VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PORADCE PŘI NÁKUPECH

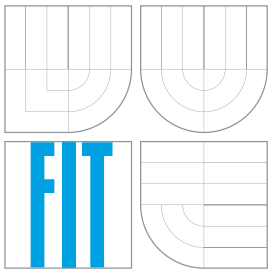
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB KOCOUREK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PORADCE PŘI NÁKUPECH

SHOPPING ADVISOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB KOCOUREK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. DUŠAN KOLÁŘ,

BRNO 2010

Abstrakt

Tato práce se zabývá problematikou nalezení nejkratší cesty v obchodu pro zvolený nákup. Jsou zde popsány technologie pro tvorbu www stránek. Pro procházení stavového prostoru jsou uvedeny algoritmy k nalezení nejkratší cesty mezi dvěma body a k řešení problému obchodního cestujícího. Dále je uveden návrh a implementace aplikace se všemi použitými technologiemi (PHP, Nette, MySQL, JS, JQuery) za použití Dijkstrova algoritmu pro nalezení nejkratší cesty a genetického algoritmu pro řešení problému obchodního cestujícího.

Abstract

The attached bachelor thesis deals with finding the shortest ways in a shop for the selected purchase. The different technologies of creating web pages are described. As for the state space search algorithms are given for finding the shortest path between two points and for solving the travel salesmen problem. The thesis continues with the description of a design and implementation of application with all used technologies (PHP, Nette, MySQL, JS, JQuery) applying the Dijkstra's algorithm for finding the shortest path and genetic algorithm for solving the travel salesmen problem.

Klíčová slova

Problém obchodního cestujícího, genetický algoritmus, simulované žíhání, nejkratší cesta v grafu, Dijkstrův algoritmus, Bellmanův-Fordův algoritmus, Floydův-Warshallův algoritmus, MySQL, PostgreSQL, MS SQL, Oracle, PHP, Java, ASP.NET, Perl, Nette, Javascript, JQuery.

Keywords

Travelling salesman problem, genetic algorithm, simulated annealing, shortest path problem, Dijkstra's algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm, MySQL, PostgreSQL, MS SQL, Oracle, PHP, Java, ASP.NET, Perl, Nette, Javascript, JQuery.

Citace

Jakub Kocourek: Poradce při nákupech, bakalářská práce, Brno, FIT VUT v Brně, 2010

Poradce při nákupech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Dr. Ing. Dušana Koláře a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Kocourek
18. května 2010

Poděkování

Děkuji vedoucímu bakalářské práce panu Doc. Dr. Ing. Dušanu Kolářovi za odborné vedení, rady, cenné připomínky a osobní konzultace při tvorbě této práce.

© Jakub Kocourek, 2010.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**
1.1 Struktura práce . . . 3

2 Technologie pro WWW aplikace **4**
2.1 Databázové systémy . . . 4
2.1.1 MySQL . . . 5
2.1.2 PostgreSQL . . . 5
2.1.3 MS SQL . . . 5
2.1.4 Oracle . . . 5
2.2 Programovací jazyky . . . 6
2.2.1 PHP . . . 6
2.2.2 Java . . . 7
2.2.3 ASP a ASP.NET . . . 8
2.2.4 Perl . . . 9

3 Systém pro zobrazení, zadávání a uložení organizace zboží **10**
3.1 Specifikace požadavků . . . 10
3.2 Diagram případu užití . . . 10
3.3 Návrh databáze . . . 11
3.4 Implementace . . . 12
3.4.1 Editor . . . 12
3.4.2 Běžné zobrazení . . . 13
3.4.3 Historie změn . . . 13

4 Uživatelské rozhraní **16**
4.1 Specifikace požadavků . . . 16
4.2 Diagram případu užití . . . 16
4.3 Návrh databáze . . . 16
4.4 Implementace . . . 16
4.4.1 Uživatelský profil . . . 18

5 Systém ideálního průchodu hypermarketem **19**
5.1 Prohledávání stavového prostoru . . . 19
5.2 Hledání nejkratší cesty . . . 19
5.2.1 Dijkstrův algoritmus . . . 20
5.2.2 Bellmanův-Fordův algoritmus . . . 20
5.2.3 Floydův-Warshallův algoritmus . . . 20
5.3 Problém obchodního cestujícího . . . 20

5.3.1 Genetické algoritmy 21
5.3.2 Simulované žíhání 22
5.4 Návrh 23
5.4.1 Specifikace požadavků 23
5.4.2 Návrh databáze 23
5.5 Implementace 23
5.5.1 Generování bodů a cest 23
5.5.2 Vyhledávání 24

6 Závěr **25**

A Kompletní Use-case diagram **29**

B Kompletní ER diagram **30**

C Obsah CD **31**

Kapitola 1

Úvod

Bakalářská práce Poradce při nákupech se zabývá problémem nalezení nejkratší cesty v obchodě pro požadovaný nákup. Využití je velmi jednoduché a praktické, místo pobíhání po hypermarketu s plným vozíkem a hledání poslední položky v seznamu si můžete celý nákup jednoduše vyhledat a vygenerovat pro něj nejkratší cestu. Díky tomu nestrávíte v obchodě ani o minutu déle než je potřeba a můžete tak svůj čas využít mnohem zábavnějším a efektivnějším způsobem. Navíc Vám Vaše nákupy zůstanou uloženy v historii a můžete je pro příště znovu využít.

1.1 Struktura práce

Dokument je strukturován tak, jak probíhal vývoj aplikace řídící se zadáním práce. V kapitole 2 se věnuji použitým technologiím pro WWW aplikace, od databázových systémů po programovací jazyky na straně klienta a serveru s popsáním důvodů, proč jsem se pro vybranou technologii rozhodl. Následuje návrh a implementace systému pro zobrazení, zadávání a uložení organizace zboží v obchodě v kapitole 3. Navazuje kapitola 4 s podobnou strukturou, kde se věnuji návrhu a implementaci uživatelského rozhraní, sloužící pro registraci, přihlašování a přístupu ke zboží a historii nákupů. Kapitola 5 je zaměřena na nalezení ideálního průchodu obchodem. Obsahuje základní seznámení s procházením stavového prostoru a algoritmy pro nalezení nejkratší vzdálenosti mezi dvěma body a pro řešení problému obchodního cestujícího s implementací použitou v této aplikaci. Poslední kapitola 6 obsahuje zhodnocení dosažených výsledků a možný budoucí vývoj práce.

Kapitola 2

Technologie pro WWW aplikace

Pro vytváření dynamických www aplikací je nutno nejdříve pečlivě vybrat mezi řadou technologií k tomu potřebných, od systémů pro databáze po programovací jazyky a jejich knihovny. Rozhodující je především dostupnost, rychlost, stabilita, bezpečnost, cena a samozřejmě jednoduchost. U programovacích jazyků rozhoduje také znovupoužitelnost a množství doplňků či knihoven, jenž jsou k dispozici, a kterými lze znatelně urychlit vývoj.

Na poli jazyků spouštěných na straně klienta už je rozhodování jednodušší, většina těchto skriptů je napsána v *JavaScriptu*. Existuje sice konkurenční *VBScript* od společnosti *Microsoft*, ale ten je podporován pouze prohlížeči od stejnojmenné firmy a pro širší využití je tedy nepoužitelný.

Větší možnosti jsou ve výběru frameworku pro *JavaScript*. Samozřejmě lze vše napsat bez pomoci jiných knihoven, ale vývoj se tím stává značně neefektivní a pomalý. Nejznámějším a nejpoužívanějším je *JQuery*. Jednoduchý, malý, ale přesto velmi efektivní framework zavádí několik vylepšení klasického *JavaScriptu*, jako např. selectory (umožňují vybrat jakékoliv prvky stránky a dále s nimi manipulovat), řízení událostí, animace a další. Ke knihovně je k dipsozici také balíček *UI*, jenž obsahuje metody pro manipulaci s objekty pomocí myši, kalendář, tabulky, posuvník a mnoho jiných užitečných prostředků. Další předností je aktivní vývojářská komunita, díky které je k dispozici velké množství doplňků volně ke stažení.

Z ostatních frameworků je nutno zmínit *Prototype*, obsahující sadu funkcí pro zjednodušení neefektivních JavaScriptových metod, selectory nebo intuitivnější práci s objekty. Poslední knihovna, o které se zmíním, je *MooTools*, která se naopak snaží o vylepšení a rozšíření původního jazyka s plnou podporou objektového programování a kompletní dokumentací.

2.1 Databázové systémy

Pro ukládání statických dat, které jsou potřeba po delší dobu života aplikace, a s kterými je nutno dále pracovat, se využívají databáze a nahrazují tak běžné dokumenty, jako jsou např. kartotéky, rejstříky a registry. Elektronická podoba těchto úložišť přináší samozřejmě výhody, jako je snadné a velmi rychlé prohledávání záznamů, jednoduchá aktualizace stávajících dat a také možnost kombinace s dalšímy systémy či aplikacemi.

Data jsou uložena ve formě tabulek a jsou oddělena od klientských aplikací, tím je zajištěna nezávislost a ke stejné databázi je možno přistupovat z více programů. K jednotlivým záznamům přistupují přes dotazovací jazyk, který umožňuje alespoň výběr, omezení,

projekci a spojení. [13]

I zde je samozřejmě možné vybírat z více produktů, které se od sebe liší cenou, rychlostí, funkčností nebo nástroji pro správu. Syntaxe vychází u všech typů ze standartního dotazovacího jazyka *SQL*, rozdíly jsou mezi nimi minimální a přechod na jiný systém není z programátorského hlediska obtížný. Problém může nastat při přechodu s již vytvořenými daty, i když jsou si jazyky podobné, nemusí používat stejné datové typy a drobné odchylky v syntaxi znemožňují použití totožných příkazů.

