazeném úložišti, možno vyhledávat pro expedici.

3.2.3 Vyhledání zboží

Proces vyhledávání zboží začne ve chvíli, kdy pracovník vyhledávání načte kód vyhledávacího úložiště. V ten okamžik systém vyhodnotí podmínky ve frontě objednávek a přidělí objednávky k vyhledání.

Pracovník vyhledávání má tak k dispozici seznam produktů, které je potřeba ve skladu vyhledat. Pokud pracovník nalezne zboží na přiděleném místě, načte kód vybraného úložiště, poté načte kód nalezeného zboží a zároveň jej přesune z úložiště do vyhledávacího úložiště.

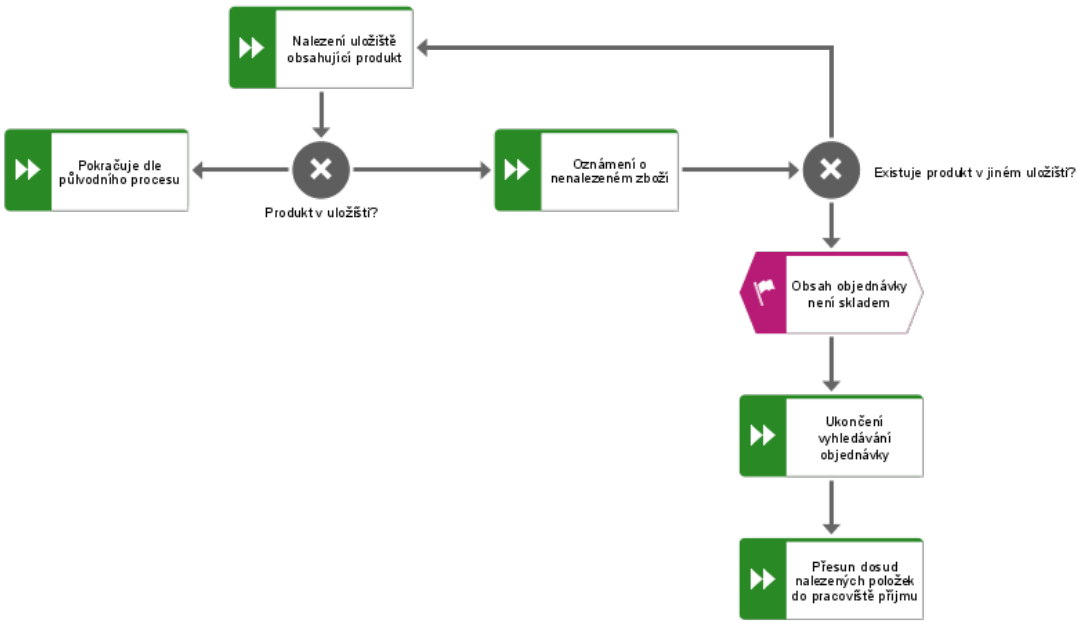


Obrázek 3.5: Proces vyhledání zboží [vlastní]

Poté, co pracovník nalezne veškeré hledané produkty, musí je rozřadit do balících úložišť, a to podle jednotlivých objednávek tak, jak mají být zabaleny a odeslány. Poté, co zboží takto rozřadí, je vyhledávací úložiště označeno jako prázdné a opět využitelné k vyhledávání, a zároveň se změní stav daných objednávek na stav *balení*. Celý proces je zobrazen na diagramu 3.5.

Podproces procesu vyhledání zboží objednávek je samotné nalezení jednoho kusu produktu v úložišti. Tento proces je zobrazen na diagramu 3.6. Zde se řeší případ, kdy hledaný produkt fyzicky není umístěn v úložišti tak, jak by dle systému měl být. V tom případě je systému oznámena nepřítomnost zboží, načež je možno vyhledat produkt v jiném úložišti ve skladu, pokud je ještě někde umístěn.

Není-li produkt již skladem, je objednávka označena stavem *není skladem* a vyhledávání této objednávky je ukončeno. Poté, co jsou vyhledány zbývající přidělené objednávky



Obrázek 3.6: Podproces pro případ nenalezeného zboží [vlastní]

dochází k rozřazování, přičemž produkty z nenalezené objednávky jsou umístěny do speciálního úložiště, jež je určeno k opětovnému naskladnění.

3.2.4 Balení a expedice zboží

Poslední ze čtyř hlavních logistických procesů probíhá na pracovišti balení a expedice. Výsledkem procesu vyhledání zboží je rozřazení jednotlivých objednávek do balících úložišť, odkud jednotlivé produkty odebírá pracovník balení. Tyto produkty zabalí do dostupných lepenkových balíků, přičemž jedna objednávka může být rozdělena do několika balíků.

Ve chvíli, kdy je vytvořen balík, je zasláno avízo zvolenému dopravci, čímž se u něj balík zaeviduje. Poté je ze systému dopravce získán štítek balíku, který se vytiskne a nalepí. Posledním krokem je pak fyzické přesunutí jednoho či více balíků do klece pro balíky specifického dopravce, čímž je změněn stav objednávky na *připraven k expedici*.

3.2.5 Podpůrné procesy

Krom hlavních procesů existují ve skladu i dva důležité podpůrné procesy, a to procesy inventura a přeskladnění.

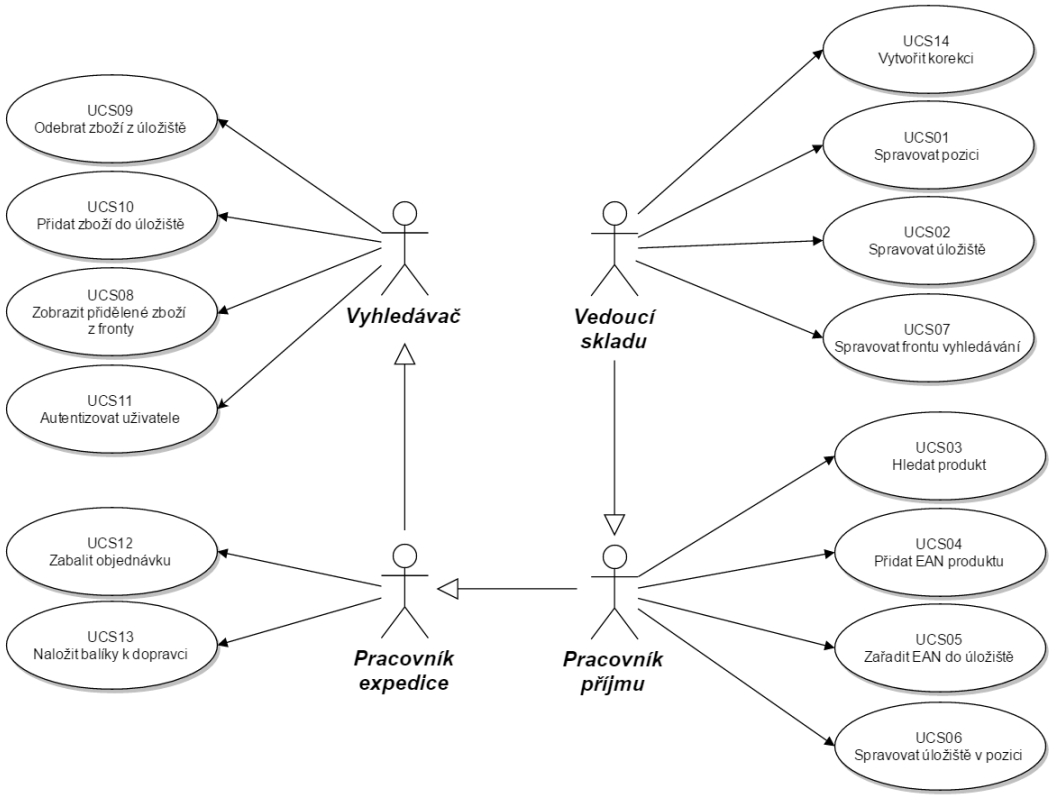
Inventura Jedná se o klasickou inventarizaci zboží, tedy kontrolu, zda jednotlivá úložiště obsahují to, co obsahovat mají. Pracovníci skladu tedy fyzicky zkontrolují úložiště určená k inventuře a opětovným načtením kódů všech produktů se zjistí skutečný stav úložiště. Výsledkem tak může být buďto chybějící či přebývající zboží.

Inventura nemusí probíhat nad celým skladem, ale jen nad jeho určitou částí. Rozsah kontroly určuje vedoucí skladu.

Přeskladnění Ve skladu často dochází k situaci, kdy mnoho úložišť obsahuje jeden či malé množství produktů, zatímco na pracovišti příjmu se hromadí zboží v úložištích, pro které neexistují volná úložiště. Proto existuje proces přeskladnění, kdy se obsah vybraných úložišť sloučí, čímž vzniknou prázdné pozice pro úložiště.

3.3 Identifikace aktérů

Ve skladu byli identifikováni čtyři aktéři. Use-case diagram aktérů včetně jejich oprávnění lze vidět na obrázku 3.7. Aktér s nejvyššími pravomocemi je vedoucí skladu, pod kterého spadá pracovník příjmu. Na nižších úrovních jsou pak pracovníci expedice a vyhledávání zboží. V následující části jsou jednotliví aktéři blíže analyzováni.



Obrázek 3.7: Případy užití logistického systému [vlastní]

3.3.1 Vyhledávač

Nejméně možností práce se systémem má aktér vyhledávač. Ten může jen odebírat a přidávat zboží z, respektive do úložiště. To obsahuje přesun zboží z úložného boxu do vyhledávacího úložiště a následně do balícího úložiště.

Krom toho si může zobrazovat zboží, které již nalezl a které má právě vyhledávat, včetně zobrazení pozic, kde lze zboží nalézt. Všechny tyto operace se systémem může tento aktér provádět až po autentizaci. Ta nepracuje s heslem, ale jen s EAN kódem aktéra na jeho visačce, která je mu přidělena při nástupu do zaměstnání.

3.3.2 Pracovník expedice

Oproti aktérovi vyhledávač může pracovník expedice přichystané produkty zabalit do jednotlivých balíků, což obnáší odebrání produktů z balícího úložiště, vložení do balíku a jeho zabalení. Poté je třeba balík ještě polepit štítky pro dopravce.

Následně již může balík fyzicky přesunout do klece pro dopravce, čímž je balík připraven k expedici a dopravce si jej může naložit a odvézt.

3.3.3 Pracovník příjmu

Pracovník příjmu nemá sice největší oprávnění, ale má pravděpodobně největší odpovědnost, protože dojde-li k chybě na příjmu, projeví se v celém životním cyklu produktu ve skladu. Při příjmu pracovník nejprve přijímaný produkt vyhledává v databázi načtením jeho EAN kódu. Pokud se jej nepodaří nalézt pomoci tohoto kódu, musí využít název produktu či jiné atributy a následně EAN kód produktu přiřadit.

Poté, co je produkt přijat, může jej zařadit do úložiště. Nejedná se o totožný případ užití, jako u aktéra vyhledávač, ale o prvotní plnění vyhledávacího boxu. Blíže je tento případ užití popsán v další části textu.

3.3.4 Vedoucí skladu

Aktér vedoucí skladu má největší pravomoci. Jako jediný může řešit problémy s chybějícím či přebývajícím zbožím v úložišti, a to díky provádění korekcí. Dále má vedoucí skladu možnost spravovat informace o pozicích a úložištích, lze z jeho pozice tedy reorganizovat fyzické uspořádání skladu.

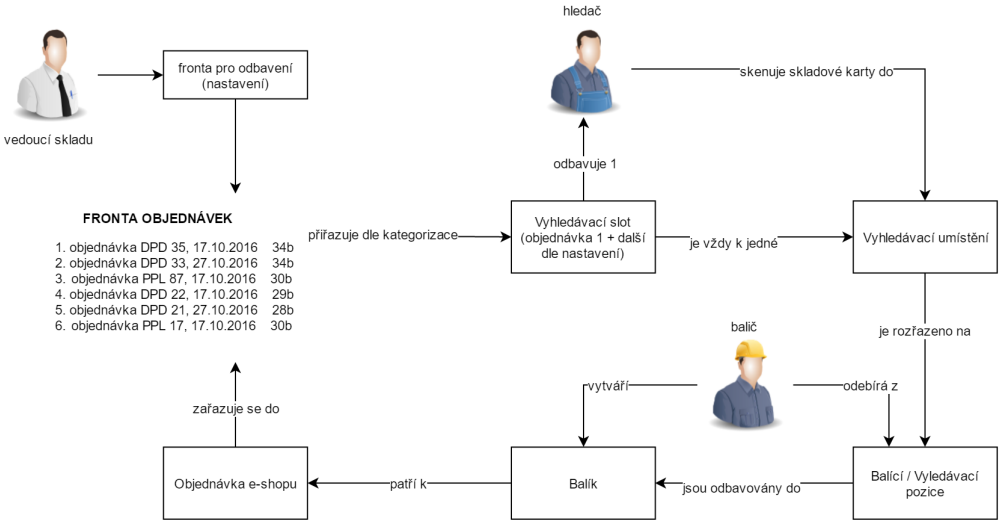
Další důležitým oprávněním je možnost spravovat frontu vyhledávání. Zejména v době špičky je třeba upřednostnit odbavení specifického typu objednávek. Může se jednat o případ, kdy je očekáván příjezd jednoho z dopravců, který si sváží balíky k odeslání. Vedoucí skladu tak má možnost upřednostnit odbavování objednávek pro tohoto dopravce.

3.4 Identifikace komponent

Na základě pozorování a získávání požadavků zadavatele bylo identifikováno několik základních komponent vznikajícího informačního systému. Základem je samotné zboží ve skladu, které je popsáno jako skladová karta. Další nedílnou součástí jsou úložiště, jež mohou obsahovat jednotlivé kusy zboží. Každé úložiště je pak umístěno v určité pozici, jež je určující pro vyhledávání zboží, které bude probíhat na základě vzniklé objednávky z elektronického obchodu firmy. Z objednávek pak na pracovišti expedice vznikají balíky. Krom toho jsou důležité i další entity, které by se daly zařadit do jádra sytému. Jedná se například o uživatele, dopravce, prodejní kanál či frontu objednávek. V následující části textu jsou tyto entity podrobně analyzovány. Na obrázku 3.8 lze vidět zapojení jednotlivých komponent do workflow skladového systému.

3.4.1 Pozice a úložiště

Základními jednotkami skladu pro umisťování zboží jsou pozice a úložiště, mezi nimiž existuje přímý vztah nadřazenosti. Pozice v reálném skladu reprezentuje část regálu či místo ve skladu, do kterého můžou být umístěna jednotlivá úložiště.