2.1.1 MySQL

MySQL je multiplatformní databázový systém vyvinutý švédskou společností *MySQL AB*. V dnešní době, s více než 100 milióny stáhnutých nebo jinak rozšířených kopií, je na vrcholu bezplatných databázových programů. Je klíčovou součástí tzv. *LAMP* (*Linux*, *Apache*, *MySQL*, *PHP*), představující platformu svobodného softwaru pro tvorbu dynamických www stránek. [18]

Při výběru databáze jsem již ze začátku vyřadil komerční *Oracle* a *MS SQL*, který je navíc k dispozici pouze na platformách *Microsoft Windows*. Co se týče rychlosti, tak je *MySQL* optimalizováno pro menší databáze a v porovnání s *PostgreSQL* dosahuje lepších výsledků. Pokud ovšem tabulka naroste do velikosti v řádech Gb, je na tom znatelně lépe konkurenční *PostgreSQL*. Poslední aspekt, který rozhodl pro *MySQL*, jsou nástroje pro správu databáze, jež jsou reprezentovány aplikací *phpMyAdmin* umožnující kompletní přehled o datech, tabulkách a databázích. Existuje obdobný nástroj pro *PostgreSQL*, pod názvem *phpPgAdmin*, ale bohužel nedosahuje takových kvalit a neobsahuje stejné množství možností.

2.1.2 PostgreSQL

PostgreSQL je výkonný multiplatformní objektově-relační databázový systém s více než 15 letou historií aktivního vývoje. Popularitu si získal především stabilitou, bezpečností, integritou dat a nulovou cenou. Občas bývá využit i v platfromě *LAMP* namísto *MySQL* a úspěšně mu tak konkuruje na poli bezplatných databázových systémů. Pro správu jsou k dispozici nástroje *psql* (příkazová řádka), desktopová aplikace *pgAdmin* nebo www rozhraní *phpPgAdmin*, ovšem jak jsem již zmínil, nejsou to plně uspokojující aplikace.[20]

2.1.3 MS SQL

Microsoft SQL je relační databázový systém, vyvinutý společností *Microsoft*, využívající dotazovací jazyk *SQL* a *T-SQL*. Jako většina produktů této firmy není systém multiplatformní a musí tedy počítat i s chybami operačního systému, na kterém běží. Dalším bodem, kterým se snižuje garantovaná dostupnost serveru, je nutnost restartu po aktualizaci. Z hlediska zátěže procesoru a spotřeby operační paměti na tom také není zrovna nejlépe a to je nutné ještě připočíst nároky operačního systému. Naopak nástroj *Management studio* pro správu databází je na velmi vysoké úrovni a poskytuje rychlejší, komplexnější a komfortnější využití než webový *phpMyAdmin*.[21]

2.1.4 Oracle

Oracle je multiplatformní databázový systém společnosti *Oracle Corporation*, využívající jednak standartní jazyk *SQL*, ale také vlastní procedurální nadstavbu *PL/SQL*. Mezi ko-

merčními systémy zaujímá přední pozici díky téměř dokonalé stabilitě, bezpečnosti, podpoře zákazníků a množství dalších vývojářských produktů, schopných spolu spolupracovat a urychlit tak vytváření a používání databázových aplikací. Obdobně jako *MS SQL*, značně zatěžuje server a klade velké nároky na operační paměť.[21][2]

2.2 Programovací jazyky

Ke správnému a efektivnímu vytvoření aplikace je potřeba vybrat správný programovací jazyk. Momentálně jsou všechny aktivně vyvíjeny a těžko se tedy dá nějaký vybrat s tím, že je nejlepší nebo naopak jiný vyloučit z důvodu, že by v něčem znatelně zaostával. Rozhodovat tak může počet a kvalita doplňků, knihoven nebo předchozí zkušenosti s daným jazykem.

2.2.1 PHP

PHP (*Hypertext Preprocessor*) je open-source multiplatformní skriptovací jazyk vytvořený pro tvorbu dynamického webu. Nejčastější použití je zakomponování do *(X)HTML*, tedy pro internetové stránky, ale lze využít i pro konzolové či desktopové aplikace. Momentálně se řadí mezi nejpoužívanější jazyky pro tvorbu www stránek. Příznivce si získal především svojí jednoduchostí, snadnou komunikací s databázemi, robustností a v neposlední řadě nulovou pořizovací cenou. Vzhledem k masové podpoře na poli webových hostingů jeho popularita neustále roste. Nutno poznamenat, že neroste pouze počet uživatelů, ale i celková kvalita jazyka. Podporou objektového programování se zařadil mezi moderní programovací jazyky a usnadnil tak psaní komplexnějších a „čistších“ aplikací.

Jedná se o jazyk vyšší generace, nedosahuje tudíž takových rychlostí jako například jazyk *C*, ale v porovnání s konkurencí na tom není špatně. Mnoho neoficiálních testů naznačuje, že se rychlostí minimálně vyrovná *ASP* a i ve většině aplikací snadno předčí *JSP*.[22]

PHP patří do skupiny jazyků, které se provádějí na straně serveru. Výhody jsou zřejmé. Nejsou kladeny žádné nároky na vybavení uživatele, vše se zařídí na serveru a klient už dostane pouze zpracovaná data. Dále klient nevidí zdrojový kód, ve kterém jsou citlivé informace, jako například přihlašovací údaje do databáze.

Obsahuje bezpočet funkcí, které jsou popsány na oficiálních stránkách [19]. Oblíbené jsou také frameworky, které vnáší další funkce a velmi efektivní a užitečnou architekturu *MVC* (*Model View Controller*), která je pro zkušenější vývojáře nebo firmy standartem, a bez které se dnešní pokročilé aplikace neobejdou.

Pro tuto práci jsem zvolil právě *PHP*, nenalezl jsem žádné podklady, které by potvrzovali, že ho nějaký jiný jazyk v čemkoliv převyšoval, ať už se jedná o rychlost, stabilitu nebo bezpečnost. Hlavní důvod byly také předchozí zkušenosti, které již s tímto jazykem mám a možnost využít framework *Nette* [5] a databázovou vrstvu *Dibi* [4].

Historie

Jazyk vznikl v roce 1994 v jazyce *Perl* a sloužil pouze pro správu a údržbu osobních stránek autora Rasmuse Lerdorfa (od toho se odvíjí i tehdejší název *Personal Home Page Tools*, což volně v překladu znamená *Nástroje pro osobní stránky*). Vzhledem k tomu, že *Perl* značně zatěžoval www server, přepsal jej Rasmus Lerdorf do jazyka *C*. V roce 1995 byla na žádost uživatelů, kteří si již stačili projekt vyzkoušet, sestavena první verze, jež brzy následovala verze *PHP/FI* (nebo *PHP 2.0*). Ta obsahovala nejen sadu mnoha nových funkcí, ale přinesla

i rozšíření o *SQL* modul, který umožnoval práci s databází a formuláři. Tím se otevřela cesta pro vytváření jednoduchých dynamických www aplikací.

V polovině roku 1997 využívalo *PHP* přibližně 50 000 stránek. Pro samotného autora začal být jazyk moc velkým soustem, a proto se tentýž rok k projektu přidali dva Izraelští vývojáři Zeev Suraski a Andi Gutmansa, kteří přepsali jádro jazyka a položili tím základy nové verze *PHP 3.0*. Ta spatřila světlo světa v roce 1998. V roce 2000 vyšla verze *PHP 4.0*, která obsahovala zcela nové jádro pod názvem *Zend 1.0*. Mezi těmito verzemi zaznamenal projekt raketový vzestup a ve stejném roce dosáhl hranice 2 miliónů domén využívajících jakýmkoliv způsobem *PHP*. Do roku 2008, kdy se přestala 4. verze vyvíjet, se postupně přidávala řada funkcí a bezpečnostních opatření. S novým jádrem *Zend 2.0* vyšla i nová verze *PHP 5.0* a to v roce 2004. Ta přinesla podporu objektového programování a v současné době je to jediná stabilní verze ve vývoji. Paralelně s touto verzí je vyvíjeno *PHP 6.0*, které má mít lepší podporu *Unicode* a nově možnost využití jmenných prostorů (*namespaces*).[6][22]

Nette

Pro ušetření zdlouhavé a rutinní práce, zvýšení přehlednosti zdrojových kódů, zvýšení bezpečnosti nebo zvýšení produktivity sáhne valná většina vývojářů v *PHP* pro framework. Existuje jich ovšem velký počet, od nejznámějších *Zend*, *Symfony*, *CakePHP*, *CodeIgniter* po ty menší, jako jsou *Kohana* nebo *Epicode* a je velmi obtížné si zvolit mezi nimi ten pravý, který uspokojí Vaše potřeby. Mezi tyto ambiciózní projekty se řadí také český framework vývojáře Davida Grudla *Nette*.

Největší silou, jak uvádí sám autor, je rychlost a bezpečnost. V porovnání s ostatními patří mezi ty nejrychlejší a ochrání Vaši aplikaci od útoků typu *XSS* (*Cross-Site Scripting*), *CSRF* (*Cross-Site Request Forgery*), *URL attack*, *control codes*, *invalid UTF-8*, *session hijacking*, *session fixation* a dalších. Je kompletně založen na objektovém programování, čímž přispívá k čistému návrhu aplikace a znovupoužitelnosti kódu. Další výhodou je jeho český původ, díky kterému existuje česká dokumentace a aktivní česká vývojářská komunita.[5]

2.2.2 Java

JAVA je plně objektově orientovaný multiplatformní programovací jazyk (neumožňuje již tvorbu programů, které by nebyly objektově orientované). Jeho univerzální využití, počínaje programy pro mobilní telefony a internetové aplikace, až po rozsáhlé aplikace pro firmy a desktopové aplikace, ho vysazuje na přední příčky všech používaných jazyků.

Pro www stránky se využívá platformy *Java EE*, která specifikuje technologie pro vývoj webových aplikací. Je několik možností, jak Javu využít (*Servlety*, *JSP*, *Apllety*, ...) a všechny se opírají o výhody tohoto jazyku (bezpečnost, přenositelnost, stabilita, ...), a proto velmi rychle získávají na oblibě.