Obrázek 3.8: Workflow skladového systému [vlastní]

Úložiště reprezentuje objekt, který obsahuje jednotlivé kusy zboží, takzvanou skladovou zásobu v úložišti. Úložiště může být různého druhu, jedná se například o papírovou krabici, plastový box, či celou paletu. Krom toho existují speciální typy úložišť. Prvním z nich je vyhledávací úložiště, jež se používá pro nalezení položek objednávky a v reálném skladu jsou využívány nákupní košíky či vozíky. Dělí se na čtyři typy dle velikostních kategorií. Dalšími typy úložišť jsou balící, na které se umísťují kompletně nalezené objednávky či převodky určené k balení. Úložiště je jednoznačně reprezentováno kódem označující typ a pořadovým číslem.

I pozice může být různých typů, kdy základní jsou pozice pro umístění boxů a palet. Zvláštním typem je pozice pro umístění hokejek, jelikož tvar těchto produktů je značně odlišný oproti většině ostatních. Pozice je jednoznačně identifikována pomocí kombinace údajů o regálu, řadě a sloupci.

3.4.2 Skladová karta

Reprezentuje specifický produkt ve skladu. Každý produkt má minimálně jeden unikátní kód EAN, který byl popsán v předchozí kapitole, přičemž stejný produkt může obsahovat více těchto kódů.

Krom toho skladová karta musí obsahovat obrázek produktu, aby měli pracovníci skladu možnost porovnat načtený produkt se vzorem. Důležitou součástí skladové karty jsou i rozměry produktu a jeho balení, a to jak pro umístění na skladě, tak zejména pro dopravce, pro které jsou tyto údaje povinné. Každý produkt se také váže k určitému typu produktů a velikostní kategorii.

Pracovníci skladu musí mít možnost procházet seznam skladových karet či detail skladové karty včetně fyzického umístění jednotlivých kusů v úložištích. Podmínkou je zajištění importu produktů ze systému elektronického obchodu.

3.4.3 Objednávka

Stejně jako u skladové karty je třeba zajistit import objednávek ze systému elektronického obchodu, aby mohly být objednávky ve skladu odbaveny a expedovány.

Objednávka obsahuje informace o zákazníkovi a doručovací adrese. Dále se zásilka váže na vybraného dopravce a prodejní kanál. Každá objednávka má svůj stav odbavení.

Krom těchto základních informací obsahuje objednávka seznam položek. Každá položka je definována produktem, tedy skladovou kartou, a množstvím produktu, v tomto případě počtem kusů.

3.4.4 Balík

Každá objednávka může být zabalena do několika balíků, ovšem všechny balíky jsou poslány jako jedna zásilka. Balík tak obsahuje položky objednávky a počet kusů daného produktu. U balíku je třeba znát i čas zabalení (okamžik, kdy je balík připraven k expedici) a čas odeslání (okamžik, kdy si jej dopravce fyzicky převzal).

Krom toho má balík své unikátní číslo, kterým je reprezentován v systému dopravce. Právě pomocí tohoto čísla může pracovník skladu či jiný zaměstnanec s oprávněním či zákazník přes webový portál dopravce sledovat stav zásilky.

3.4.5 Sklady

Jednou z komponent skladového systému jsou i sklady. Jedná se o rozdělení na centrální sklad a sklad jednotlivých prodejen. Zboží se eviduje centrálně, nicméně každý uživatel má právo pracovat pouze nad vybraným skladem. Ke skladům jednotlivých prodejen se vážou i prodejní místa, kde může docházet k osobním odběrům zboží. Jednotlivé produkty lze mezi sklady přesouvat díky takzvané převodce, která se v systému chová jako klasická objednávka. Proto musí mít každý sklad svou adresu. Zároveň položky objednávky, která je vydávána pomocí osobního odběru, jsou automaticky odeslány na vybrané prodejní místo.

3.4.6 Účetní doklad

Tato komponenta se skládá z příjemek a výdejek a dohromady dává úplný přehled o aktuální skladové zásobě každého produktu. Příjemky vznikají při příjmu zboží, kdy se vážou k objednávce zásob z informačního systému prodejce. Tím se příjemce přiřadí nákupní cena. Z nákupních cen a skladové zásoby lze poté vypočítat váženou nákupní cenu produktu, stejně jako získat celkové ocenění skladu. Výdejky vznikají při expedici balíku objednávky ze skladu, popřípadě osobním odběrem zboží. Výdejky však nevznikají při expedici balíku převodky. Zvláštním typem účetního dokladu jsou pak korekční příjemky a výdejky, které řeší nesrovnalosti zjištěné při inventuře.

3.4.7 Ostatní komponenty jádra

Další důležitou komponentou je dopravce. Krom obecných informací je základem této komponenty komunikace s webovými službami jednotlivých dopravců. Jelikož se nabízené služby jednotlivých dopravců značně liší, nelze vytvořit univerzální řešení.

Pracovníci skladu jsou také jednou z komponent, přesněji uživatelé. Pro přístup k systému se musí uživatel autentizovat, přičemž uživatelské role rozdělují oprávnění jednotlivých uživatelů.

Poslední z důležitých obecnějších komponent je prodejní kanál. Jelikož firma provozuje hned několik elektronických obchodů, je třeba rozlišovat, z jakého prodejního kanálu objednávka vznikla. Prodejní kanál je důležitý i u balení zásilek, kdy balík pro různý prodejní kanál obsahuje jiné štítky a zpětné adresy.

3.5 Nefunkční požadavky

Ke specifikaci nefunkčních požadavků lze využít model FURPS[1]. Zatímco předchozí část tohoto textu pokryla funkční požadavky, nefunkční požadavky týkající se architektury, výkonu či kvality systému jsou rozebrány v této části.

Z hlediska použitelnosti lze hovořit o intuitivním uživatelském rozhraní, nicméně frontend systému není součástí této práce. I tak lze ale použitelnost specifikovat, a to právě na základě toho, jak bude frontend se systémem komunikovat. K tomu bude využito rozhraní Rest API, jež bylo představeno v předchozí části této práce.

Velký důraz je u tohoto systému kladen na spolehlivost a výkonnost, přičemž u výkonnosti jde především o odezvu systému, jelikož tento parametr je jedním z důvodu zavádění nového systému namísto aktuálně používaného.

Z oblasti omezení je pak třeba zmínit nutnost komunikace skladového systému se systémem elektronického obchodu. Zde jde především o import objednávek a produktů ze systému elektronického obchodu do skladového systému. Naopak ze skladového systému budou volány webové služby vybraných dopravců s avízy balíků.

[1] Model FURPS - Functionality, Usability, Reliability, Performance, Supportability - model specifikace, který popisuje požadavky na systém z hlediska funkčnosti, použitelnosti, spolehlivosti, výkonu a podpory.

Kapitola 4

Návrh řešení

Tato kapitola je věnována návrhu skladového informačního systému. První část je věnována návrhu databáze, včetně ER diagramu. Poté následuje návrh architektury, rozdělení systému do jednotlivých vrstev a jejich popis. V poslední části je pak blíže rozebrán návrh rozhraní aplikace, které je důležité pro komunikaci mezi backend a frontend částí systému.

4.1 Návrh schéma databáze

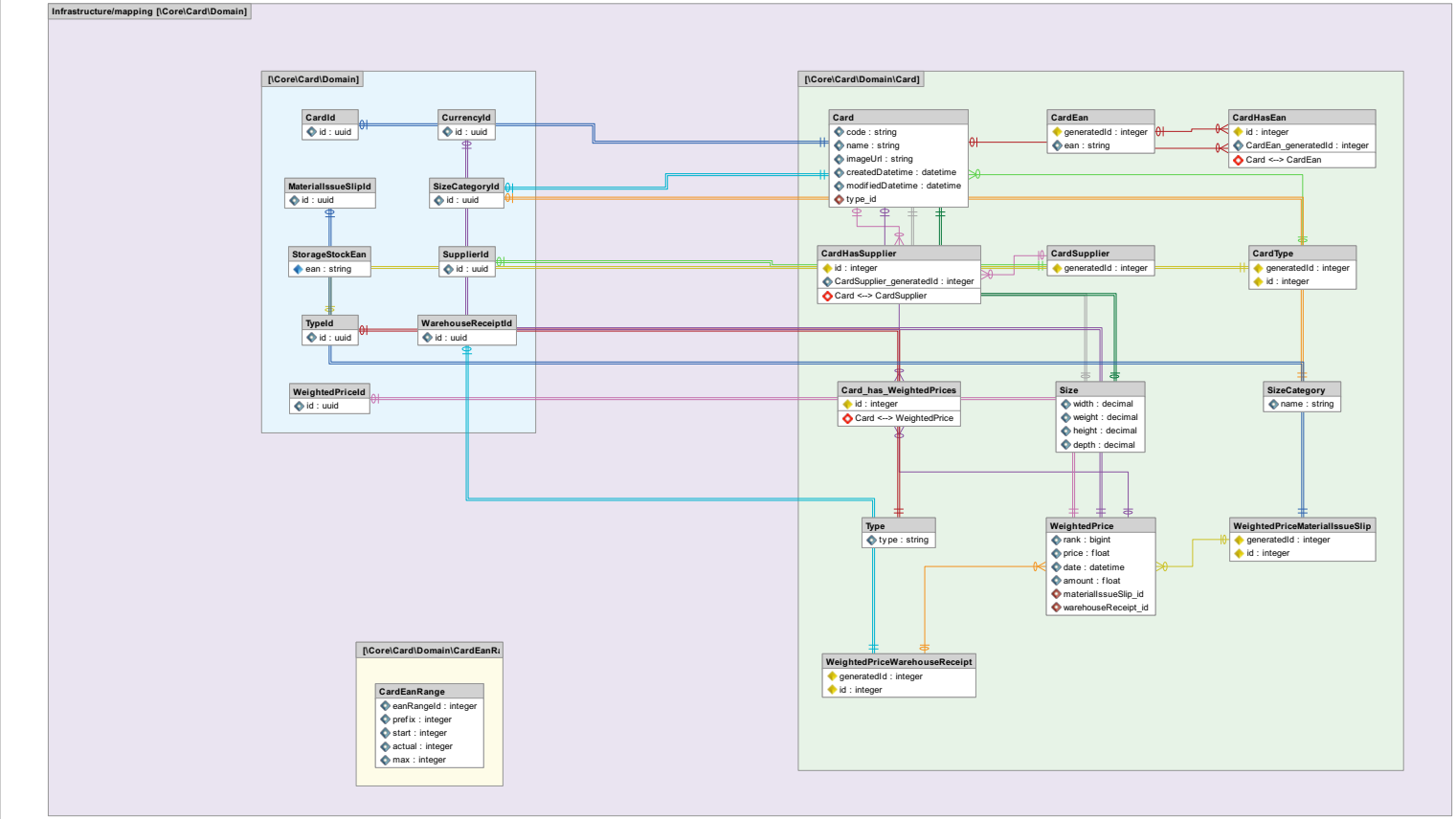
Na základě specifikace komponent a požadavků byl vytvořen návrh schématu relační databáze skladového systému. V ER diagramu je použita notace Crow Foot [8]. Pro lepší přehlednost byly jednotlivé tabulky rozděleny do společných kontextů, přičemž vazby mezi jednotlivými tabulkami existují i mezi kontexty. Vzhledem k velikosti diagramu je originál k nahlédnutí i na přiloženém CD, v této práci jsou zobrazeny diagramy pro jednotlivé kontexty. V následující části jsou blíže popsány jednotlivé kontexty, včetně diagramu daného kontextu a stručného popisu jednotlivých tabulek v návrhu:

4.1.1 Kontext skladové karty

Pro evidenci skladových karet, tedy jednotlivých produktů ve skladě, jejich kategorizaci a evidenci pohybů existuje kontext skladová karta. ER diagram lze vidět na obrázku 4.1.

Skladová karta (Card) Skladová karta reprezentuje produkt ve skladovém systému. U produktu jsou zaznamenány potřebné informace, jako jeho kódy EAN či logistické údaje (rozměry, hmotnost apod.). Zároveň se karta váže na velikostní kategorii či typ produktu. Každá skladová karta má svou váženou nákupní cenu včetně její historie.

Pohyby skladové karty (Card history record) V systému je nutné evidovat i jednotlivé pohyby skladových zásob na skladě, kvůli čemuž existuje tabulka pro zápis těchto pohybů. Eviduje se, jaké množství, jakým uživatelem a do/z jakého úložiště bylo měněno.



Obrázek 4.1: ER diagram návrhu kontextu skladové karty [vlastní]

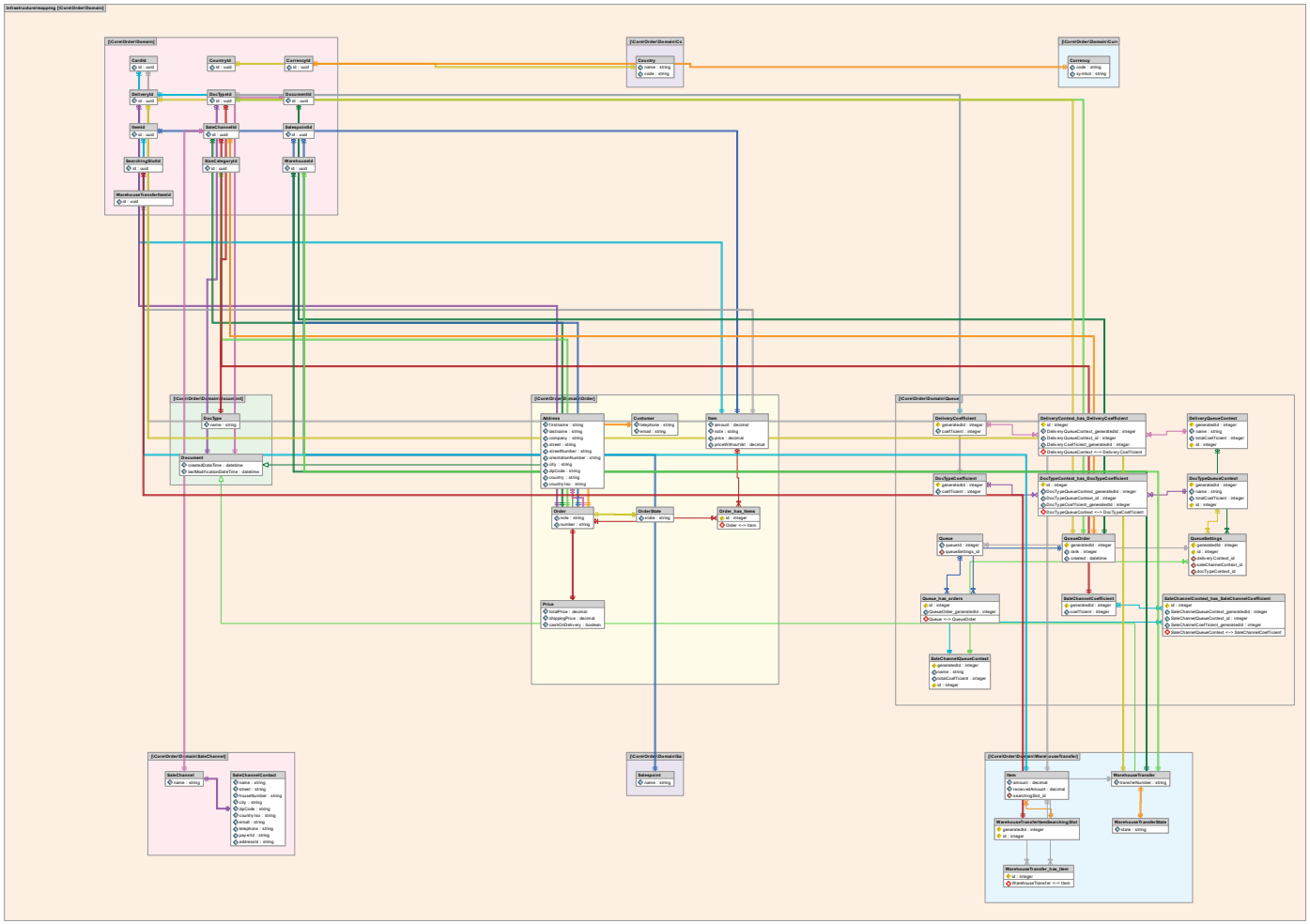
4.1.2 Objednávkový kontext

Kontext objednávek obsahuje základní tabulky jako měna, země, prodejní kanál či prodejní místo. Krom toho zde patří dva základní typy dokladů, a to objednávky a převodky. Poslední významnou entitou patřící do tohoto kontextu je fronta objednávek. Diagram lze vidět na obrázku 4.2.

Fronta (Queue) Obsahuje tabulky určující nastavení fronty, stejně jako samotnou frontu objednávek. Fronta má své nastavení, na jehož základě je vypočten koeficient objednávky. Výpočet ovlivňuje zvolený dopravce, prodejní kanál a typ objednávky. Samotná fronta objednávek je pak seřazena na základě tohoto koeficientu.

Objednávka (Order) Popisuje samotnou objednávku zároveň se zákazníkem a doručovací adresou. Objednávka obsahuje svůj stav a jednotlivé položky, přičemž položka je tvořena skladovou kartou a množstvím.

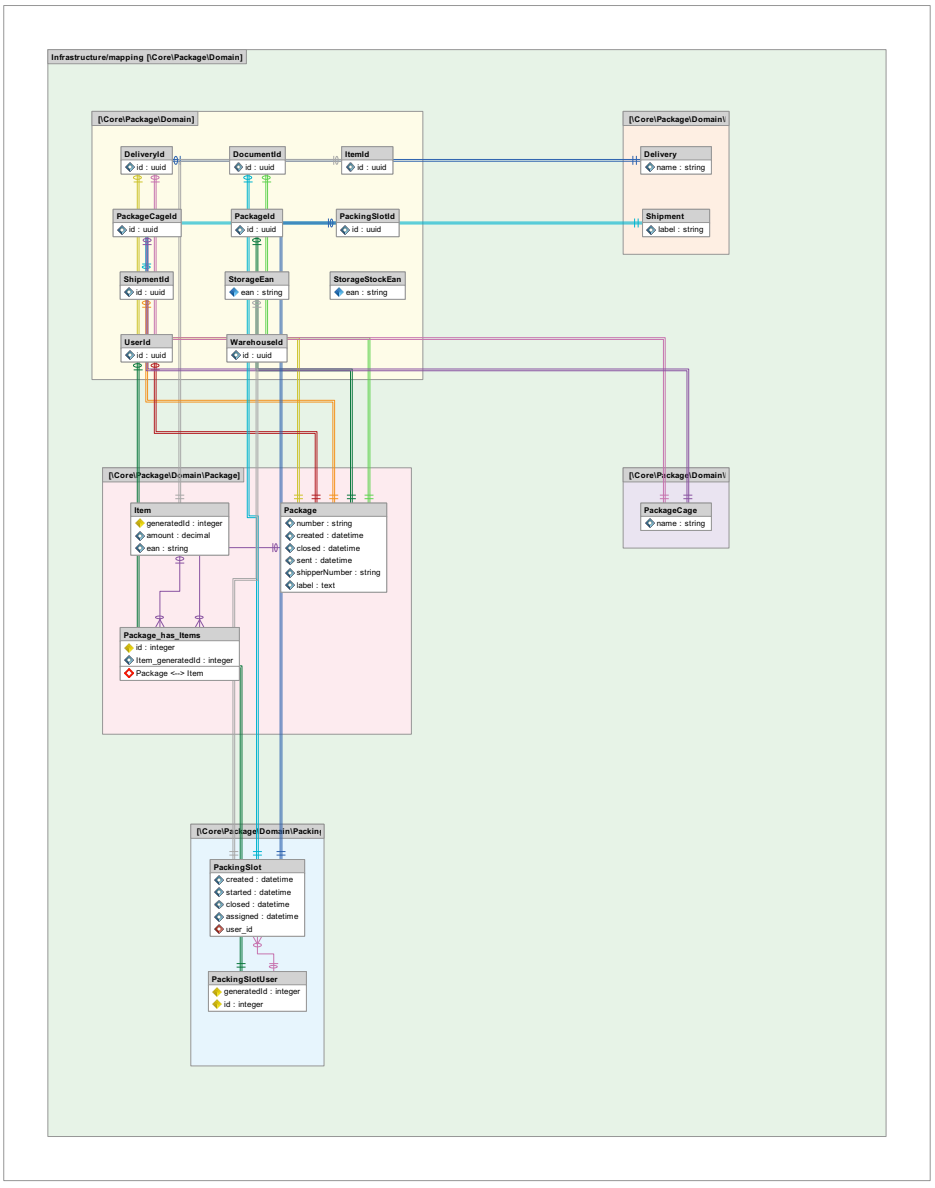
Převodka (Warehouse transfer) Tabulka popisující převodku mezi jednotlivými sklady. Obsahuje aktuální stav převodky a položky. Každá položka je tvořena skladovou kartou, množstvím a množstvím již přijatým na cílovém skladě.



Obrázek 4.2: ER diagram návrhu objednávkového kontextu [vlastní]

4.1.3 Kontext balíku

Tabulka balík eviduje jednotlivé zásilky s vazbou na objednávku včetně obsažených položek. Virtuální entita balící slot zastřešuje balení a expedici jedné objednávky, která může obsahovat více záznamů v tabulce balíků. Do tohoto kontextu patří navíc i tabulka dopravců. ER diagram lze vidět na obrázku 4.3.



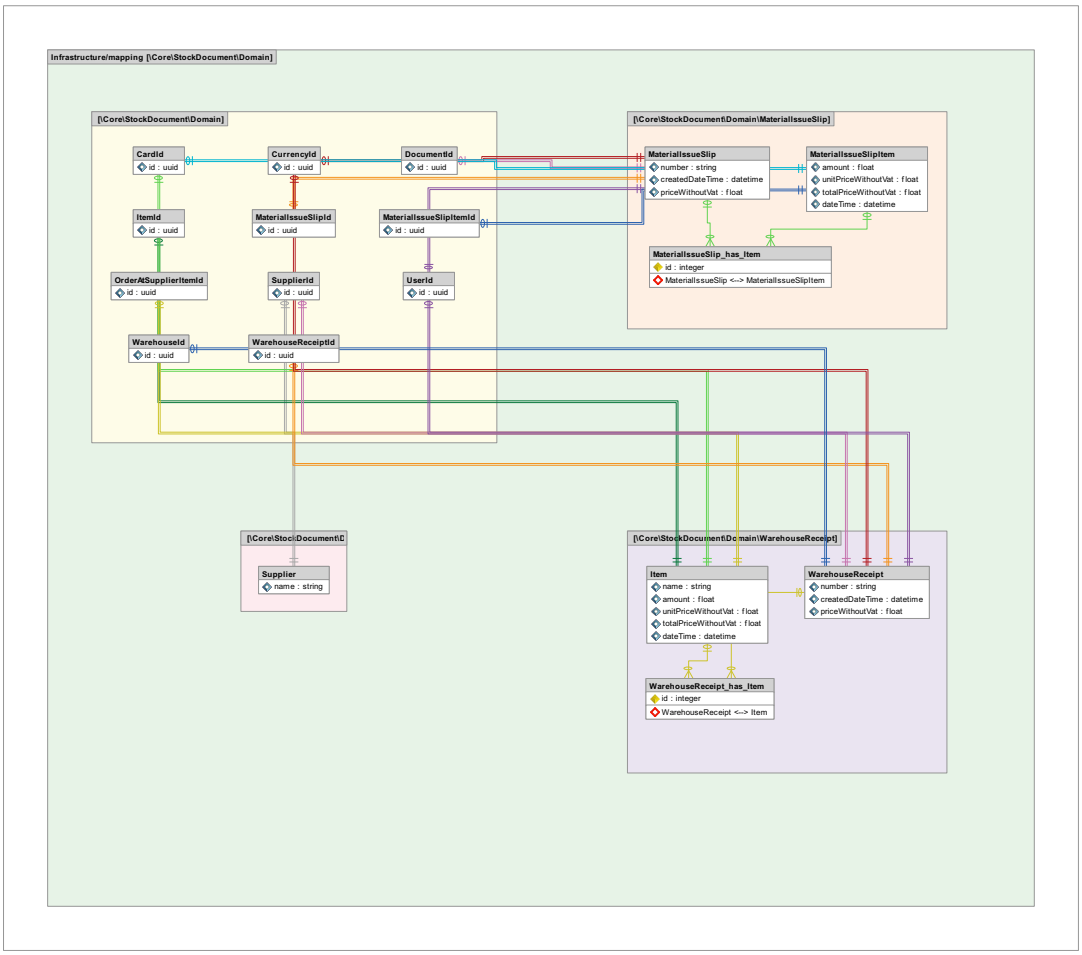
Obrázek 4.3: ER diagram návrhu kontextu balíku [vlastní]

4.1.4 Kontext účetního dokladu

V tomto kontextu se nachází tabulky pro dva základní účetní doklady skladu, a to příjemku a výdejku. Kompletní ER diagramu tohoto kontextu lze vidět na obrázku 4.4.

Příjemka (Warehouse receipt) Tabulka evidující jednotlivé příjmy zboží tak, jak jsou zpracovány na pracovišti příjmu. Každá příjemka se váže k určitému časovému okamžiku, uživateli a položkám. Zároveň se každá položka váže na dodavatele. Položky obsahují i nákupní cenu, díky které lze později spočítat váženou nákupní cenu zboží.

Výdejky (Material issue slip) Odcházející zboží ze skladu je evidováno v tabulce výdejek. Stejně jako u příjemky i zde je nutné evidovat položky zboží s cenou, za jakou byly vydány, časový okamžik vytvoření či uživatele, který zboží vydal.



Obrázek 4.4: ER diagram návrhu kontextu účetního dokladu [vlastní]

4.1.5 Skladový kontext

Tento kontext obsahuje především fyzické entity, tedy sklady, pozice a úložiště. Dále jsou součástí tabulky pro evidenci skladové zásoby včetně dalších pomocných tabulek, jako korekce či nenalezené zboží. Zároveň zde patří i tabulky nutné pro proces vyhledávání. ER diagram lze vidět na obrázku 4.5.

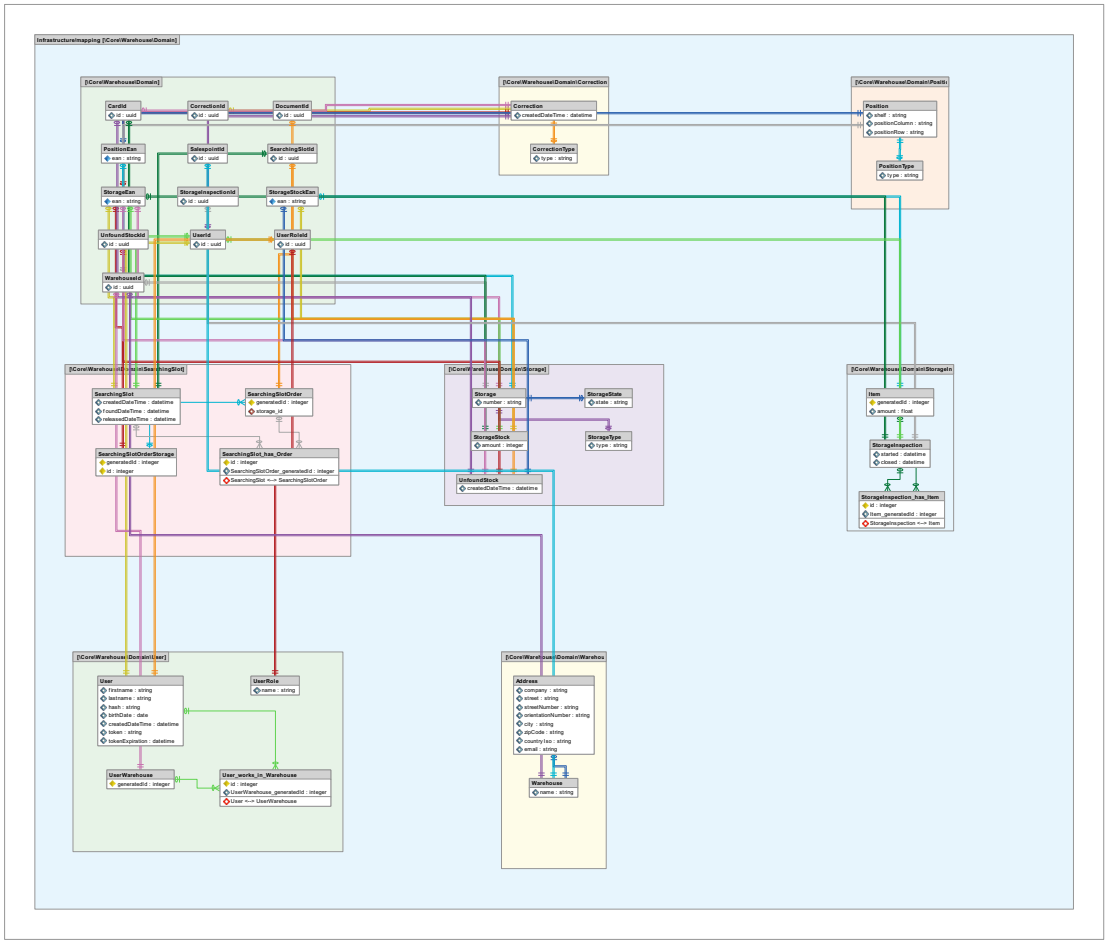
Pozice & úložiště (Position & Storage) Tabulky pro pozici a úložiště obsahují informace o typu a fyzickém umístění. Dále se k úložišti vážou tabulky pro zásobu v úložišti či nenalezené zboží.