První z nich jsou *servlety*. Komponenty běžící na serveru, jejichž hlavní využití je pro generování dynamického obsahu stránek (ukládání dat do databáze, zpracování formulářů, složitější výpočty atd).

JSP (*JavaServer Pages*) a *JSTL* (*JSP Standard Tag Library*) jsou technologie pro vývoj *(X)HTML* stránek s využitím jazyka *Java* pro vytváření dynamického obsahu. Zdrojový kód se píše do šablon *(X)HTML* podobně jako v *PHP*. *JSP* se využívá především u statických stránek, pro pokročilejší aplikace je vhodnější využít již zmíněných servletů. *JSTL*, jak už název napovídá, je standartizovaná knihovna *XML* značek, které lze využít při psaní *JSP* stránek.

Java Applet je program v jazyce *Java*, který se spouští z www stránky. Na rozdíl od servletu běží na straně klienta. Používá se pro interaktivní či jiné aplikace, které nejdou napsat pouze s *(X)HTML*. Applety mezi sebou mohou komunikovat a mohou například spolupracovat i s JavaScriptem.[7]

Historie

Jazyk *Java* vyvinula společnost *Sun Microsystems*. V roce 1990 se z iniciativy viceprezidenta firmy Billa Joye vytvořila vývojová skupina vedená Jamesem Goslingem. Měla za úkol vytvořit nový jazyk, který by umožňoval jednoduchý a efektivní zápis programů. Jejím prvním cílem bylo vytvořit systém pro domácí spotřebiče, který z důvodu různých komponentů v nich obsažených, měl být multiplatformní, bez nutnosti kompilovat zdrojový kód. Jelikož jazyk *C++*, a později ani *Pascal*, nesplňoval jejich požadavky, navrhl James Gosling nový interpretovaný jazyk jménem *Oak* (v překladu *Dub*).

Po dokončení ovšem nastal problém s uplatněním jazyka, vytvářet programy pro domácí elektroniku se na trhu neujal. Bill Joy (spoluzakladatel společnosti *Sun*) si ovšem všiml vzrůstajícího potenciálu internetu, ve kterém chyběl potřebný nástroj pro tvorbu dynamického webu schopného komunikovat s databázemi. Vznikly tzv. *Applety*, které vnášely nový rozměr do doposud statických prezentací.

V roce 1995 se zjistilo, že již jazyk stejného názvu existuje, a proto se jazyk *Oak* přejmenoval na *Java* (podle slangového označení kávy). V následujícím roce byl vydán první *JDK 1.0* (*Java Development Kit*), který obsahoval veškeré potřebné nástroje pro tvorbu appletů. V tomto roce také zaznamenala *Java* největší průlom díky nové verzi internetového prohlížeče *Netscape*, který dominoval trhu z 80%, a který zavedl podporu Java appletů.

Kvůli sporům se společností *Microsoft* se jejich operační systém přestal dodávat s aktuální verzí jazyku *Java* a po vzestupu *Internet Exploreru* pokleslo i používání Java appletů a aplikací. Naštěstí již v té době měla platforma *Java EE* používaná na serverech stabilní pozici a spory s Microsoftem ji nijak neohrozily.

V roce 1998 byl *JDK* nahrazen *J2SE* (*Java 2 Platform Standard Edition*), který vydržel až do dnešních dob, kdy je aktuální verze *Java SE 6*. Od 8.5.2007 se Java vyvíjí jako open source.[17][11]

2.2.3 ASP a ASP.NET

ASP (*Active Server Pages*) je serverové skriptovací prostředí, vytvořené firmou *Microsoft*, pro dynamické generování www stránek. Obsahuje několik základních tříd a metod, ale nelze vytvářet nové. Proto se většinou využívá s jiným skriptovacím jazykem. Nejvíce používaný je *VBScript*, ale lze použít i jiný (například *Perl*).

Občas bývá zaměňován s jiným skriptovacím prostředím od Microsoftu a to *ASP.NET*. I když je tento jazyk nástupcem *ASP*, ve značné míře se od něj liší. Lze využít ze širší nabídky podporovaných jazyků, jako například *C#*, *C++* nebo *Visual Basic*. Nově zavádí tzv. kontrolky, které umožňují oddělit funkční kód od *(X)HTML*, což v *ASP* možné nebylo. Jako největší výhodu ovšem považuji kompilaci kódu, která se provede po prvním zpracování serveru. Kromě zrychlení se stane kód nečitelný a je prakticky nemožné, aby ho někdo zkopíroval či jinak zneužil.[14]

Historie

První verzi *ASP 1.0* vydal *Microsoft* jako doplněk pro *IIS 3.0*(*Internet Information Services*) ve *Windows NT 4.0*. Díky velkému úspěchu byl poté použit jako součást *Windows 2000 Server*. V roce 1997 a 2000 byly vydány další a dodnes poslední verze *2.0* a *3.0*. Od roku 2002 je vývoj plně zaměřen na jeho následníka *ASP.NET*, jehož poslední verze *3.5* vyšla v prosinci 2007.[14]

2.2.4 Perl

PERL (*Practical Extraction and Report Language*) je interpretovaný dynamický programovací jazyk. Umožňuje procedurální, funkcionální i objektově orientované programování a s jeho více než 18 000 volně dostupnými moduly se řadí mezi nejuniverzálnější jazyky. Je vhodný jak pro jednoduché skripty, tak i pro ty složité. Téměř od počátku byl také často využíván pro dynamické ww stránky, díky kterým získal na popularitě.

Často bývá kritizován za svou značnou volnost v syntaxi. S možností minimalizovat zdrojový kód získal pověst jazyka, ve kterém se vytváří nečitelné a nesrozumitelné kódy. Právě díky této vlastnosti, ale získal mnoho příznivců. Za zmínku také stojí soutěž pod názvem *Obfuscated Perl Contest* z roku 1996 a 2000, kde je výhercem autor nejobdivuhodnějšího, nejúžasnějšího a hlavně nejzamotanějšího kódu.

Autorem jazyka Perl je bývalý pracovník *NASA* Larry Wall. Na projektu začal pracovat v roce 1986 pro osobní potřebu, za účelem zjednodušení zpracování textu a dálkové řízení počítačů. Ihned po vydání první verze v roce 1987 se projekt setkal s velkým ohlasem a během několika dalších let se stal jazyk velmi oblíbený. Následující rok byla vydána verze pod názvem *Perl 2.0*, která přinesla lepší podporu regulerních výrazů. Hned další rok byla vydána verze *Perl 3.0* s podporou binárních datových toků. S vydáním knihy *Programming Perl*, která se stala neoficiálním manuálem jazyka, vyšla nová verze *Perl 4.0*, která byla vydána pouze kvůli této knize a nepřinášela žádné zásadní změny.

V roce 1993 začala práce na verzi *5.0*, která vyšla v květnu dalšího roku. Obsahovala téměř kompletně přepsaný interpret jazyka, podporu objektového programování, řadu nových funkcí a v neposlední řadě také podporu modulů. Ty umožňovaly rozšířit jazyk o nové funkce, aniž by se muselo zasahovat do jádra jazyka a díky aktivní účasti vývojářů z celého světa se stala knihovna modulů jednou z největších zbraní tohoto jazyka. V současnosti je nejnovější stabilní verze *5.11.1*, ale od roku 2008 probíhají práce na verzi *6.0*.[1]

Kapitola 3

Systém pro zobrazení, zadávání a uložení organizace zboží

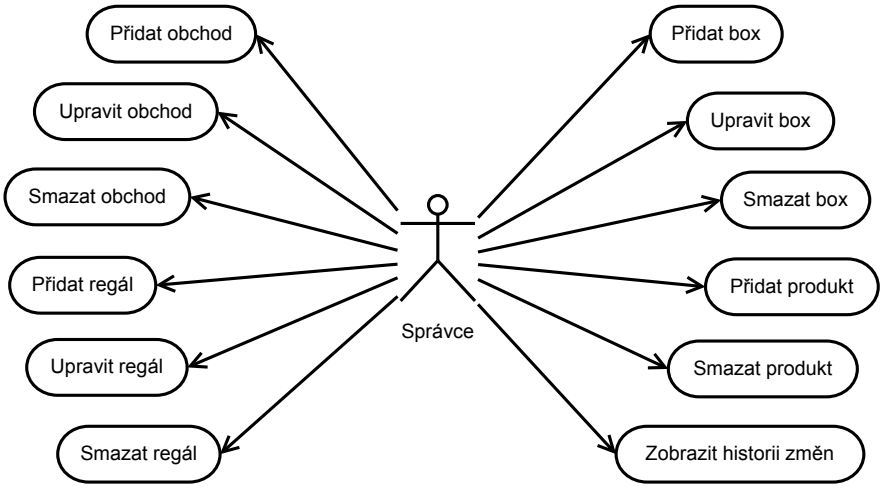
3.1 Specifikace požadavků

Systém musí být přehledný a jednoduchý, ale ne na úkor funkčnosti. Uživatel musí mít přehledně všechny použitelné nástroje viditelné a pokud možno umístěné na jednom místě. V obchodě jsou umístěny kromě regálů také zdi, pokladny nebo vstupní turnikety. Pro nastavení pozice a rozměrů těchto objektů navrhnout vhodný systém s přesným zadáním v metrech. Pro přehlednost zajistit alespoň barevné rozlišení různých typů regálů (mražené výrobky, zelenina, ...) a ostatních objektů (vstup, stěna, pokladna).

Regály mohou mít různou strukturu, pokud ho například uživatel umístí ke zdi, bude pro přístup a umístění zboží k dispozici pouze jedna stěna. Přepážky mezi těmito částmi by tedy měly být snadno posouvatelné a opět s možností zadání přesného rozměru. Regály se skládají z různých boxů obsahujících specifický druh zboží (pečivo, mléčné výrobky, ochucovadla, ...). Pro obchod, regály i boxy udržovat přesnou historii změn a umožnit převzít již jednou uložená data.