Uživatel (User) Tabulky pro uživatele a uživatelské role, včetně uchování autentizačních údajů uživatele.

Vyhledávací slot (SearchingSlot) Jedná se o virtuální entitu, která zastřešuje proces vyhledávaní. Obsahuje informaci o vyhledávaných objednávkách, již nalezeném zboží či stavu vyhledávání.

Korekce Zápis v tabulce nenalezeného zboží je podnětem pro kontrolu úložiště. Proto existuje entita kontrola úložiště, ve které se evidují jednotlivé prováděné kontroly obsahu úložiště včetně jejich výsledku. Na základě této kontroly pak mohou vznikat plusové či minusové korekce.

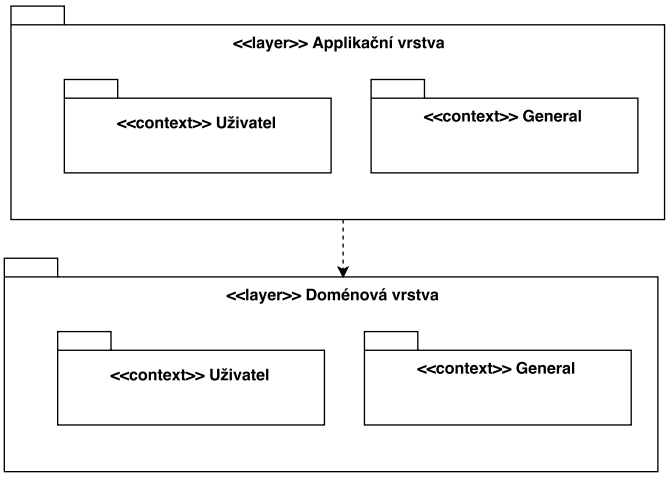
Sklad Systém se dělí na jednotlivé sklady, proto existuje tabulka skladů. Ke každému skladu je navíc nutné evidovat jeho adresu pro přeposílání zboží pomocí převodek.



Obrázek 4.5: ER diagram návrhu skladového kontextu [vlastní]

4.2 Návrh architektury

Struktura aplikace je rozdělena do dvou vrstev, a to do vrstvy doménové a vrstvy aplikační. Každá vrstva obsahuje jednotlivé kontexty známé z návrhu databáze. Aplikační vrstva zajišťuje zpracování HTTP požadavků, které zasílá frontend systému včetně odpovědí na ně. Vzhledem k využití CQRS, který je představen v úvodu této práce, jsou požadavky rozděleny na čtecí a zápisové. Čtecí požadavky zpracovává Query s využitím přímého přístupu do databáze, zatímco zápisové požadavky jsou zpracovány i s použitím doménové vrstvy. Jednoduchý model vrstev aplikace lze vidět na obrázku 4.6



Obrázek 4.6: Návrh architektury [vlastní]

4.2.1 Aplikační vrstva

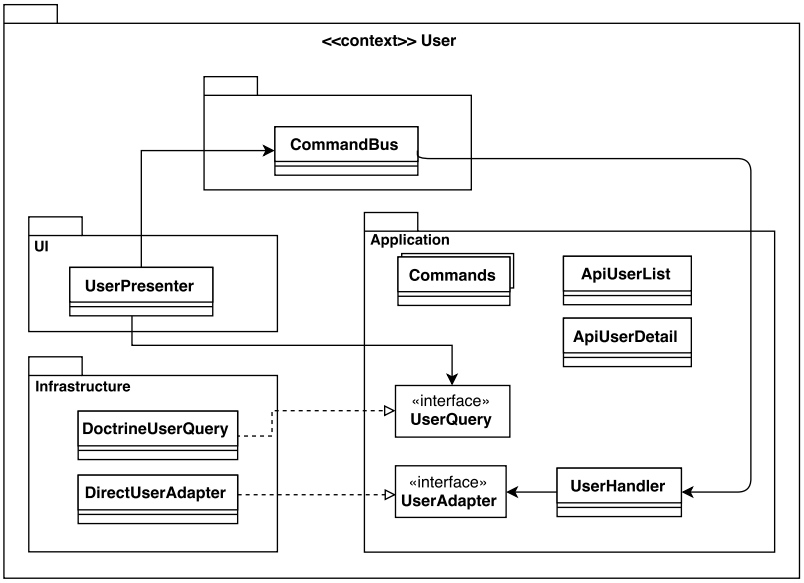
Na obrázku 4.7 lze vidět architekturu aplikační vrstvy aplikace pro kontext uživatele. Přestože se jedná o specifický kontext, lze model značně zobecnit pro popis ostatních kontextů. Základem je Presenter ve vrstvě UI, který má reference na Query a CommandBus.

Presenter zpracovává a vyhodnocuje příchozí požadavky, přičemž jde-li o čtecí požadavek, využije třídu Query k získání požadovaných dat, jak lze vidět na sekvenčním diagramu na obrázku 4.8, přičemž Query si data pomocí repositáře získá přímo z databáze. Data poté vrátí v podobě objektu třídy typu Api, v našem případě například ApiUserDetail, pro data s detaily uživatele. Všechny třídy tohoto typu zároveň určují formát výstupních dat ve formátu JSON. Presenter tedy nakonec získá právě instanci třídy typu Api, jejíž naformátovaný výstup vrátí spolu se zvoleným návratovým kódem.

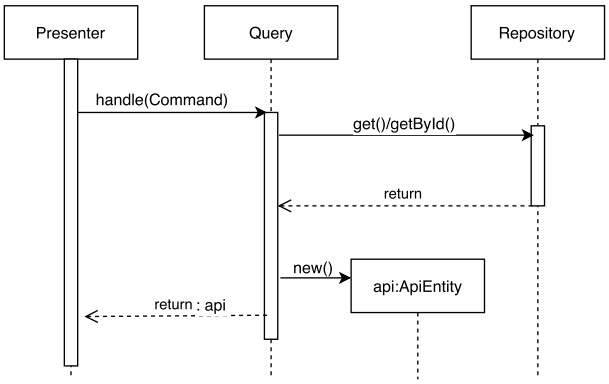
V případě zápisového požadavku je vytvořen specifický objekt třídy typu Command, který je odeslán na sběrnici CommandBus. Command je následně zpracován příslušným objektem třídy typu Handler, který předá řízení adaptéru, ze kterého se již volá mikroslužba daného kontextu v doménové vrstvě. Přesný postup jednotlivých volání lze vidět na sekvenčním diagramu 4.9.

4.2.2 Doménová vrstva

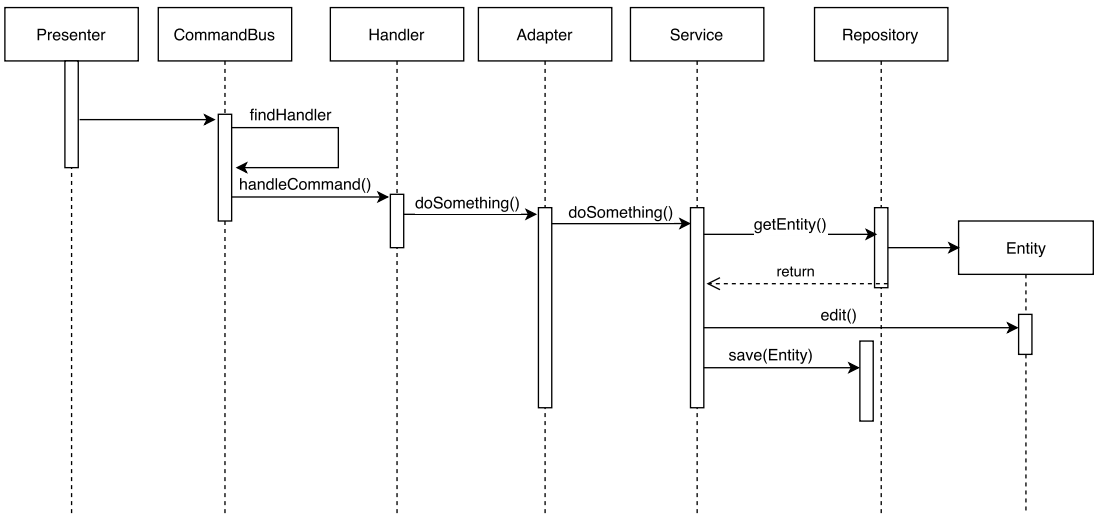
Základem doménové vrstvy jsou jednotlivé kontexty, přičemž každý kontext obsahuje svou mikro službu. Na diagramu 4.10 lze vidět kontext uživatele. Vstupním bodem je třída User-



Obrázek 4.7: Návrh architektury aplikační vrstvy pro kontext uživatele [vlastní]

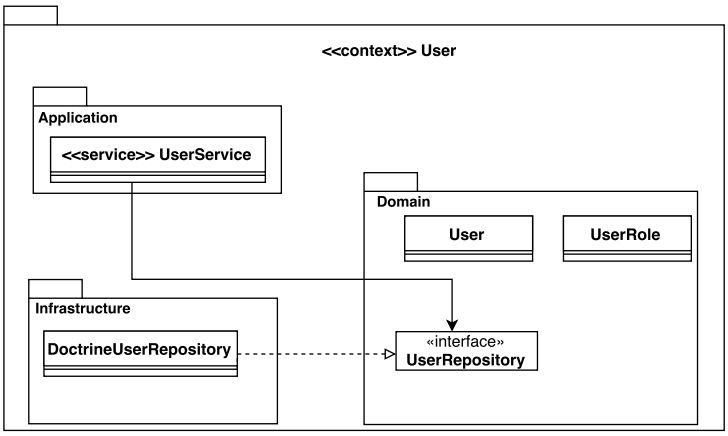


Obrázek 4.8: Sekvenční diagram zpracování dotazu na systém [vlastní]



Obrázek 4.9: Sekvenční diagram zpracování požadavku na systém [vlastní]

Service, která obsahuje referenci na UserRepository, jež umožňuje do databáze uložit či aktualizovat entitu, nebo ji naopak z databáze přečíst či smazat. Díky tomu lze získat specifický zápis z tabulky Uživatel ve formě objektu User, upravit jej a opětovně jej uložit či smazat. Přestože je zde znázorněn jediný kontext doménové vrstvy, ostatní kontexty jsou navrženy totožným způsobem.



Obrázek 4.10: Návrh architektury doménové vrstvy pro kontext uživatele [vlastní]

4.3 Návrh rozhraní aplikace

V tabulce 4.3 lze vidět navrhované metody rozhraní pro jednotlivé entity systému. Tyto metody bude využívat frontend pro komunikaci s backend aplikací skladového systému. Přístupné budou tyto metody pouze přihlášeným uživatelům. Pro rozhraní byla vytvořena kompletní dokumentace v nástroji Apiary v popisovacím jazyce API blueprint, který byl představen v jedné z předchozích kapitol.

V této práci jsou uvedeny pouze názvy jednotlivých metod spolu s detailním popisem vybrané metody pro přiblížení popisu. Kompletní dokumentace rozhraní je k dispozici v textovém dokumentu na přiloženém CD.

Ukázku definice metody rozhraní lze vidět v tabulce 4.2, ve které je podrobně popsána metoda Create pro entitu Uživatel. Ta využívá HTTP metodu POST. Adresa uvedená je pouze ve formě zdroje na uvedeném serveru, přičemž server se specifikuje pro celou dokumentaci. Dále je popsán dotaz, kde je specifikován formát těla a samotné tělo. Nakonec je ukázána očekávaná odpověď na dotaz, obsahující tělo, návratový kód a hlavičku odpovědi.

Attributes	Name	Create
	Method	POST
	Address	/user/
Request	Headers	Content-Type:application/json

	Body	{
		"firstname": "Jarda",
		"lastname": "Jagr",
		"birthDate": "2015-09-29",
		"role": "3d005e5b-c41a-45a6-82dc-d32229576b83"
		}

Response	Code	201
	Headers	Location: /user/3d005e5b-c41a-45a6-82dc-d32229576b83

	Body	{
		"uuid": "3d005e5b-c41a-45a6-82dc-d32229576b83"
		}

Tabulka 4.1: Popis metody Create entity Uživatel

Attributes	Name	List
	Method	GET
	Address	/sizeCategory
Response	Headers	Content-Type:application/json
	Code	200

	Body	{
		"sizeCategories": [
		{
		"uuid": "3d005e5b-c41a-45a6-82dc-d32229576b83",
		"name": "Velké"
		}
		]
		}

Tabulka 4.2: Popis metody List entity Velikostní kategorie

SizeCategory	Create	**Supplier**	List
	List	**Country**	List
	Delete	**Salespoint**	List
Card	List	**Storage**	List
	Search		Create
	Change size category		Delete
	Change product size		Change type
	Change package size		Change position
	List of card		Search cards in storages
	List of card's ean		Search eans in storages
	Detail		Add EAN
	Change eans		Change EAN amount
	List of card's moves		Remove EAN
	Weighted price		Start inspection
	List card's stock changes		Add EAN into inspection
	List weighted price history		Finish inspection
Position	Create	**Package**	List
	List		Detail
	Delete		Add goods
	Change type		Close package
SaleChannel	List		Send package
Delivery	List		Create
PackingSlot	Assigne user	**UnfoundStock**	Create
	Is all order packages packed		List
	Close packing slot		Delete
	Detail		Delete
SearchingSlot	Create	**UserRole**	Create
	Detail		List
	Add EAN		Delete
	Card not found		Change name
	Orders		
	Assign order to storage		
	Set order storage full		

Tabulka 4.3: Tabulka metod rozhraní pro komunikací ze strany frontend aplikace

User	Create	**Queue**	Detail
	Delete		Change delivery coefficient
	List		Change docType coefficient
	Change firstname		Change salChan. coefficient
	Change lastname		Change delivery priorities
	Change birthDate		Change salChan. priorities
	Change role		Change docType priorities
	Log in		List of orders in queue
	Log out		
	Change warehouses		
Warehouse	List	**Material issue slip**	Detail
	Search cards in warehouses		List
Warehouse receipt	List	**Warehouse transfer**	Detail
	Create		List
	Detail		Create
	Change item amount		Delete
	Assign item to order at supplier		Change state
Currency	List		Change state
Order	List		Change sender warehouse
	Detail		Change recipient warehouse
Correction	List		Change item amount
	Add missing goods		Add item
	Add redundant goods		Delete item

Tabulka 4.4: Pokračování tabulky 4.3

Kapitola 5

Implementace

Tato kapitola je věnována implementaci skladového informačního systému. Pro implementaci byly použity nástroje a technologie, jež jsou popsány v teoretickém úvodu této práce, informační systém byl vytvořen dle návrhu z předchozí kapitoly. V úvodu kapitoly je popsáno rozdělení aplikace do dvou základních vrstev, které jsou dále dělitelné. U obou vrstev je blíže popsán vybraný kontext aplikace pro přiblížení funkčnosti komunikace mezi aplikační a doménovou vrstvou a komunikaci uvnitř každé z vrstev. V další části je popsána implementace nosných procesů skladového systému, načež je část práce věnována popisu zpracování některých vybraných komplexních problémů společně s implementací komunikace informačního systému s webovými službami dopravců. Na závěr je krátce popsáno vytvoření jednotkových testů.