3.2 Diagram případu užití

Diagram na obrázku 3.1 znázorňuje funkce dostupné pro uživatele s administrátorskými právy. Jedná se pouze o možnosti v rámci editoru, kompletní diagram se všemi uživatelskými rolemi a metodami je v příloze A.



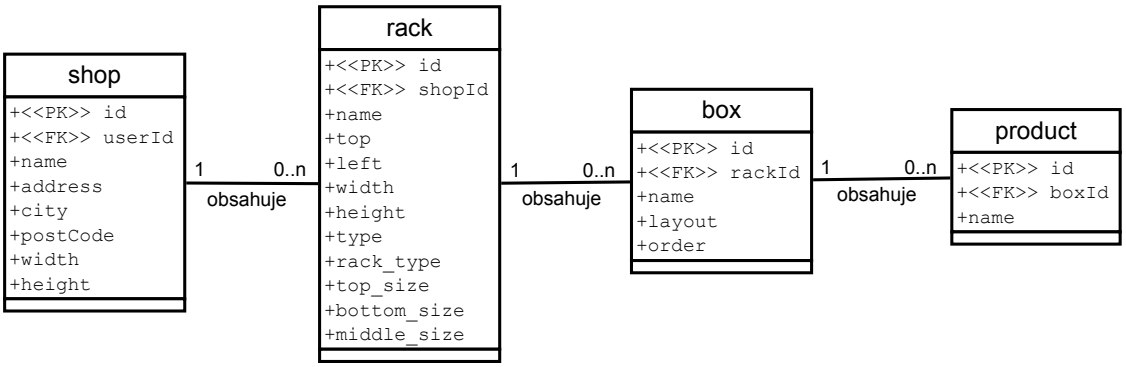
Obrázek 3.1: Část use-case diagramu pro editor

3.3 Návrh databáze

Na obrázku 3.2 je znázorněna část ER diagramu pro editor. V tabulce shop jsou nejdůležitější hodnoty *width* a *height* určující rozměry obchodu. Přes cizí klíč *userId* je obchod spojen s uživatelem, který obchod vytvořil, a který ho může dále upravovat nebo smazat.

Tabulka *rack* slouží pro uložení regálů, zdí, pokladen a vstupů. K uložení pozice slouží hodnoty *top* a *left*, rozměry jsou zaznamenány ve *width* a *height*. Sloupce *top_size*, *middle_size* a *bottom_size* představují vzdálenosti přepážek rozdělující regál na několik částí. V *rack_ type* lze přesněji specifikovat typ (např. mrazící box, ovoce a zelenina, . . .).

Pro přiřazení boxu k regálu slouží v tabulce box cizí klíč *rackId*, ale k přesnému určení pozice jsou potřeba ještě další dva sloupce a to *layout*, udávající v které části regálu se nachází a *order* určující pořadí, pokud existuje více boxů ve stejné části.



Obrázek 3.2: ER diagram editoru

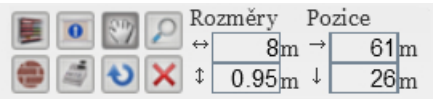
3.4 Implementace

Vzhledem k rostoucím možnostem a funkčnosti knihoven v JavaScriptu není potřeba sáhnout k jiným technologiím a pro vytvoření editoru obchodu bude stačit *XHTML* a framework *JQuery*, případně jiné doplňky založené na této knihovně. Pro serverovou část je využito *PHP* frameworku *Nette* a databázové vrstvy *Dibi* pro spolupráci s *MySQL*.

Nette je postaveno na modelu *MVC* (*Model-View-Controller*) zajišťující oddělení *PHP* kódu, *SQL* dotazů a *XHTML* šablon. Všechny dotazy a přístupy do databáze jsou umístěny v souborech ve složce *models*. Pro obsluhu modelů, generování hodnot a další například výpočetní funkce jsou použity tzv. *Presentery*, pro každou sekci aplikace je vytvořen nový a umístěn ve složce *presenters*. Poslední částí jsou *XHTML* šablony, ve kterých se generuje finální obsah. Ty jsou uloženy ve složce *templates*.

3.4.1 Editor

Panel nástrojů na obrázku 3.3 obsahuje sadu 8 tlačítek z toho první 4 jsou pro vytvoření nového regálu, zdi, vstupu nebo pokladny. Bloky mohou měnit velikost a polohu pomocí myši tzv. *Drag & Drop* (volně přeloženo *stiskni a táhni*), tlačítko ručičky slouži pouze pro odstranění ostatních funkcí, aby nedošlo k nechtěné změně při přesouvání nebo úpravě rozměrů, jedná se také o výchozí nástroj při načtení editoru. Modrá šipka umožňuje otáčení bloků, ale pouze do dvou pozic vzájemně posunutých o 90°. Pokud je vybrán červený křížek, může uživatel odstranit libovolný prvek po kliknutí do jeho ohraničeného prostoru. Políčka pod nadpisy *Rozměry* a *Pozice* zobrazují aktuální hodnoty aktivního prvku, ten se pozná podle červeného orámování. Změnou hodnot v těchto polích může uživatel upravovat umístění a velikost prvku s přesností 5cm (je zvolen poměr 1px = 5cm, menší změna se tedy v editoru neprojeví).

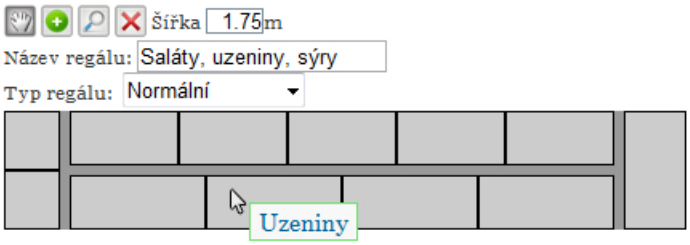


Obrázek 3.3: Panel nástrojů editoru

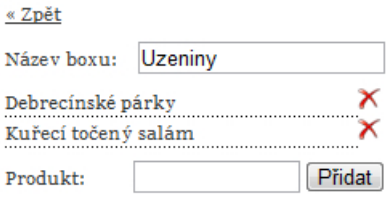
Lupa slouží pro zobrazení detailu regálu, který je znázorněn na obrázku 3.4. Opět je zde podobný panel nástrojů jako na výchozí stránce. Novým nástrojem je křížek v zeleném poli, kterým se po kliknutí do jedné ze čtyř sekcí vytvoří nový box. Ten je možno přesouvat mezi sekcemi nebo měnit pořadí v aktuálně umístěné části. Pokud má box nastaven jméno, je zobrazeno v informativním rámečku (stejný se zobrazí i u pojmenovaného regálu na hlavní stránce editoru). Ve výběru *Typ regálu* lze zvolit druh regálu, který se projeví odlišným podbarvením. Šedé pruhy, rozdělující prvek na 4 části, se dají posouvat a lze tak měnit strukturu regálu. Přesné nastavení lze docílit změnou hodnoty v poli šířka, čímž se změní velikost plochy u naposledy použitého posuvníku.

Opět tlačítkem lupy se uživatel dostane do poslední části editoru a to do detailu boxu zobrazeném na obrázku 3.5. V něm je možno přidávat produkty pomocí jednoduchého formuláře nebo je naopak odebrat pomocí červeného křížku. Tlačítkem *zpět* se uživatel dostane do detailu regálu obsahující editovaný box.

Všechny akce probíhají pouze v rámci JavaScriptu, nedochází tak ke zdlouhavému načítání obsahů jednotlivých prvků a editor je proto velice svižný. Většina JavaScriptového kódu



Obrázek 3.4: Detail regálu



Obrázek 3.5: Detail boxu

se zabývá organizováním a ovládáním funkcí přiřazených k jednotlivým objektům v editoru. Všechny se musí ukládat do proměnné, aby je bylo možno odstranit, pokud uživatel zvolí jiný nástroj. Provedené změny se ihned uloží do jednoho objektu, aby bylo možné editor kdykoliv uložit do databáze.

Použité technologie

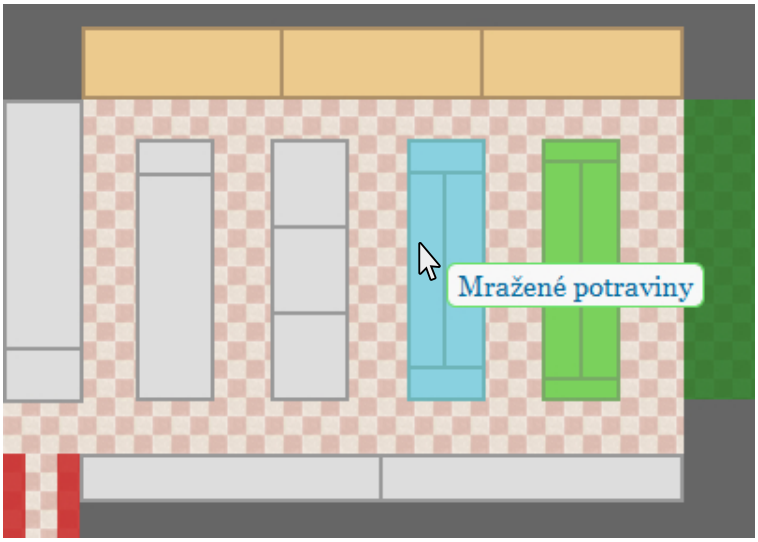
Posouvání objektů je zajištěno funkcí *Draggable* obsažené v balíčku *JQuery UI*. Ve stejné knihovně jsou umístěny metody *Resizable* pro změnu velikosti prvků a *Sortable* pro přesun boxů v rámci regálu. Zobrazení regálu s posuvníky umožnující změnu struktury je implementováno doplňkem *JQuery UI.Layout* mírně upraveným, aby splňoval potřebné požadavky. Zobrazení detailu regálů nebo boxů využívá knihovnu *Colorbox*, která je použita napříč celou aplikací pro zobrazení různých formulářů a dotazníků, kvůli kterým je zbytečné načítat novou stránku. Rámečky s názvy jsou vytvořeny jednoduchou knihovnou *tooltip.JQuery*.