5.1 Vrstvy aplikace

Aplikace skladového informačního systému je rozdělena do dvou základních vrstev, které jsou dále členěny na menší části. Následující řádky jsou věnovány obecnému popisu implementace obou vrstev, včetně základních příkladů použití.

5.1.1 Vrstva aplikačního rozhraní

Pro implementaci této vrstvy byly využity aplikační rámce Slim a Nette. Zatímco Nette je v této vrstvě aplikace využito pouze pro vytvoření aplikačního kontejneru, framework Slim je využit pro směrování, zpracování HTTP dotazu a zajištění odpovědi či operace v mezivrstvě.

Na základě konfiguračních souborů ve formátu NEON je nejprve sestaven aplikační kontejner, což se děje v souboru *bootstrap.php*. Ten se využije v kořenovém souboru *index.php*. Kontejner je použit při vytváření instance aplikace aplikačního rámce Slim. Následně se volají dvě statické metody, které modifikují právě vytvořenou instancí aplikace. *addMiddlewares* třídy *MiddlewareFactory* pro vytvoření jednotlivých mezivrstev, a *createRoutes* třídy *RouteFactory* pro zajištění překladu URL adres na akce jednotlivých tříd typu *Presenter*. Následně se již spustí instance aplikace.

Pro účely skladového systému jsou implementovány dvě mezivrstvy (middleware). Tou první je mezivrstva pro zachycení a zpracování výjimek, které se mohou vyskytnou v rámci zpracování dotazu. V případě, že k této situaci dojde, je odchycena výjimka a proveden rollback již zpracovaných databázových operací. Následně se na základě informací z výjimky vytvoří odpověď systému ve formátu JSON obsahují kód chyby a zprávu o chybě. Zároveň

jsou do logovacího souboru zapsány podrobnosti o chybě. Příklad chybové hlášky ve formátu JSON lze vidět na ukázce 5.1.

```
{
   "code": 404,
   "status": "error",
   "errors": [
      {
         "field": "cardId",
         "code": 1013,
         "message": "Card was not found"
      }
   ]
}
```

Obrázek 5.1: Příklad chybové hlášky informačního systému [vlastní]

Důležitou součástí této vrstvy jsou třídy typu *Presenter*. Ty zpracovávají požadavky klienta, které na ně přesměruje Router, a vytváří požadovanou odpověď. Právě překlad adres na metody je hlavní činnost, o kterou se stará druhá mezivrstva, tzv. *Router*. Příklad vytvoření překladu URL adresy na akci třídy typu *Presenter* lze vidět na ukázce 5.2.

```
$app->get('/card/{uuid}', 'CardPresenter:getCardDetail');
```

Obrázek 5.2: Příklad zapsání překladu adresy na akci presenteru [vlastní]

V tomto příkladu lze vidět relativní adresu zdroje jako první parametr funkce *get*. Druhým parametrem je název třídy typu *Presenter* a dvojtečkou oddělený název příslušné metody dané třídy. Každý *Presenter* v implementované aplikaci je děděný z třídy *BasePresenter*, která obsahuje funkce pro zpracování vstupních parametrů. Obecně pak specifický *Presenter* obsahuje příslušnou *Query* a *CommandBus*.

Query slouží pro získání dat z databáze bez využití logiky doménové vrstvy. Pro tyto účely každá *Query* využívá *EntityManager*, který je součástí Doctrine, pomocí něhož získá data z databáze. Ty pak předá objektu typu *Api*, což je jednoduchý data transfer object (DTO). Navíc ovšem obsahuje metodu *toApiArray()*, která vrací naformátované pole obsahující získaná data. Právě toto pole je základem odpovědi ve formátu JSON.

Druhou důležitou součástí třídy typu *Presenter* je *CommandBus*, tedy jakási sběrnice příkazů. Jedná se o součást knihovny LiteCQRS. Poté stačí zavolat metodu *handle()* instance třídy *CommandBus*, která příslužný objekt typu *Command* zpracuje. Zpracování probíhá na základě informací z konfiguračního souboru, kdy je každé existující třídě typu *Command* přiřazen specifický handler, tedy objekt, který ví, jak daný *Command* zpracovat. Vybraný handler má poté referenci na adaptér pro vybranou entitu. Na základě zpracovávaného objektu typu *Command* tak vyber specifickou funkci adaptéru. Adaptér je posledním stupněm vrstvy aplikačního rozhraní, kdy je předáno řízení mikroslužbě příslušné entity v doménové vrstvě.

Příklad implementace ve vrstvě aplikačního rozhraní

Pro příklad implementace byla vybrána entita skladové karty. První část kódu na ukázce 5.3 je z třídy typu *Presenter*, přičemž se jedná o metodu pro získání detailu skladové karty.

Funkce má tři parametry. Prvním je samotný dotaz, který může obsahovat tělo ve formátu JSON či GET parametry. Response je objekt odpovědi, do které se vkládá tělo ve formě pole, přičemž zobrazeno bude ve formátu JSON, či návratový kód HTTP. Posledním parametrem je pole argumentů, které obsahuje parametry z příslušné adresy definované v objektu *Router* při vytváření pravidel pro překlad adres. K získání dat *Presenter* využívá již zmíněnou *Query*, která vrací data transfer object typu *Api*, v našem případě detail skladové karty na základě jejího identifikátoru.

```
public function getCardDetail(
   Request $request,
   Response $response,
   array $args
) {
   $cardId = $this->getArrayUuidProperty(
      $args,
      'uuid'
   );
   $card = $this->query->getCardById($cardId);
   return $response->withJSON(
      [
         'card' => $card->toApiArray()
      ])->withStatus(200);
}
```

Obrázek 5.3: Příklad funkce třídy *Presenter* pro získání dat s využitím *Query* [vlastní]

Druhá ukázka kódu 5.4 zobrazuje funkci třídy *CardPresenter* pro editaci velikostní kategorie skladové karty. Hlavička funkce je totožná, jako u předchozí metody. V první části jsou nejprve zpracovány parametry, na jejichž základě je vytvořený objekt typu *Command*, konkrétně *ChangeSizeCategoryCommand*. Pro něj existuje v konfiguraci přidělený handler, který jej rozpozná a zavolá vybranou funkci adaptéru. Ten poté už volá funkci mikroslužby *CardService*, která spadá do doménové vrstvy.

5.1.2 Doménová vrstva

Implementace doménové vrstvy proběhla s využitím Nette a Doctrine. Prvně zmíněný byl využit pro vytvoření kontejneru, zatímco framework Doctrine k práci s databází. Stejně jako u vrstvy aplikačního rozhraní, je doménová vrstva rozdělena do jednotlivých kontextů. Kontext je pak dělen na další tři podvrstvy, a to aplikační, doménovou a infrastrukturu, přičemž každý kontext může obsahovat několik entit.

Aplikační vrstva obsahuje mikroslužbu pro práci s vybranou entitou. Tato mikroslužba je vstupním bodem pro každou entitu v doménové vrstvě, a právě zde se odehrává důležitá logika systému. Krom toho se v aplikační vrstvě můžou vyskytnout objekty pro přenos dat (Data transfer object). Mikroslužba v sobě obsahuje referenci na repositář pro získávání dat, zároveň může obsahovat reference na ostatní mikroslužby ze stejného kontextu. Pokud je nutné, aby komunikovala s jinou mikroslužbou z odlišného kontextu, je to zajištěno přes adaptér.

Infrastruktura má na starosti práci, kterou není třeba vystavovat přímo do aplikační mikroslužby. Především se tak jedná o získávání dat z databáze, což má na starosti repositář

```
public function changeSizeCategory(
   Request $request,
   Response $response,
   array $args
) {
   $cardId = $this->getArrayUuidProperty(
      $args,
      'uuid'
   );
   $data = $request->getParsedBody();
   $sizeCategory = $this->getArrayUuidProperty(
      $data,
      'sizeCategory'
   );
   $command = new ChangeSizeCategoryCommand(
      $cardId,
      $sizeCategory
   );
   $this->commandBus->handle($command);
   return $response->withStatus(204);
}
```

Obrázek 5.4: Příklad funkce pro editaci dat s využitím třídy *CommandBus* [vlastní]

dané entity. Krom toho se zde mohou objevit složitější logické operace, které jsou pro větší přehlednost vyčleněny z mikroslužby. Pro příklad lze uvést generování převodek pro objednávky s osobním odběrem či komunikace s dopravci skrze webové služby. Obě tyto operace jsou popsány v další části této práce.

```
public function changeSizeCategory(
   UuidInterface $cardUuid,
   UuidInterface $sizeCategoryUuid
) {
   $card = $this->repository->get(new CardId($cardUuid));
   $sizeCategoryId = new SizeCategoryId($sizeCategoryUuid);
   if (!$this->sizeCategoryRepository->has($sizeCategoryId)) {
      throw new SizeCategoryNotFoundException();
   }
   $card->changeSizeCategory($sizeCategoryId);
   $this->repository->save($card);
}
```

Obrázek 5.5: Příklad funkce pro editaci dat s využitím repositáře [vlastní]

Poslední podvrstvou je doménová vrstva. Ta obsahuje doménové objekty tak, jak jsou namapovány do databáze. K mapování objektů na databázi je využíván framework Doctrine. Pro tuto implementaci bylo využito mapovacích souborů ve formátu *YAML*. Ke každému doménovému objektu tak existuje příslušný mapovací soubor v tomto formátu.

Příklad implementace v doménové vrstvě

Příklad implementace navazuje na předchozí příklad z vrstvy aplikačního rozhraní, který skončil na volání funkce mikroslužby skladové karty pro změnu velikostní kategorie.

```
public function get(CardId $cardId): Card {
   try {
      $qb = $this->entityManager->createQueryBuilder();
      $qb->select('c, e, t, sc, p');
      $qb->from(Card::class, 'c');
      $qb->leftJoin('c.eans', 'e');
      $qb->leftJoin('c.weightedPrices', 'p');
      $qb->leftJoin('c.type', 't');
      $qb->leftJoin('c.sizeCategory', 'sc');
      $qb->where('c.cardId.id = :id');
      $query = $qb->getQuery();
      $query->setParameter(':id', $cardId->getId());
      return $query->getSingleResult();
   } catch (NoResultException $exception) {
      throw new CardNotFoundException;
   }
}
```

Obrázek 5.6: Funkce repositáře skladové karty pro získání zápisu entity [vlastní]

V příkladu 5.5 lze vidět využití repositáře, pomocí něhož se nejprve z databáze načte požadovaný záznam entity dle zvoleného identifikátoru. Následně je pomocí repositáře entity velikostní kategorie ověřeno, že příslušný zápis v tabulce velikostních kategorií opravdu existuje. Poté už je možno skladové kartě přiřadit novou velikostní kategorii, načež je pomocí repositáře tento zápis v tabulce skladových karet aktualizován.

Repositář entity skladové karty obsahuje funkce pro přidání, aktualizaci, smazání, načtení a ověření existence zápisu. Krom funkce pro získání zápisu entity lze pro ostatní využít přímo třídu *EntityManager*, a její příslušné metody jako *remove()* či *persist()*. Složitější to je u metody pro získání zápisu.

Jak lze vidět z příkladu 5.6, pro načtení jednoho zápisu entity skladové karty, zde je nutné vytvářet komplexnější SQL dotaz pomocí objektu *QueryBuilder*. To je zapříčiněno existujícími vazbami mezi entitami. Pokud pro entitu neexistují žádné vazby na ostatní entity, lze využít funkci *find()* objektu třídy *EntityManager*.

5.2 Implementace hlavních procesů

Implementovaný skladový systém odbavuje procesy, které byly identifikovány při specifikaci požadavků. Tři nejdůležitější procesy jsou příjem, vyhledání a balení zboží. Následující část této práce je věnována stručnému popisu implementace těchto nosných procesů i z pohledu metod aplikačního rozhraní.

Jednotlivé metody aplikačního rozhraní nevolá přímo uživatel, ale jsou volány skrze frontend aplikace. Jelikož ovšem frontend a jeho implementace není součástí této práce, je pro zjednodušení použito spojení *"volání metody uživatelem"*.

5.2.1 Příjem

Nejjednodušší je proces příjmu zboží, pro který existuje jediná metoda aplikačního rozhraní. Jedná se o metodu *AddEan* entity *Storage*. Přestože je to obecná metoda pro přidání zboží s daným kódem EAN do vybraného úložiště, lze v těle dotazu specifikovat typ vložení, přičemž se může jednat o klasické vložení zboží, příjem zboží či příjem převodky.

Ve všech případech je v databázi zapsána nová skladová zásoba v označeném úložišti, přičemž proces příjmu navíc obsahuje vytvoření příjemky, tedy účetního dokladu. Vytvořený účetní doklad obsahuje vazbu na uživatele, který zapříčinil jeho vznik, vazbu na skladovou kartu a sklad, ve kterém příjem proběhl.

Důležitou součástí příjemky je ovšem i cena. Tu ale ve chvíli příjmu uživatel nezná, proto se vytváří příjemka bez uvedení ceny, nicméně s vazbou na dodavatele. Cena na příjemce je pak upravena pomocí komunikace s informačním systémem elektronického obchodu, ve kterém se evidují objednávky u dodavatele. Po příjmu všech položek objednávky jsou položky příjemky spárovány s položkami objednávky u dodavatele z informačního systému a jsou jim přiřazeny příslušné ceny. Komunikace probíhá skrze aplikační rozhraní pro import dat do skladového systému, které ovšem není součástí této práce.

Při příjmu převodky se eviduje již přijaté množství dané převodky. V případě, kdy jsou všechny položky převodky přijaty, je převodka označena stavem dokončeno. Zároveň, pokud je třeba přijaté zboží ihned přeposlat na další sklad, je vygenerována další převodka. Tento proces je blíže popsán v další části této práce.

Po naplnění úložiště zbožím probíhá jeho zapozicování, tedy přesunutí úložiště do některé z volných pozic na skladě. To se děje pomocí metody *ChangePosition* entity *Storage*. Zde ovšem existuje omezení, kdy je možné zboží vložit jen do některých pozic. Systém se tak snaží zabránit uložení zboží velké velikostní kategorie do pozic umístěných v prvním patře skladu.

5.2.2 Vyhledání

Implementace procesu vyhledávání je podstatně složitější. Z pohledu aplikačního rozhraní musí uživatel nejprve vytvořit virtuální entitu vyhledávacího slotu pomocí metody *Create*, a to se zvoleným vyhledávacím úložištěm. Tato operace způsobí výběr vybraných objednávek z fronty objednávek určených pro vyhledání, přičemž bližší popis tohoto algoritmu je popsán v další části této práce.

Poté si uživatel může pomocí metody *Detail* zobrazit data o aktuálním vyhledávacím slotu. Ty obsahují obecné informace jako počet objednávek či datum vytvoření, ale především pak zboží určené k nalezení včetně úložišť a příslušných pozic, ve kterých je zboží dostupné.

Další operace pak uživatel využívá během samotného vyhledávání. Jedná se o metodu *AddEan* pro označení nalezeného zboží vloženého do vyhledávacího úložiště, popřípadě metoda *CardNotFound*, která označí chybějící zboží v nabízeném úložišti.

Je-li označené některé zboží během vyhledávání jako nenalezené, vznikne zápis do tabulky nenalezeného zboží, které funguje jako podnět pro kontrolu skladové zásoby v příslušném úložišti. Zároveň, není-li zboží dostupné v žádném jiném úložišti, je každá objednávka z vyhledávacího slotu obsahující toto zboží zastavena, přičemž její položky nejsou dále vyhledávány. Navíc se to projeví změnou stavu objednávky.

Poté, co uživatel nalezne veškeré hledané zboží, může si pomocí metody *Orders* zobrazit přiřazení jednotlivých položek zboží k objednávkám. Následně volá metodu *AssignOrderToStorage*, která přiřadí dané objednávce balící úložiště. Pokud je ovšem objednávka

během vyhledávání zastavena, je ji přiřazeno úložiště pro nenalezené objednávky. V posledním kroku pak uživatel potvrdí přesunutí zboží z dané objednávky do přiřazeného balícího úložiště.

Naprosto stejným procesem probíhá i vyhledávání převodky. Změny nastávají až ve fázi přiřazování balícího úložiště, kdy jsou pro převodku určeny speciální balící úložiště.

5.2.3 Balení

Balení zastřešuje virtuální entita Balícího slotu. Nad tou je nejprve zavolána metoda *AssignSlotToUser*, která přiřadí uživateli objednávku umístěnou v jednom z balících úložišť. V případě, že není žádná objednávka přichystána k balení, vrací systém chybovou hlášku. Po přiřazení objednávky si může uživatel zobrazit *Detail* balícího slotu, obsahující informace o balené objednávce. Především jsou v tuto chvíli důležité informace o vybraném dopravci a položkách objednávky, včetně jejich množství.

S prvním označením zboží uživatel volá metodu *AddGoods* nad entitou Balíku. Již při vytvoření balícího slotu je totiž zároveň vytvořen první balík pro objednávku. Dále může tedy uživatel vkládat další zboží do balíku, přičemž může celou objednávku zabalit do jediného balíku. V případě, že se mu to nepodaří, musí nejprve pomocí metody *ClosePackage* stávající balík uzavřít. Spolu s touto metodou jsou volány webové služby dopravců, od kterých je získán štítek pro polepení balíku, což je podrobněji popsáno v 5.4. Poté může uživatel pomocí metody *Create* vytvořit balík nový a opět do něj vkládat zboží.

Po zabalení celé objednávky musí ještě uživatel uzavřít celý vyhledávací slot metodou *ClosePackingSlot*, která kontroluje, zdali bylo vše správně zabaleno. Poslední operaci nad objednávkou pak provádí další pracovník balení, který sbírá z balících pultů zabalené balíky a umísťuje je do klecí pro jednotlivé dopravce. Nad každým takto vkládaným balíkem je zavolána metoda *SendPackage*. Ve chvíli, kdy jsou všechny balíky objednávky odeslány, je změněn i její stav.

U převodek je proces balení implementován totožně. Jediný rozdíl je ve zvoleném dopravci, protože převodky se posílají do cílového skladu pomocí vlastní dopravy, čímž odpadá nutnost tisku štítku na balík.

5.3 Implementace vybraných problémů

Krom hlavních procesů jsou ve vytvářeném skladovém informačním systému některé složité problémy, jejichž implementace je popsána právě v této části práce. Jedná se především o práci s frontou objednávek pro odbavení ve skladu či generování převodek pro objednávky s osobním odběrem. Dalším složitějším problémem je implementace vážené nákupní ceny. Do této části byl zařazen i popis historie pohybů skladové karty, který je řešen oddělenou databází.

5.3.1 Fronta objednávek

Algoritmus pro přiřazení objednávek z fronty pro odbavení je závislý na vybraném vyhledávacím úložišti, které si pracovník vyhledávání vybere. Než se ovšem dostaneme k tomuto algoritmu, je třeba vysvětlit, jak se objednávky ve frontě pro odbavení řadí.

Objednávky jsou ve frontě seřazeny dle svého ohodnocení, přičemž v případě stejného ohodnocení více objednávek má přednost ta, která je v systému delší dobu. Ohodnocení se spočítá dle vzorce 5.1.

Velikostní kategorie vyhl. úložiště	**Malý košík (VM)**	**Střední košík (VS)**	**Velký košík (VV)**				**(VT) Thule**
			varianta I	**varianta II**	**varianta III**	**varianta IV**	
Vel. kategorie skladové karty	malá	min. 1 střední + malá	malá + střední	malá + střední	min. 1 velká + malá + střední	všechny	min. 1 Thule + všechny
Limitní počet kusů	10	5	15	1 objednávka o 16+ kusech	1 objednávka	20 ks převodky	1 objednávka
Poznámka					Prioritní pro variantu velkého košíku		

Tabulka 5.1: Tabulka rozdělení položek vyhledávání do vyhledávacího slotu

$$RANK = C_d * P_d + C_t * P_t + C_p * P_p \tag{5.1}$$

Proměnná C znamená vždy koeficient daného kontextu. Pod indexy se skrývají jednotlivé kontexty, a to kontext dopravce (d), typu dokumentu pro odbavení (t) a prodejního kanálu (p). Proměnná P pak znamená prioritu vybrané vlastnosti. Pro kontext typu dokumentu je to převodka či objednávka, pro případ dopravců pak například DPD či GEIS. Díky těmto koeficientům a prioritám má vedoucí skladu možnost upřednostňovat odbavování vybraných objednávek. Při každé změně jakékoliv hodnoty použité při výpočtu je přepočteno ohodnocení pro všechny objednávky a sestavena nová fronta.

V okamžiku, kdy pracovník vyhledávání vybere jedno z vyhledávacích úložišť, je pro něj vytvořen vyhledávací slot s vazbou na objednávky. Objednávky jsou z fronty vybírány z vrcholu fronty dle tabulky 5.1. V praxi to znamená, vybere-li pracovník malé vyhledávací úložiště (VM), jsou mu přiřazeny objednávky do maximálního počtu desíti položek, přičemž všechny položky musí být malé velikostní kategorie. V praxi tak může nastat například to, že se nevybere objednávka z vrcholu fronty, ale první vyhovující, která obsahuje jen položky skladové karty malé velikostní kategorie. Další objednávky jsou do slotu přiřazeny podle zbývajících volných míst pro položky. Má-li tak první vybraná objednávka například 6 položek, jsou vybrány další objednávky se stejnými parametry až do maximálního počtu desíti kusů položek k vyhledání. Vždy se ovšem do vyhledávacího slotu musí zařadit kompletní objednávka. Obdobně tomu je i u středního vyhledávacího úložiště (VS), kdy je podmínkou, že první vybraná objednávka z fronty musí obsahovat minimálně jednu skladovou kartu střední velikostní kategorie.

Komplikovanější to je u velkého vyhledávacího úložiště (VV). Nejprve systém vyzkouší, zdali je ve frontě nějaká objednávka, která obsahuje skladovou kartu velké velikostní katego-

rie. Pokud ano, celá tato objednávka je přiřazena do vyhledávacího slotu a další objednávky se nevyberou. Pokud se tato varianta nepovede, tedy není-li ve frontě žádná taková objednávka, prochází se fronta znovu.

Nyní se vybere první objednávka obsahující jen skladové karty malé a střední velikostní kategorie, popřípadě první převodku ve frontě. V prvním případě se buďto do vyhledávacího slotu přiřadí jen tato vybraná objednávka, obsahuje-li minimálně 16 položek. V opačném případě se hledají další vyhovující objednávky do maximálního počtu 15 kusů. U převodek je tento proces zjednodušen. Převodky je totiž možno do vyhledávacího slotu dělit. Proto se pro případ velkého vyhledávacího úložiště do vyhledávacího slotu zařadí prvních dvacet položek převodky.

Samostatnou kategorií jsou pak vyhledávací úložiště Thule (VT). Thule je značka nosičů jízdních kol na automobily a příslušenství, přičemž se jedná o nadměrně rozměrné produkty. Proto je vyhrazeno speciální vyhledávací úložiště, pro které je vždy přiřazena jediná objednávka obsahující minimálně jeden produkt velikostní kategorie Thule.

V praxi může dojít k situaci, kdy pro danou velikost vyhledávacího úložiště nelze sestavit vyhledávací slot, tedy vybrat odpovídající objednávky z fronty. V tom případě je uživatel informován o této skutečnosti a může vyzkoušet jinou velikost vyhledávacího úložiště.

5.3.2 Vážená nákupní cena

Cena zboží na skladě se přepočítává průběžně po každém příjmu. Vážená nákupní cena (VNC) je průměrná cena všech nákupních cen u dané skladové karty. Po každém novém přírůstku skladové zásoby dané karty na sklad vypočteme na základě příjemek novou váženou nákupní cenu podle vzorce 5.2.

$$VNC = \frac{Hodnota\,zasob\,na\,sklade\ +\ Posledni\,prirustek\,hodnoty}{Mnozstvi\,zasob\,na\,sklade\ +\ Posledni\,prirustek\,mnozstvi} \tag{5.2}$$

Vzorec bere v potaz aktuální hodnotu skladové zásoby, která se vypočítá jako součin aktuální vážené nákupní ceny a počtu kusů. K aktuální hodnotě skladové zásoby se přičte součin počtu přijímaných kusů a jejich nákupní ceny, čímž získáme čitatel vzorce. Ve jmenovateli se pak pouze sečtou počty kusů přijatého zboží a celkové zásoby.

Po příjmu se získá z informačního systému elektronického obchodu nákupní cena skladové karty podle objednávky u dodavatele, přičemž právě z takto získané ceny se přepočítá vážená průměrná cena při každém přírůstku na skladě. Při výdeji zboží se využívá aktuální vážená nákupní cena, která je uvedena i na příslušné výdejce.

V rámci skladového informačního systému byl zpracován přehled vývoje vážené nákupní ceny u každé skladové karty, včetně referencí na příjemky a výdejky, které tuto cenu ovlivňovaly. Zároveň je vážená nákupní cena využívána při výpočtu celkového ohodnocení zásob na skladě.

5.3.3 Objednávky s osobním odběrem

Každý sklad v systému může mít přiřazeno prodejní místo. Každá objednávka pak může být doručována nikoliv pomocí jednoho z dopravců, ale jako osobní odběr na jednom z vybraných prodejních míst. Má-li objednávka přiřazenu vazbu na prodejní místo, nebude zařazena do fronty pro vyhledávání. Naopak bude snaha dostat všechny položky objednávky na cílový sklad, který odbavuje přiřazené prodejní místo.

To zajišťuje implementovaný algoritmus pro generování převodek na cílový sklad. Nejprve se ověří, zdali se na cílovém skladě nachází všechny položky objednávky. Pokud ano, není třeba generovat žádné převodky a může tento proces skončit.

```
public function generateTransferForOrder(
   SalespointId $salespointId,
   array $items
) {
   $salespointWarehouse = $this->warehouseAdapter->getWarehouseBySalespoint(
      $salespointId
   );
   $missingItems = $this->getItemsNotAvailableOnWarehouse(
      $salespointWarehouse,
      $items
   );
   if (count($missingItems)) {
      if ($this->isItemsAvailableOnCentralWarehouse($missingItems)) {
         $this->createTransferFromCentralToSalespointWarehouse(
            $salespointWarehouse,
            $missingItems
         );
      } else {
         $itemsInCentral = $this->getItemsAvailableOnCentralWarehouse(
            $missingItems
         );
         $this->createTransferFromCentralToSalespointWarehouse(
            $salespointWarehouse,
            $itemsInCentral
         );
         $missingItemsInCentral = $this->getItemsNotAvailableOnCentralWarehouse(
            $missingItems
         );
         $itemsOnOtherWarehouses = $this->getItemsInWarehouses(
            $missingItemsInCentral,
            $salespointWarehouse
         );
         $this->createTransferToCentralWarehouse(
            $itemsOnOtherWarehouses
         );
      }
   }
}
```

Obrázek 5.7: Funkce vygenerování převodek pro osobní odběr objednávky [vlastní]

Chybí-li ovšem na cílovém skladě některá položka, je třeba ji tam dostat pomocí převodky. Právě převodky slouží pro posílání zboží mezi jednotlivými sklady. Navíc převodky jsou přepravovány vlastní dopravou a je možno zasílat vždy jen z centrálního skladu, popřípadě na centrální sklad.

V prvním kroku, kdy na cílovém skladě chybí některé z položek, se ověří, zdali jsou všechny dostupné na centrálním skladě. Pokud ano, je třeba identifikovat převodku mezi centrálním skladem a cílovým skladem. Ta už může existovat, pak stačí pouze na tuto převodku přidat chybějící položky objednávky. Pokud převodka mezi těmito sklady ještě neexistuje, je vytvořena, a poté jsou na ni přidány chybějící položky objednávky.

Nejsou-li některé položky objednávky dostupné na cílovém ani centrálním skladě, je nutné vytvořit převodky z ostatních skladů na centrální, a poté ihned na cílový. Vytvoření převodky ze skladu se skladovou zásobou požadovaného zboží na centrální sklad proběhne podobně, jako u převodky z centrály na cílový sklad, jen vznikne navíc vazba mezi položkou objednávky a položkou převodky. Při příjmu převodky ze zdrojového skladu na centrálním skladě pak díky této vazbě je jasné, že je nutné přijímanou položku zařadit ihned na převodku z centrálního skladu na cílový sklad. Tím je zajištěno, že budou veškeré položky nakonec doručeny na cílový sklad. Funkci pro vygenerování převodek, lze vidět na ukázce 5.7.

5.3.4 Historie pohybů skladové karty

Implementace historie pohybů skladové karty není nikterak složitá. Jedná se o tabulku pro každou skladovou kartu obsahující záznamy o tom, kdo a kdy provedl změnu skladové zásoby skladové karty a v jakém úložišti. Protože je ovšem očekáváno velké množství zápisů do této tabulky, je využita pro tyto účely druhá databáze. Ta již nevyužívá framework Doctrine, ale základní třídy aplikačního rámce Nette pro práci s databází. Navíc v tomto případě není možné dopředu definovat všechny tabulky, jelikož skladové karty budou v průběhu doby v systému přibývat a ubývat. Hlavní motivací pro implementaci této funkčnosti je dohledatelnost změn skladové zásoby jednotlivých karet z důvodu ztráty či chyby uživatele.