3.4.2 Běžné zobrazení

V běžném zobrazení obchodu na obrázku 3.6 jsou mírné rozdíly oproti zobrazení v editoru. Samozřejmě chybí ovládací prvky, které jsou k dispozici pouze autorovi obchodu. Do regálů jsou doplněny boxy, není proto potřeba otevírat nové okno s detailem regálu. Barevné výplně a ohraničení objektů je zvoleno v méně kontrastních barvách, bílá, použitá na podlahu je nahrazena texturou dlaždic.

3.4.3 Historie změn

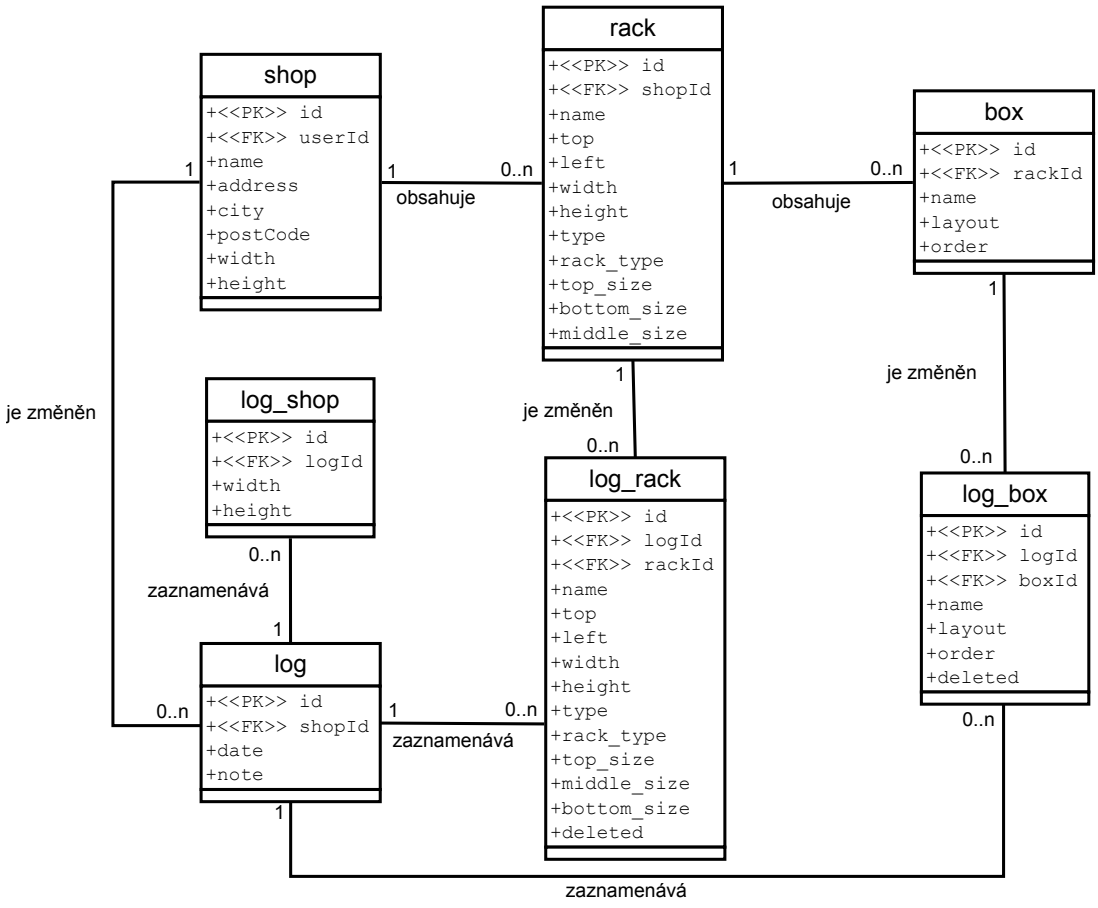
Přesná historie změn je zajištěna čtyřmi novými tabulkami v databázi. Na obrázku 3.7 je znázorněn upravený ER diagram. Jakmile dojde k jakékoliv úpravě obchodu, regálu nebo



Obrázek 3.6: Detail obchodu

boxu, uloží se kopie změny do stejnojmenné tabulky s prefixem *log_*. Aktuální nastavení je ponecháno v originální tabulce, nedochází v ní proto k vytváření zbytečných a nepoužívaných dat, kvůli kterým by se mohl celý systém zpomalit. Pro lepší orientaci v záznamech je možno uložit ke změnám poznámku.

K zobrazení historie slouží nabídka umístěná v horní části editoru. Jednotlivé záznamy se skládají z datumu a poznámky, pokud byla při uložení vyplněna. Stránka se po výběru požadované hodnoty automaticky načte s aktualizovanými objekty, jenž odpovídají zvolené revizi. Pokud uživatel takto obchod uloží, vrátí tím všechny provedené změny, o které ovšem nenávratně nepřijde, poněvadž zůstanou uloženy a může je kdykoliv znovu obnovit.



Obrázek 3.7: Část ER diagramu pro ukládání historie editoru

Kapitola 4

Uživatelské rozhraní

4.1 Specifikace požadavků

Systém je potřeba doplnit o možnost vložení požadovaných položek do nákupního košíku. Kromě základního vyhledávání zajistit přidávání položek přímo ze seznamu produktů umístěných v jednotlivých regálech. Uživatel tím získá přehled o umístění zboží a může si případně projít obchod a zkontrolovat, zda na něco nezapomněl. Může nastat situace, kdy zákazník hledá určitý produkt, ale neví, kde se nachází nebo neví, jak se jmenuje, je nutné proto doplnit vyhledávání o jednoduché filtrování podle názvu regálů nebo boxů. Pro přehlednost ještě na stránku vhodně umístit nákupní košík s možností odstranit nechtěné položky.

Dalším bodem návrhu je historie použitých nákupů, která zákazníkům umožní převzít již jednou vytvořené seznamy a ušetřit tak čas strávený zbytečným vyhledáváním. Pro splnění této funkce je potřeba vytvořit registraci a přihlašování do uživatelských profilů, ve kterých bude historie umístěna. Registrací ovšem nezískají zákaznící možnost vytvořit nebo upravit obchod, tato privilegia mají pouze správci.

4.2 Diagram případu užití

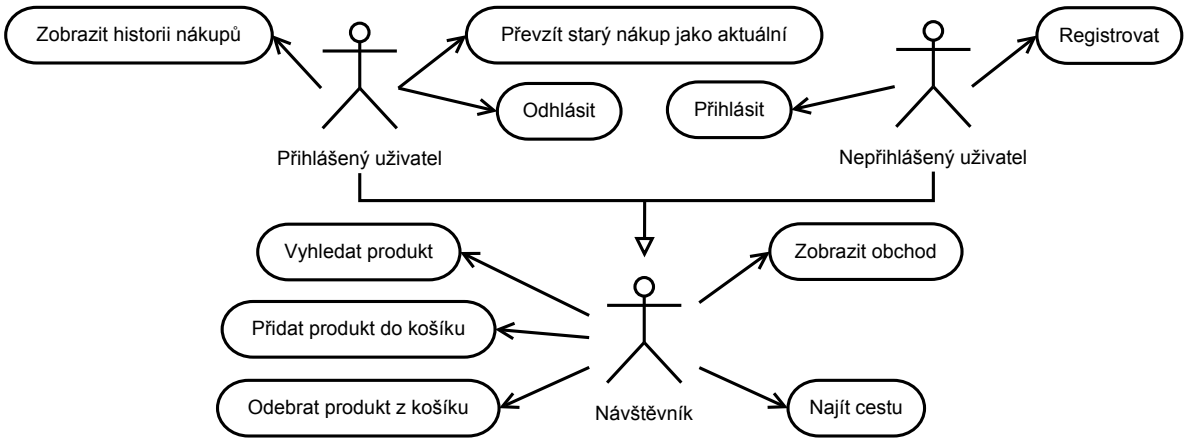
Z diagramu na obrázku 4.1 je patrné, že pro využívání aplikace není nutná registrace, všichni návštěvníci si mohou vytvořit nákup a vygenerovat nejkratší trasu, ale pouze registrovaní uživatelé mají k dispozici historii, ze které mohou převzít již použitá data.

4.3 Návrh databáze

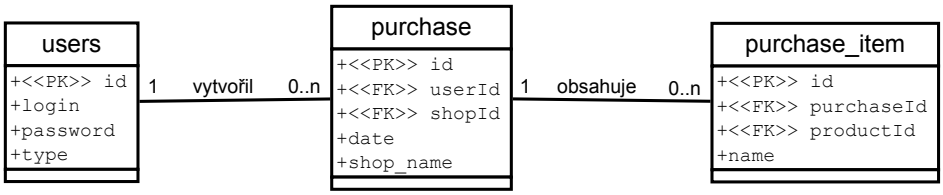
Pro ukládání uživatelů stačí jednoduchá tabulka *users*, obsahující přihlašovací jméno, heslo a uživatelskou skupinu do které patří (běžný uživatel nebo správce). V tabulkách uchovávající informace o nákupu jsou na první pohled nadbytečné hodnoty *shop_name* a *name*, které ale uchovávají názvy pro případ, že by došlo k odstranění originálních položek z databáze. Je tak zajištěno, že uživatel bude mít stále kompletní historii o svých nákupech. Část ER diagramu zobrazující tyto tři tabulky je znázorněna na obrázku 4.2.