5.4 Implementace komunikace s WS dopravců

Důležitou součástí skladového systému je napojení na webové služby dopravců. To umožňuje dát dopravci informaci o zasílaném balíku a vytisknout štítek na balík. Implementováno bylo napojení na webové služby firem Direct Parcel Distribution CZ s.r.o. (DPD) a Geis Parcel CZ s.r.o. Při implementaci obou bylo využito třídy *SoapClient*, která je součástí PHP.

Mikroslužba entity balíku, která má na starosti uzavření balíku a tím pádem i zajištění odeslání avíza dopravci, obsahuje referenci na objekt třídy *DelegatingShipperAdapter*. Ten v sobě obsahuje reference na jednotlivé adaptéry, pro komunikaci s webovými službami jednotlivých dopravců. Podstatnou implementací zde je funkce *register*, která obdrží podstatné informace o zásilce, deleguje jej příslušnému adaptéru, a mikroslužbě entity balíku vrátí odpověď obsahující štítek ve formátu PDF zakódovaný pomocí base64 a referenční číslo balíku u dopravce.

Implementaci funkce *register* lze vidět na ukázce 5.8. Delegování probíhá výběrem, který je určen podle zvoleného dopravce v objednávce od zákazníka.

5.4.1 GEIS

Objekt třídy *GeisShipperAdapter* má na starosti komunikaci s webovými službami dopravce GEIS. Obsahuje dvě veřejné metody - *register* a *closeShipment*. První zmíněná je určená k odeslání avíza o balíku dopravci a získání štítku tohoto balíku. Ta nejprve volá funkci *insertPackage*, kterou lze vidět na ukázce kódu 5.9. Obsahuje volání webové služby pro ode-

```
public function register(PackageInfo $packageInfo): ShipperResponse {
   switch ($packageInfo->getShipper()) {
      case GeisShipperAdapter::GEIS_NAME:
         return $this->geisShipperAdapter->register($packageInfo);
      case DPDShipperAdapter::DPD_NAME:
         return $this->dpdShipperAdapter->register($packageInfo);
      default:
         throw new UnknownShipperException;
   }
}
```

Obrázek 5.8: Funkce delegování odpovědnosti za odeslání avíza o balíku [vlastní]

slání avíza. Nejprve je pomocí funkce *createRequest* získáno pole obsahující nezbytné informace o zásilce, které je poté součásti odesílané zprávy.

```
private function insertPackage(PackageInfo $packageInfo) {
   try {
      $request = $this->createRequest(
         $this->createInsertExportRequest(
            $packageInfo
         )
      );
      $response = $this->soapClient->InsertExport(
         array('Request' => $request)
      );
   } catch (SoapFault $fault) {
      throw new SoapCommunicationException(
         'SOAP',
         1300,
         'CODE: ' . $fault->getCode() . ' ,MSG: ' .$fault->getMessage()
      );
   }
   $this->checkInsertResponseForError(
      $response
   );
   return $response->InsertExportResult
      ->ResponseObject
      ->PackNumber;
}
```

Obrázek 5.9: Funkce odeslání avíza o balíku [vlastní]

V případě, že proběhne vše v pořádku, je navráceno referenční číslo balíku přidělené dopravcem. V opačném případě vznikne výjimka obsahující informace o chybě. Poté je volána funkce *getLabel*, která zajišťuje získání štítku balíku. Funkce je velmi podobná té předchozí. Tentokrát ovšem výsledkem není referenční číslo balíku, ale data souboru PDF, který obsahuje štítek určený k tisku na balík.

Druhou podstatnou funkcí této třídy je *closeShipment*. Ta se volá v okamžiku příjezdu dopravce do skladu. Má totiž za následek získání soupisu balíků určených k odeslání, které jsou nutné pro řidiče přepravní společnosti.

5.4.2 DPD

Implementace volání webových služeb dopravce DPD je velmi podobná, jako u předchozího dopravce. Implementováno je v třídě *DPDShipperAdapter*. Rozdíly jsou zejména ve formátu odesílaných dat. Navíc webové služby DPD mají hned dva cílové body (end point), na které je třeba zasílat požadavky. První slouží jen k odesílání avíz o balících a získávání štítků, druhý je určen k uzavření soupisu zásilek před příjezdem dopravce a vytisknutí tohoto soupisu, jež je nutný při předání zásilek dopravci.

Rozdílná je i implementace odesílatele. Zatímco u předchozího dopravce bylo možné pro každý balík zadat odesílatele zvlášť z informací o balíku či objednávce, v případě DPD je nutné zadávat identifikátor odesílatele, který je zaregistrován přímo u tohoto dopravce.

5.5 Popis implementace automatizovaných testů

V rámci implementace byly vypracovány automatizované jednotkové a integrační testy pro ověření správné funkčnosti a odladění základních chyb aplikace a jednotlivých částí. K tomuto účelu byl využit nástroj Nette Tester, který je součástí Nette. Veškeré testy lze spustit jediným příkazem, viz. první příkaz v ukázce 5.10. Samostatně lze spouštět testy nad jednotlivými kontexty, což umožňuje druhý příkaz ze stejné ukázky. K účelům testování se využívá databáze SQLite.

```
vendor/bin/tester -p php -c /PATH_TO_PHPINI/php.ini tests/
vendor/bin/tester -p php -c /PATH_TO_PHPINI/php.ini tests/src/Core/Warehouse/
```

Obrázek 5.10: Příkaz pro spuštění automatizovaných testů a spuštění testů nad vybraným kontextem [vlastní]

Ve vrstvě aplikačního rozhraní jsou implementovány testy nad jednotlivými třídami typu *Presenter*. Jedná se o integrační testy, ve kterých se ověřuje správné chování aplikace jako celku, tedy od přijetí HTTP požadavku až po promítnutí změn do databáze. Kontroluje se správný tok dat, ověřování vstupních parametrů či zpracování chybových stavů. Zároveň je tato úroveň testů velmi důležitá i pro ověření korespondence mezi definovaným aplikačním rozhraním v dokumentaci a skutečným stavem.

Příklad testovací funkce lze vidět na ukázce 5.11. Jedná se o test funkčnosti změny velikostní kategorie pro vybranou skladovou kartu. Funkce *runApp()* má na starost volání aplikace na zadané adrese s vybranou metodou, v našem případě PUT, s příslušnými parametry, tak jak je definuje směrovací tabulka aplikace. Nejprve se ověřuje správný návratový kód, načež se získá detail skladové karty, který se porovná s očekávaným výsledkem.

Druhou důležitou částí automatizovaného testování jsou testy nad doménovou úrovní. Ty by se daly rozdělit na dvě části, a to jednotkové a integrační testy. Jednotkové testy se využívají především u doménových objektů a repositářů. Příklad jednotkového testu nad doménovým objektem skladové karty lze vidět na ukázce 5.12. Nejprve se pomocí privátní funkce vytvoří doménový objekt skladové karty, načež se pomocí veřejné metody tohoto objektu změní jeho velikostní kategorie. Výsledkem testu je porovnání, zdali byla tato změna úspěšně v objektu propagována.

```
public function testChangeSizeCategory() {
   $sizeCategoryUuid = Uuid::uuid4();
   $uuid = $this->createCard();
   $sizeCategory = ['sizeCategory' => (string) $sizeCategoryUuid];
   $responseA = $this->runApp(
      'PUT',
      '/card/' . $uuid->toString() . '/sizeCategory',
      $sizeCategory
   );
   $this->assertEquals(204, $responseA->getStatusCode());

   $responseB = $this->runApp('GET', '/card/'. (string) $uuid);
   $actual = json_decode((string) $responseB->getBody(), TRUE);
   $expected = $this->getCardDetail($uuid, sizeCategoryUuid);
   $this->assertEquals($expected, $actual);
}
```

Obrázek 5.11: Příklad testu presenteru pro změnu velikostní kategorie skladové karty [vlastní]

Integrační testy na doménové úrovni jsou implementovány nad aplikační podvrstvou, tedy nad třídami mikroslužeb, kde se opět testuje správná komunikace mezi jednotlivými mikroslužbami a repositáři, včetně promítnutí změn v databázi.

```
public function testChangeSizeCategory() {
   $card = $this->createFullCard();
   $sizeCategory = new SizeCategoryId();
   $card->changeSizeCategory($sizeCategory);
   Assert::equal(
      getProperty($card, 'sizeCategory')
         ->getSizeCategoryId()
         ->toString(),
      $sizeCategory->toString());
}
```

Obrázek 5.12: Příklad testu třídy *CardPresenter* pro změnu velikostní kategorie skladové karty [vlastní]

Kapitola 6

Testování a zhodnocení výsledků

Kapitola testování se věnuje ověření funkčnosti a správnosti implementovaného skladového informačního systému. V úvodu jsou popsány výsledky automatizovaného testování, a to jak testů jednotkových, tak integračních. V druhé části jsou pak popsány uživatelské testy, včetně zhodnocení výsledků těchto testů. Kromě dvou zmíněných typů testů byla aplikace i její součásti průběžně testovány během vývoje i pomocí debugování, čímž byly odstraněny nejzásadnější chyby v aplikaci.

6.1 Automatizované testy

Implementace automatizovaných testů byla popsána v předchozí kapitole, proto bude tato část práce věnována především vyhodnocení těchto testů. Velký význam mají automatizované testy především v průběhu implementace, kdy se nejprve napíše test určité části aplikace či funkce, který definuje správné a očekávané chování, načež se požadovaný kus kódu implementuje. Díky tomu jsou ihned odhaleny nejzásadnější chyby v implementaci. Zároveň jsou testy užitečné v případě, kdy se v další fázi implementace zasahuje do již existujícího kódu. Tehdy lze pomocí těchto testů ověřit, že se do programu nezanesly nové chyby.

Vrstva aplikačního rozhraní obsahuje celkem 109 testů, přičemž se jedná vždy o integrační testy jednotlivých funkcí tříd typu *Presenter* každé entity. Na doménové vrstvě bylo implementováno celkem 340 jednotkových a integračních testů. Dohromady je tedy obě vrstvy aplikace pokrývá celkem 449 testů. Veškeré automatizované testy končí úspěšně.

6.2 Uživatelské testy

V rámci uživatelského testování, ať už interního nebo ve skladě, byly vždy testovány nosné procesy, tedy příjem zboží, vyhledání a expedice objednávek. Přestože má být cílem této práce vytvořit pouze backend skladového informačního, všechny testy probíhaly nad kompletní webovou aplikací. Byl tedy testován najednou jak backend, tak frontend. Protože ale frontend nebyl součástí této práce, nejsou chyby související s touto částí aplikace a poznatky k němu zahrnuty ve výsledcích testů.

Během interního testování byly odladěny největší problémy a chyby na obou částech aplikace. V každé z fází testování je zobrazena tabulka s jednotlivými procesy rozdělenými do specifických operací, načež je vždy zobrazen konečný výsledek testu, nalezené chyby

a připomínky. Každý test byl v rámci každé fáze mnohokrát opakován pro odhalení co nejširšího spektra problémů.

Entita - Operace	**Chyba**	**Poznámka**	**Vyřešeno**
		Přidat dodavatelský kód ke skladové kartě	ANO
Storage - changeEanAmount	Neustále vrací 404		ANO
Storage - detail	Chybějící atribut obrázku skladové karty v detailu obsahu uložiště		ANO
	Vyhledávací slot není provázán se skladem, nelze určit, ve kterém skladě se nachází		ANO
SearchingSlot - assignOrderToStorage	Při označení některé objednávky jako nenalezené neustále vrací chybu kompletně nenalezeno		ANO
SearchingSlot - orders	Špatné výstupní data v přehledu objednávek, chybí množství jednotlivých EANů		ANO
SearchingSlot - orders		Zbytečný atribut balícího úložiště	ANO
SearchingSlot - detail	V případě označení všech objednávek jako nenalezené se vyhledávací slot neukončí		ANO
Package - close	Vrácení štítku od dopravce v nesprávném formátu		ANO
PackingSlot - assignSlotToUser	Nelze přiřadit balící slot, přestože v systému existuje volný		ANO
Package - addGoods	Lze do balíku vložit něco, co do objednávky nepatří		ANO
PackingSlot - close	Nekontroluje se obsah balíků při uzavírání slotu, tedy zdali bylo vše zabaleno		ANO
Package - addGoods	Nelze do balíku vložit dvakrát stejný EAN, respektive nenavyšuje se množství		ANO
		Chybí metoda přeskladnění zboží	ANO

Tabulka 6.1: Tabulka odhalených chyb během interního testování

6.2.1 Interní testování

Interní testování odhalilo chyby nedůležité i zásadní, ovlivňující funkčnost kompletních procesů. Důležité chyby odhalené během testování jsou popsány v přiložené tabulce 6.1. Testování bylo prováděno nad interně vytvořeným vzorkem dat.

6.2.2 Testování ve skladu

Testování bylo prováděno nad vhodně vybraným vzorkem dat dodaným zadavatelem, přičemž na přiloženém CD je SQL skript obsahující tyto vzorová data. První testování u klienta neodhalilo žádné zásadní chyby v základních procesech, přesto bylo sepsáno několik připomínek. Připomínky a problémy, jejichž zdrojem je backend aplikace, jsou popsány na následujících řádcích. Testovací scénáře včetně výsledků jsou zobrazeny v přehledných tabulkách níže.

Problémy se týkaly zejména entity skladové karty. U té je třeba řešit problém dodavatelského kódu, a především velikostní kategorie. Defaultně je nastavovaná velikostní kategorie jako střední, nicméně při příjmu by měl mít skladník možnost tuto vlastnost editovat. Zároveň v systému správně nefungovalo omezení příjmu, pokud skladová karta nemá zadané logistické údaje. Také je požadavek na možnost editace EANů u skladové karty i v detailu karty, nejen při příjmu. Všechny tyto problémy byly po testování odstraněny.

Dalším objeveným problémem, již daleko důležitějším, je možnost zastavení objednávky. Storno objednávky přichází z informačního systému elektronického obchodu a může přijít v různých fázích odbavení objednávky. Vzhledem ke složitosti tohoto procesu bylo rozhodnuto o prozatímním odložení implementace.

Tabulka 6.2 zobrazuje výsledky testů jednotlivých scénářů během procesů balení. Všechny testované scénáře proběhly dle očekávání a testy tak byly splněny. Zároveň v tabulce 6.3 lze vidět testování očekávaného chování systému při chybových vstupech či jiných podmínkách, které by měly zapříčinit chybu. Zde lze vidět již opravenou chybu u testu s logistickými údaji, která byla zmíněna již výše. Testování prošlo tedy bez chyby.

Scénář	**Podmínky**	**Očekávaný výstup**	**Výsledek**
Vložení zboží do úložiště	Typ vložení je příjem	Vytvořena příjemka, zboží uloženo v úložišti	OK
Změnit množství přijatého zboží	Zboží již bylo jednotlivě vloženo	Množství bylo změněno	OK
Přemístění naplněného úložiště do pozice	Pozice může obsahovat zboží z úložiště	Úložiště bylo zapozicováno	OK

Tabulka 6.2: Scénáře testované během procesu příjmu

Tabulka 6.4 zobrazuje výsledky testování jednotlivých scénářů procesu vyhledávání objednávek, zatímco tabulka 6.5 zobrazuje testování očekávaného chování při vybraných chybách.

Posledním nosným procesem, který byl podroben důkladnému testování, je proces balení. Výsledky testů jednotlivých scénářů lze vidět v tabulce 6.6. Očekávané chování při scénářích, které by měly vyvolat chybu, bylo rovněž testováno, přičemž výsledky těchto testů jsou zobrazeny v tabulce 6.7.

Scénář	**Podmínky**	**Očekávaný výstup**	**Výsledek**
Přihlášení	Neplatný uživatel	Chyba 1110	OK
Vložení zboží do úložiště	Zboží nemá zadáno logistické údaje	Chyba 1014	OK
Vložení zboží do úložiště	Neznámý EAN zboží	Chyba 1013	OK
Přemístění naplněného úložiště do pozice	Úložiště obsahuje zboží, které nesmí být v dané pozici (např. velké zboží v patře)	Chyba 1083	OK

Tabulka 6.3: Očekávané chyby testované během procesu příjmu

Scénář	**Podmínky**	**Očekávaný výstup**	**Výsledek**
Vytvoření vyhledávacího slotu	Vhodné objednávky ve frontě pro dané úložiště	Vytvořený vyhledávací slot s objednávkami	OK
Přidání zboží při vyhledávání	Zboží je v objednávkách	Zboží umístěno ve vyhledávacím úložišti	OK
Označení nenalezeného zboží	Zboží je dostupné v dalších úložištích	Vytvořen zápis o nenalezeném zboží	OK
Označení nenalezeného zboží	Zboží není dostupné v dalších úložištích	Vytvořen zápis o nenalezeném zboží, objednávky obsahující dané zboží jsou vyřazeny z vyhledávacího slotu	OK
Přidání zboží při vyhledávání	Jedná se o poslední hledaný kus	Vyhledávací slot je kompletně nalezen	OK
Přiřazení balícího úložiště	Objednávka byla nalezena	Přiřazeno balící úložiště	OK
Přiřazení balícího úložiště	Objednávka byla částečně nalezena, poté pozastavena	Přiřazeno úložiště pro nenalezené objednávky	OK
Označení objednávky jako přesunuté do balícího úložiště	Poslední objednávka z vyhledávacího slotu	Vyhledávací slot je uzavřen, zboží je umístěno v balícím úložišti, byl vytvořen balící slot	OK

Tabulka 6.4: Scénáře testované během procesu vyhledávání

Scénář	Podmínky	Očekávaný výstup	Výsledek
Vytvoření vyhledávacího slotu	Nevhodné objednávky ve frontě pro dané vyhledávací úložiště	Chyba 1082 nebo 1070	OK
Vytvoření vyhledávacího slotu	Nevhodné úložiště pro vyhledávání	Chyba 1080	OK
Vytvoření vyhledávacího slotu	Uživatel již má vytvořen vyhledávací slot	Chyba 1073	OK
Přiřazení balícího úložiště	Žádné balící úložiště není dostupné	Chyba 1008	OK
Přidání zboží při vyhledávání	Zboží není ve vyhledávaných objednávkách	Chyba 1071	OK
Přidání zboží při vyhledávání	Zboží již je nalezeno v požadovaném množství	Chyba 1072	OK
Přiřazení balícího úložiště	Objednávky nebyly kompletně dohledány	Chyba 1076	OK

Tabulka 6.5: Očekávané chyby testované během procesu vyhledávání

Scénář	Podmínky	Očekávaný výstup	Výsledek
Přiřazení balícího slotu	Existuje objednávka v balícím úložišti	Balící slot byl přiřazen, vytvořen první balík	OK
Přidání položky do balíku	Existuje otevřený balík k objednávce	Zboží přidáno do balíku	OK
Uzavření balíku	Existuje otevřený balík a balící slot k objednávce	Balík byl uzavřen, odesláno avízo na dopravce, vytisknut štítek	OK
Vytvoření nového balíku	Neexistuje otevřený balík k objednávce	Balík byl vytvořen	OK
Odeslání balíku	Všechny balíky objednávky byly uzavřeny	Balík byl odeslán, byla vytvořena výdejka	OK
Uzavření vyhledávacího slotu	Všechny balíky objednávky byly odeslány	Vyhledávací slot byl uzavřen	OK

Tabulka 6.6: Scénáře testované během procesu balení

Scénář	**Podmínky**	**Očekávaný výstup**	**Výsledek**
Přiřazení balícího slotu	Neexistuje objednávka v balícím úložišti	Chyba 1019	OK
Přiřazení balícího slotu	Uživatel má již přiřazený balící slot	Chyba 1024	OK
Vytvoření balíku	Objednávka má již vytvořený balík	Chyba 1020	OK
Vložení zboží	Zboží je v balících k objednávce nadbytečné	Chyba 1063	OK
Uzavření balícího slotu	Balíky objednávky neobsahují veškeré zboží objednávky	Chyba 1064	OK

Tabulka 6.7: Očekávané chyby testované během procesu příjmu

Celkem bylo tedy otestováno 33 scénářů, při kterých se ověřila funkčnost implementovaného skladového informačního systému dle požadavků zadavatele.

Kapitola 7

Závěr

Cílem této práce bylo analyzovat a specifikovat požadavky a následně pak navrhnout a implementovat backend skladového informačního systému pro prodejce sportovního vybavení, respektive jeho prototyp. V první části práce byla rozebrána teorie související s návrhem a implementací, načež byla další kapitola věnována specifikaci požadavků s důrazem na přesné identifikování procesů probíhajících ve skladě. Na základě této specifikace byl vytvořen kompletní návrh skladového informačního systému, včetně návrhu architektury a databáze. Důležitou součástí návrhu byla i specifikace rozhraní pro to, jak spolu bude backend a frontend aplikace komunikovat, k čemuž byl využit styl REST. Další část práce byla věnována implementaci. K té byl vybrán jazyk PHP ve verzi 7, společně s využitím některých existujících aplikačních rámců, jako jsou Nette, Slim či Doctrine. Poslední část práce je věnována testování implementované aplikace, ve které byla ověřena funkčnost implementace a potvrzena správnost aplikace v souvislosti s existujícími procesy.

Prototyp skladového informačního systému se podařilo v rámci této práce úspěšně implementovat a jeho funkčnost ověřit v rámci testování. Prozatím byl nasazen pouze na vývojovém a testovacím serveru, přičemž nasazení kompletního systému včetně plánovaných rozšíření uvedených v další části se plánuje na léto 2017. V rámci nasazení systému je v plánu nasadit i nový informační systém elektronického obchodu, s nímž bude skladový systém komunikovat.

7.1 Budoucí rozšíření skladového systému

V nejbližší budoucnosti bude následovat implementace dalších části skladového informačního systému. První z nich je zavedení nového dopravce, jímž je Česká pošta. Ačkoliv dle původního přání zákazníka měla být tato implementace již součástí této práce, bohužel kvůli problémům s vystavením certifikátu a chybami v nově zaváděných webových službách České pošty se tato část implementace odložila.

Druhou částí, která se bude brzy implementovat, je proces přeskladnění zboží z palet do běžných úložišť. Důvodem je fakt, že zboží v paletách není dostupné pro vyhledávání, proto je třeba vždy vedoucího skladu upozornit na případ, kdy je zboží dostupné pouze v paletách.

V plánu je také implementace možnosti zastavení objednávky v různých fázích odbavení, jelikož byl tento poznatek konzultován při testování. Problém tkví především ve fázi balení, kdy je nutné vyladit postup pracovníka balení při náhlém stornování objednávky.

Další plánované vylepšení systému je rezervace skladových zásob. Jedná se o proces, kdy jsou příchozí objednávky zařazeny buď do stavu připravena k vychystání, je-li k dispozici všechno zboží, popřípadě do stavu čekajících objednávek, není-li dostatek zboží na skladě. K tomu bude využito rezervace zásob, která upraví skladovou zásobu na stav bez zboží přiřazeného jednotlivým objednávkám.

Literatura

[1] Carda, A.; Kunstová, R.: *Workflow: Nástroj manažera pro řízení podnikových procesů*. Grada Publishing, a.s., 2003, ISBN 9788024762500.

[2] Evans, E.: *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004, ISBN 9780321125217.

[3] Fowler, M.: *CQRS [online]*. 2011-07-14 [cit. 2016-12-18], [Online; citováno 31.12.2016]. URL http://martinfowler.com/bliki/CQRS.html

[4] Interval.cz: *10 nejlepších PHP frameworků pro vývojáře*. [Online; citováno 1.1.2017]. URL https://www.interval.cz/clanky/10-nejlepsich-php-frameworku-pro-vyvojare/

[5] Millett, S.: *Patterns, Principles and Practices of Domain-Driven Design*. Wiley, 2015, ISBN 9781118714706.

[6] Nette Foundation: *About the Foundation | Nette Foundation*. [Online; citováno 1.1.2017]. URL https://nettefoundation.com/cs/

[7] Parent-Wiki: *Methodological guidelines and recommendations for efficient and rational governance of patient registries*. [Online; citováno 14.12.2016]. URL http://parent-wiki.nijz.si/index.php

[8] Rayan, J. C.: *Data modelling using ERD with Crow Foot Notation*. [Online; citováno 14.5.2017]. URL https://www.codeproject.com/Articles/878359/Data-modelling-using-ERD-with-Crow-Foot-Notation

[9] Richardson, S., L. a Ruby: *RESTful web services*. Farnham, 2007, ISBN 0596529260.

[10] Robbins, J. N.: *Learning Web Design: A Beginner's Guide to HTML, Graphics, and Beyond*. O'Reilly Media, Inc., 2003, ISBN 0596004842.

[11] Russell, K.: *Representational State Transfer (REST. Computerworld*, ročník 41, č. 32, Aug 06 2007: str. 40.

[12] The Dia Developers: *BPMN*. [Online; citováno 18.12.2016]. URL http://dia-installer.de/shapes/BPMN/index.html.es

[13] W3Techs: *Usage Statistics and Market Share of Server-side Programming Languages for Websites, December 2016 [online]*. 2016-12-30 [cit. 2016-12-31], [Online; citováno

31.12.2016].
URL https://w3techs.com/technologies/overview/programming_language/all

[14] Österle, H.: *Business in the Information Age: Heading for New Processes*. Springer Science & Business Media, 2013, ISBN 3662030861.

[15] Řepa, V.: *Podnikové procesy: Procesní řízení a modelování*. Grada Publishing, a.s., 2006, ISBN 80-247-1281-4.

Přílohy

Příloha A

Obsah přiloženého CD

textSource/ Zdrojové soubory k textů práce

fig/ Adresář obsahující veškeré navržené diagramy a obrázky

fig/models/ Modely analyzovaných procesů

fig/architecture/ Diagramy návrhu aplikace

fig/diagrams/ ER diagramy návrhu databáze

interface/ Adresář obsahující dokumentaci navrženého rozhraní aplikace

source/ Zdrojové soubory implementované aplikace

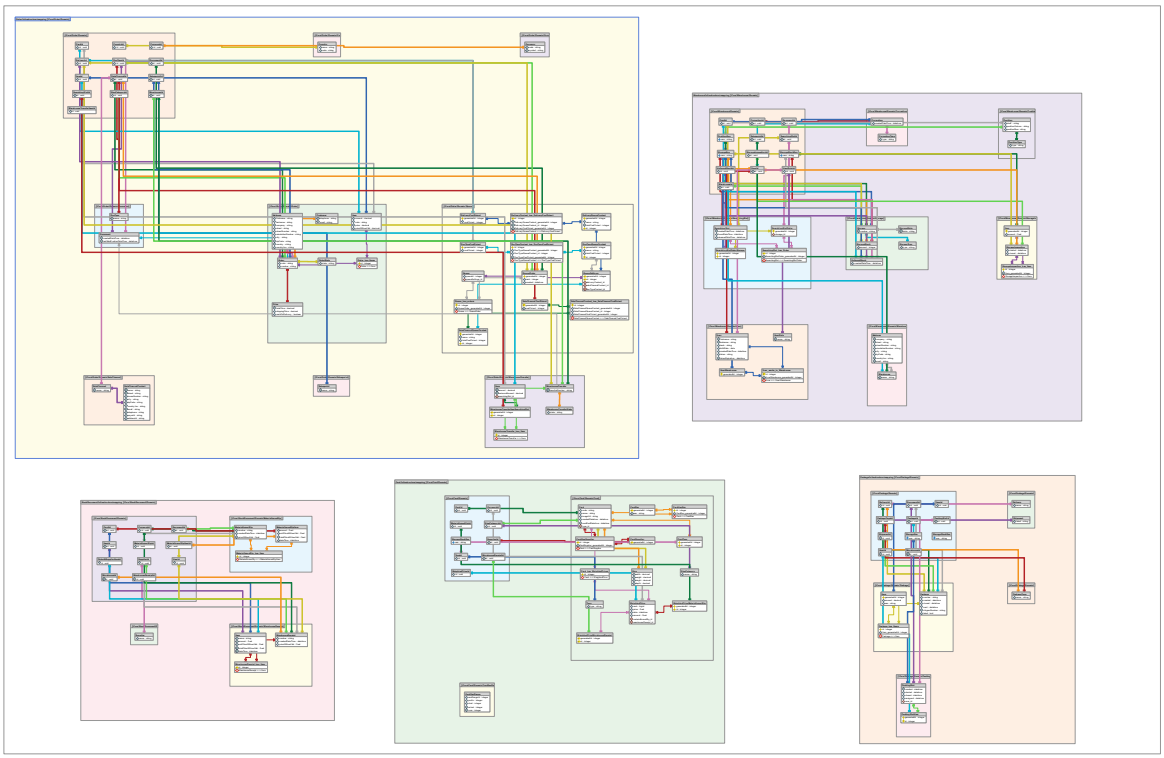
source/Readme.md Readme soubor s informacemi o aplikaci

source/sampleData/ SQL skripty se vzorovými daty použitými při testování

text.pdf PDF verze této práce

Příloha B

Kompletní ER diagram



Obrázek B.1: ER diagram návrhu databáze [vlastní]