4.4 Implementace

Vyhledávání produktů je zobrazeno na obrázku 4.3. V levé horní částí jsou tři výběry pro jednoduché filtrování podle regálů a boxů. Po zvolení produktu v políčku *Zboží* se

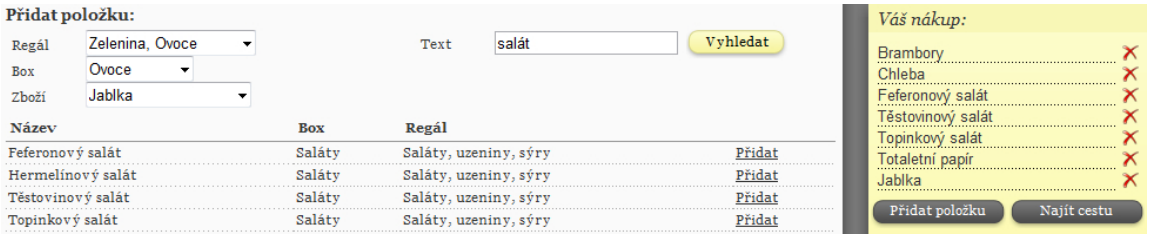


Obrázek 4.1: Část use-case diagramu pro uživatelské rozhraní



Obrázek 4.2: Část ER diagramu pro uživatelské rozhraní

položka automaticky přidá do košíku. V pravé horní části je umístěno pole pro textové vyhledávání, nalezené zboží je zobrazeno pod oběma zmíněnými formuláři s doplňujícími informacemi jako název boxu nebo regálu. Detail obchodu je zobrazen obdobně, jak bylo popsáno v kapitole 3.4.2, novinkou je pouze seznam zboží, který se zobrazí po kliknutí na část regálu.



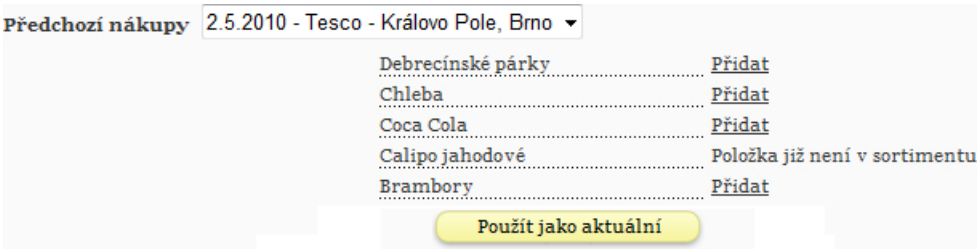
Obrázek 4.3: Vyhledávání produktů

Žlutý box vpravo od vyhledávání obsahuje seznam položek, červeným křížkem se prvek z košíku odstraní. Protože je box umístěn na více stránkách, obsahuje tlačítko *Přidat položku*, které přesměruje stránku na detail obchodu a usnadňuje tak uživateli pohyb po systému. Pokud je seznam prázdný, zobrazí se nejdříve výběr obchodů, v opačném případě se rovnou stránka přesměruje na obchod, ze kterého pochází zboží v košíku. Druhým tlačítkem *Najít cestu* se zobrazí stránka s vygenerovanou nejkratší cestou.

Většina požadavků je zpracována ajaxově (asynchronní komunikace se serverem) a uživatel tak nemusí čekat na načtení stránky. Navíc mu zůstane zobrazená stránka ve stejné pozici a neztratí tak pracně vyhledaná data. Pokud by z jakéhokoliv důvodu nebyl ajaxový požadavek zpracován, veškerá funkčnost zůstane zachována a stránka se klasicky obnoví.

4.4.1 Uživatelský profil

Registrační formulář je dostupný z hlavního menu, jakmile uživatel vyplní svoje přihlašovací údaje a přihlásí se do systému, nahradí registraci v menu historie nákupů. Hlavní část stránky je zobrazena na obrázku 5.1. Tlačítkem *Použít jako aktuální* nahradí stávající položky v košíku zbožím zobrazeným před tlačítkem, pokud tedy produkt nebyl z obchodu odstraněn.



Obrázek 4.4: Historie nákupů

Na serverové části je přihlášení zajištěno vytvořením identity uživatele a uložením do sezení. Na každé stránce, kde je nutno kontrolovat práva pro přístup, lze snadno k těmto údajům přistoupit a vyhodnotit je.

Kapitola 5

Systém ideálního průchodu hypermarketem

K nalezení nejkratší cesty v obchodě je zapotřebí využít více znalostí a algoritmů z oblasti prohledávání stavového prostoru. Konečné vyhledání optimální cesty představuje problém obchodního cestujícího, ten ale vyžaduje na vstupu ohodnocený graf mezi jednotlivými průchozími body. V obchodě je ovšem mnohem více vrcholů, a je tedy nejdříve nutné vytvořit graf právě z těchto bodů a nejkratšími cestami mezi nimi. K tomu poslouží algoritmy rozebrané v kapitole 5.2, řešící nalezení nejkratší cesty mezi dvěma vrcholy.

5.1 Prohledávání stavového prostoru

Stavový prostor je definován množinou stavů a operátorů. Můžeme si ho představit jako orientovaný graf/strom, kde jednotlivé uzly reprezentují stavy a hrany mezi nimi představují operátory, jejímž použitím se dostaneme z jednoho stavu do druhého. Řešením je pak nalezení nejlepší nebo alespoň přijatelné cesty z počátečního stavu do stavu cílového, kde cílových stavů může být i více.[9]

Další znalosti, které má mechanismus prohledávání stavového prostoru k dispozici, jako například v tomto případě délku cesty mezi body, nazýváme heuristikami. Ty různě zefektivňují použitý algoritmus, mohou ho například zrychlit nebo snížit paměťové nároky. Všechny algoritmy, které jsou dále popsány a některé nakonec i použity, obdobné heuristiky využívají a nazývají se proto informované metody.

Princip informovaných metod je založen na tzv. hodnotící funkci, která pro každý uzel určí jeho ohodnocení. Výběr dalšího stavu je řízen touto hodnotou a je zajištěno, že se při správném návrhu hodnotící funkce vybere nejlepší možný prvek. Tím se zabrání prohledávání cest, které nevedou k cíli nebo které nezajišťují nejlepší možný výsledek.[15]

5.2 Hledání nejkratší cesty

Nalezení nejkratší vzdálenosti mezi dvěma body v ohodnoceném grafu patří mezi základní problémy teorie grafů a exsituje pro ní mnoho praktických využití jako například plánovač tras nebo vyhledávání spojů v MHD. Jak již bylo zmíněno v úvodu této kapitoly, v aplikaci se tento algoritmus využije pro sestavení ohodnoceného grafu obsahujícího cesty mezi všemi body, které je nutno obchodem projít.

5.2.1 Dijkstrův algoritmus

Algoritmus, jak už napovídá jeho název, je dílem nizozemského informatika Edsgera Dijkstry. Lze si ho představit jako zjednodušené procházení grafu do šířky s kladně ohodnocenými hranami. Pracuje v tzv. vlnách, v nichž vždy vybere nejbližší bod od počátečního vrcholu a zpracuje vzdálenosti u jeho sousedů. Zpracovávají se tak vždy pouze uzly, ke kterým již byla nalezena nejkratší cesta. Jestliže není potřeba nalézt nejkratší vzdálenosti ke všem vrcholům grafu, lze proces ukončit v okamžiku, kdy algoritmus projde všemi požadovanými body a zamezit tak zbytečným výpočtům, které ve výsledku stejně neposkytnou nové údaje.

Časová složitost závisí na použité prioritní frontě, pro pole je $O(V^2)$, pro haldu je $O((V+H)logV)$ a pro Fibonacciho haldu je $O(H+VlogV)$, kde V je počet vrcholů a H počet hran. Jelikož v aplikaci není použito záporné ohodnocení hran, pro které je navržena pomalejší metoda *Bellman-Ford*, a ani není potřeba vyhledat nejkratší vzdálenosti mezi všemi vrcholy, jako dělá pomalejší *Floyd-Warshall*, zvolil jsem tuto metodu pro implementaci. [16]

5.2.2 Bellmanův-Fordův algoritmus

Tento algoritmus umožňuje nalézt nejkratší cestu mezi dvěma body i pro záporně ohodnocené hrany a tvoří tak alternativu rychlejšího Dijkstrova algoritmu, který ovšem vyžaduje pouze kladně ohodnocené hrany. Princip je podobný jako u již zmíněného rychlejšího kolegy, rozdíl je pouze v tom, že v důsledku možného výskytu záporně ohodnocených hran je nutno graf projít vícekrát. V nejhorším případě se graf projde H-1 krát a časová složitost je $O(V*H)$, kde V je počet vrcholů a H je počet hran. Proces lze urychlit kontrolováním vzdáleností mezi jednotlivými iteracemi a pokud nedojde k žádné změně, je možno algoritmus ukončit.

Problém může nastat, jestliže graf obsahuje záporný cyklus, který by neustále vylepšoval některé cesty a výsledné vzdálenosti by nebyly korektní. Proto se po ukončení algoritmu provede ještě jedna iterace a pokud nastane změna, je jasné, že v grafu je záporný cyklus a nelze v něm nalézt nejkratší cestu. [10]

5.2.3 Floydův-Warshallův algoritmus

Výhodou Floyd-Warshallova algoritmu je, že nalezne nejkratší cesty mezi všemi dvojicemi vrcholů a je tak rychlejší než x-krát zavolaný Dijkstrův algoritmus a je také mnohem jednodušší na implementaci. Pracuje pouze s kladně ohodnocenými hranami a pokud existuje více cest se stejnou délkou, vybere tu s nejméně přechody.

Algoritmus postupně prochází graf a hledá nejkratší cestu mezi vrcholy, ale sestavenou pouze z bodů odpovídajících současné iteraci. Pokud se proces nachází například v 5. iteraci, hledá cesty sestavené pouze z bodů 1 až 5. Postupně tedy projde celý graf a aktualizuje doposud nalezené hodnoty. Časová složitost je $O(V^3)$, kde V představuje počet vrcholů.[10]

5.3 Problém obchodního cestujícího

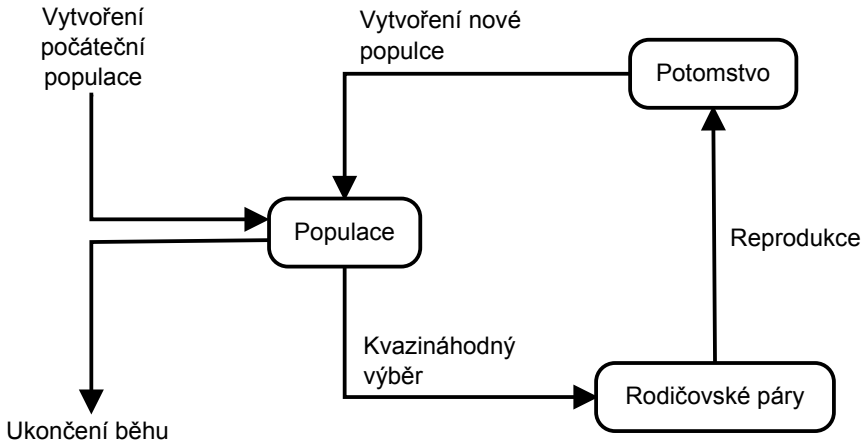
Problém obchodního cestujícího patří mezi nejznámější a nejpopulárnější optimalizační problémy a řadí se mezi NP-těžké. To znamená, že neexistuje algoritmus, který by dokázal nalézt optimální řešení v polynomiálně omezeném čase. Existuje proto množství přibližných metod, které nezaručují vždy nejlepší výsledek, ale poskytují z hlediska časové náročnosti využitelné řešení, jenž se tomu optimálnímu velmi blíží ne-li vyrovná. I když je tento problém známý přes 250 let, stále je aktivně zkoumán a hledá se co nejeefektivnější způsob jeho

řešení. Uplatnění najde například v logistice, plánování, krystalografii a mnohých dalších oborech.

Základní definice problému obchodního cestujícího zní, že obchodní cestující musí navštívit všechna zadaná města právě jedenkrát a vrátit se do výchozího bodu. To vše musí provést s co nejmenší vzdáleností celkové trasy. V našem případě se jedná o menší modifikaci tohoto problému. Cestující musí začít ve startovním bodě a projít všechna ostatní místa až do bodu cílového, může ovšem projít stejnými trasami několikrát, hlavní požadavky se kladou na co nejkratší vzdálenost.[8]

5.3.1 Genetické algoritmy

Genetické algoritmy jsou optimalizační metody, inspirované Darwinovým principem evoluce. Vychází z principu, že v životě přežívají silnější, schopnější nebo lépe přizpůsobiví jedinci. Ti také mají větší šanci na reprodukci a tím i předání svých silných vlastností do dalších generací, které se navíc mohou zkombinovat s vlastnostmi druhého rodiče a vytvořit tak ještě schopnějšího jedince než byli oni. Tímto způsobem se zvyšuje celková genetická výbava populace neboli z hlediska algoritmů se nalézají stále kvalitnější řešení problému.



Obrázek 5.1: Schématické znázornění funkce genetického algoritmu

Výběr jedinců vhodných k reprodukci se provádí formou soutěže. Silnější jedinci se vybírají podle jejich kvality či ohodnocení (neboli jejich *fitness*) a zvyšuje se tak možnost vytvoření silnější populace. K vytvoření nové generace se provádí rozmanité techniky křížení a mutace, které se provádějí s různou pravděpodobností úspěchu (mutace většinou v rozmezí 0,1–5% a křížení s pravděpodobností 75–95%). Jistou nevýhodou celého algoritmu je značná míra náhody, která figuruje při těchto operacích. Může se tak stát, že nejlepší řešení bude nalezeno hned ze začátku nebo také celá populace může zdegenerovat a oddálit tak nalezení optimálního výsledku. Z tohoto důvodu nelze zaručit garantovanou kvalitu výsledků a spíše se uvádí průměrná úspěšnost podle delšího pozorování.

Velkou výhodou a usnadněním genetických algoritmů určených pro řešení problému obchodního cestujícího, je snadné zjištění ohodnocení jedince představující celkovou délku trasy. Pro jiné použití se generuje *fitness* často velmi složitě a zabírá tak velkou měrou výpočetní výkon, který pak nelze využít pro generování populací a výsledek nemusí dosahovat požadovaných kvalit.[8]

Selekční mechanizmus

Vzhledem k snadnému zjištění fitness je první závažnější problém vybírání jedinců pro reprodukci. Jelikož genetické algoritmy představují reálný život, musí tak být vyřešen i proces vybírání silnějších kandidátů pro křížení. Nelze ovšem použít jenom nejlepší jedince a ty vzájemně křížit, poněvadž by brzy došlo ke ztrátě rozmanitosti a populace by mohla předčasně konvergovat k horšímu řešení.

Jednou z nejčastěji využívaných řešení je ruletový mechanismus, kde na rozdíl od klasické rulety nemají všechna čísla stejnou pravděpodobnost, ale jsou upřednostňováni kvalitnější jedinci. Další možností je metoda vybírání podle pořadí, kdy se všichni jedinci seřadí podle ohodnocení a podle něho se vybírají pro reprodukci. Poslední metodou je turnajový výběr, kde se náhodně vyberou dva jedinci a ten lepší je vybrán pro reprodukci. Jelikož nelze určit, že by jedna z metod byla lepší než druhá, zvolil jsem z důvodu snadné implementace turnajový způsob selekce.[8]

Operátor křížení

Nejdůležitější prvek genetických algoritmů je operátor křížení, kterým se vytváří většina nové populace. Vždy se jedná o vzájemnou výměnu určité části rodičů, pokud ale není postavena na důkladnějším prozkoumání problému a využívá pouze náhodného výběru, nedosahuje algoritmus dobrých výsledků.

Pro problém obchodního cestujícího dosahuje naopak velmi dobrých výsledků operátor křížení s rekombinací hran *ERX* (*Edge Recombination Crossover*) a je považován za jeden z nejúspěšnějších. Zajímavé na operátoru je, že ze dvou rodičů vzniká pouze jeden potomek, kde každý úsek pochází z jednoho z rodičů. Základem je tzv. hranová tabulka, která každému bodu připisuje seznam jeho sousedů v obou rodičích. Postupně se sestavuje nový potomek z bodů s nejméně sousedy, který se také odstraní z hranové tabulky a všech jejích záznamů. Operátor lze ještě dále zdokonalit, pokud totiž nastane situace, že je v hranové tabulce více bodů se stejným počtem sousedů vybere se ten, který je v ní obsažen opakovaně.[8]

Operátor mutace

Mutace i s velmi malou pravděpodobností provedení představuje důležitou část algoritmu. Zabraňuje příliš rychlému zjednotvárnění vlastností, ztrátě potenciálně užitečných úseků a předčasné konvergenci populace. Vzhledem ke specifickému použití je vybrání způsobu mutace snadnou záležitostí, kdy v podstatě jediným možným řešením je vybrání dvou náhodných bodů a prohozením jejich pozice.[8]

5.3.2 Simulované žíhání

Další oblíbenou metodou pro řešení problému obchodního cestujícího je metoda simulovaného žíhání, která naopak od genetických algoritmů má základ ve fyzice a vychází z kontrolovaného ochlazování oceli nebo jiného tuhého tělesa, při kterém se odstraňují defekty materiálu. Nejdříve se těleso ohřeje na vysokou teplotu, čímž dojde k náhodnému uspořádání molekul a postupným ochlazováním se částice dostávají do rovnovážné polohy. Ochlazování by nemělo být moc rychlé, aby defekty tzv. nezamrzly v materiálu.

Z jednoho stavu do dalšího se systém dostane použitím tzv. operátora poruchy. Provede se změna v struktuře a poté pomocí *Metropolisova kritéria*, které definuje pravděpodobnost přijetí změny se rozhodne, zda nový stav použít. Pro vyšší teploty je pravděpodobnost

velmi vysoká a přijímá se tak téměř každá změna. Jak se snižuje teplota, snižuje se i pravděpodobnost přijetí a systém se stabilizuje.[12][3][23]

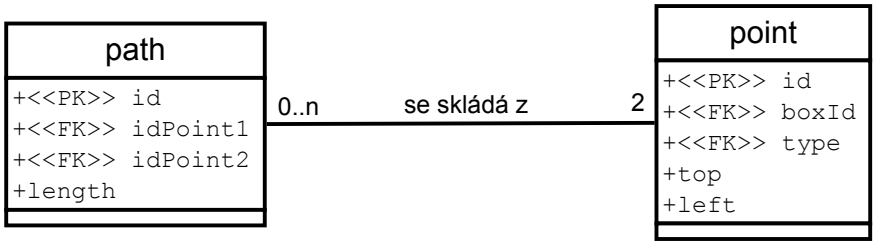
5.4 Návrh

5.4.1 Specifikace požadavků

Při ukládání obchodu vytvořit síť vrcholů a cest mezi nimi pro pohyb v hypermarketu. Cesty nesmí procházet skrz jiné objekty a body musí být k dispozici pro každý box, ve kterém je umístěno zboží. Algoritmem pro nalezení nejkratší vzdálenosti mezi body vytvořit ohodnocený graf cest mezi boxy podle aktuálního nákupu. Nakonec vhodně vykreslit nalezené řešení problému obchodního cestujícího.

5.4.2 Návrh databáze

Na obrázku 5.2 je znázorněna část ER diagramu pro uložení bodů a cest. Cesta se skládá ze dvou bodů a je doplněna její délkou. V tabulce *point* se zaznamenává pozice bodu v obchodě a přiřazení k regálu nebo jinému objektu.



Obrázek 5.2: Část ER diagramu pro ohodnocený graf

5.5 Implementace

5.5.1 Generování bodů a cest

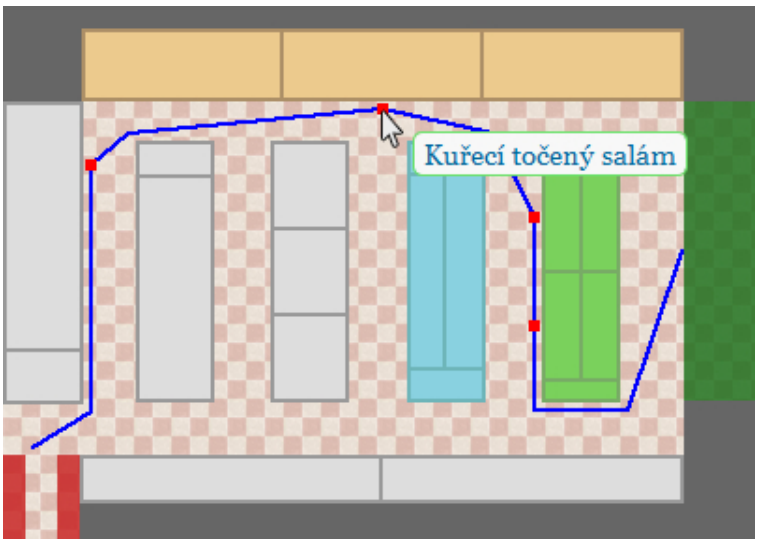
Generování bodů a cest bylo nejdříve zamýšleno provést po uložení obchodu. Jelikož je ale potřeba kontrolovat, zda cesty neprochází nějakými objekty, je operace časově velmi náročná a ukládání obchodu by se tak stalo zdlouhavé a málo využívané. Uživatel by tak svojí rozpracovanou práci ukládal velmi zřídka a vzrůstala by možnost ztráty dat. Proto byla tato část oddělena a je nutno ji vyvolat zvlášť tlačítkem *Vygenerovat cesty*.

Body jsou vytvářeny 5px od každého boxu a 5px od každého rohu regálu, zdi či pokladny. Následně jsou odstraněny ty, které leží mimo plochu obchodu nebo jsou umístěny uvnitř jiného objektu. Proces pokračuje vytvořením cest mezi všemi body. Zde nastává časově nejnáročnější operace, kdy se musí projít všechny cesty a hrany objektů a zkontrolovat, zda nedochází k jejich konfliktu, aby nenastala situace, že systém například vygeneruje cestu skrz zeď.

5.5.2 Vyhledávání

Všechny metody pro vyhledání nejkratší cesty jsou umístěny v souboru a stejně pojmenované třídě *ShopPresenetr*. Nejdříve je získán výchozí a cílový bod, pokud existuje více možností, vybere se ten, který je spojen nejvíce cestami. K nim se přidají další vrcholy odpovídající boxům se zbožím podle nákupního košíku a vygeneruje se graf mezi nimi pomocí dijkstrova algoritmu. Pro jednotlivé hrany jsou uloženy i dílčí úseky, z nichž se skládají, aby bylo možno ve výsledku správně vykreslit celou cestu.

S tímto grafem je možno začít řešit problém obchodního cestujícího. Nejdříve je vygenerována počáteční generace obsahující 100 jedinců. První cesta je vytvořena podle vzdálenosti bodů od počátku, je tím zajištěn alespoň relativně silný jedinec v první populaci, která je dále generována náhodně. Následuje cyklus genetického algoritmu, kdy nová generace je složena z 20 nejsilnějších jedinců předcházející populace a zbylých 80 je vytvořeno z 95% křížením operátorem *ERX* a 5% mutací. Počet iterací je určen množstvím bodů a je omezen časem 20 sekund, poté je zobrazen nejlepší doposud nalezený výsledek.



Obrázek 5.3: Detail obchodu s nalezenou cestou

Na obrázku 5.3 je zobrazen obchod s nalezenou cestou. Červené body znázorňují zastávky pro vložení zboží do košíku. Po najetí myši se zobrazí informační box se seznamem požadovaného zboží v tomto regálu. Modré cesty jsou zajištěny Javascriptovou knihovnou *WZ_JSGraphics*.

Kapitola 6

Závěr

Cílém této práce bylo navrhnout a implementovat systém schopný najít ideální průchod hypermarketem podle zadaného nákupu s možností vytvoření a editace těchto obchodů. Systém byl úspěšně analyzován, navržen a implementován a dosahuje poměrně dobrých výsledků. V důsledku jisté míry náhody při generování cesty není možno zaručit vždy správný výsledek, nicméně je zajištěno přijatelné řešení, které se od toho optimálního příliš neliší. Výpočty jsou časově náročné, jak pro vyhledání cesty, tak pro generování průchozích bodů a s roustoucí velikostí obchodu a nákupu roste i tento čas. Alespoň pro menší obchody je systém svižný a běžný uživatel nezaznamená časovou prodlevu.V zájmu zvýšení komfortu při práci s aplikací je komunikace mezi serverovou částí a uživatelským rozhraním v prohlížeči postavena z velké části na technologii *AJAX*.

Mezi nejnáročnější části celé aplikace z hlediska vývoje byl editor, který navzdory velkému počtu použitých knihoven vyžadoval mnohé úpravy, aby byla dosažena požadovaná funkčnost. Bylo nutné vymyslet systém pro ukládání dat schopný komunikovat mezi všemi částmi a serverovou stranou, která zajišťuje ukládání do databáze. Některé použité knihovny nebylo možné nastavit pro zajištění správné funkce a prošly tak drobnými změnami.

I když *JavaScript* a jeho knihovny prošly rozsáhlým vývojem, neposkytujípro tvorbu uživatelského rozhraní takové možnosti jako běžné knihovny pro desktopové aplikace. Editor je z těchto důvodů v určitých fázích nedokonalý. Intuitivně uživatel při práci sahá ke klávesovým zkratkám typu *Delete* nebo *Ctrl+C*, které bohužel nefungují a které by zaručeně zvýšily uživatelský komfort. Z opačného pohledu je ale implementováno mnoho funkcí, které by před několika lety byly naprosto nepředstavitelné a vytváří tak použitelný a efektní prostředek pro správu obchodů.

Pro implementaci vyhledávaní mi byla nápomocná knížka [8], ve které jsou všechny znalosti pro tvorbu genetických algoritmů názorně a podrobně vysvětleny s jasným zhodnocením kvalit jednotlivých metod. Všechny kódy pro tuto část jsou mým dílem, ale vycházejí téměř výhradně z této publikace.

Teoretické části této práce mi rozšířily znalosti o technologiích pro www aplikace a umožnili tak důkladně prozkoumat výhody či nedostatky mnou preferovaných prostředků. Dokázal jsem si vytvořit vlastní pohled na věc a zhodnotit zda opravdu využívám ty nejlepší dostupné nástroje. Znalosti z oboru procházení stavového prostoru od genetických algoritmů po nalezení nejkratších cest v grafu jsou velmi přínosné pro mojí budoucí práci a jednoznačně v ní najdou uplatnění.

Aplikace se dá využít i k jiným účelům. Po drobnější úpravě lze například vytvořit systém pro zahrádkářské organizace, kde si mohou rozvrhnou záhony a rostliny v nich. Místo procházení obchodem tak může být nalezení nejeefektivnější cesty při sklizni. Díky

objektovému přístupu lze použít jen určité metody pro procházení stavového prostoru, které jsou využitelné v mnoha oborech lidské činnosti.

Drobné mezery jsou jak již bylo zmíněno v editoru, kde je možno vylepšit nebo vytvořit nové funkce a zvýšit tak uživatelský komfort. Například není umožněno zakreslení kulatých nebo šikmých regálů. Další nedostatek je při rozhodování mezi více použitelnými body jako například u pokladen. Nyní se vybere ten, který je spojen s nejvíce vrcholy a může tak nastat situace, kdy cesta povede naprosto zbytečně k nejvzdálenější kase. Zboží může být také umístěno ve více regálech a podle aktuální cesty by se měl vybrat nejvýhodnější z nich.

Literatura

[1] Dařena, F.: *Myslíme v jazyku Perl.* Grada Publishing, a.s., 2005, iSBN 80-247-1147-8.

[2] Feuerstein, S.; Pribyl, B.: *Oracle PL/SQL programming.* O'Reilly Media, Inc., 2005, iSBN 978-0-596-00977-9.

[3] Grosan, C.; Abraham, A.; Ishibuchi, H.: *Hybrid Evolutionary Algorithms.* Springer-Verlag Berlin Heidelberg, 2007, iSBN 978-3-540-73296-9.

[4] Grudl, D.: Database Abstraction Library for PHP 5 [online]. `http://dibiphp.com/cs/`.

[5] Grudl, D.: O frameworku Nette [online]. `http://nettephp.com/cs/o-frameworku`.

[6] Gutmans, A.; Bakken, S. S.; Rethans, D.: *PHP 5 Power Programming.* Pearson Education, Inc., 2005, iSBN 0-131-47149-X.

[7] Hall, M.: *Core Servlets and JavaServer Pages (JSP).* Prentice Hall, Inc., 2000, iSBN 80-86330-06-0.

[8] Hynek, J.: *Genetické algoritmy a genetické programování.* Grada Publishing, a.s., 2008, iSBN 978-80-247-2695-3.

[9] doc. Ing. František Zbořil, C.; Ing. František Zbořil, P.: Základy umělé inteligence IZU studijní opora. 2006.

[10] Jakub Černý, P.: Základní grafové algoritmy [online]. `http://kam.mff.cuni.cz/~kuba/ka/min_cesta.pdf`, 2008.

[11] Keogh, J.: *Java Demystified.* The McGraw-Hill Companies, Inc., 2005, iSBN 80-251-0839-2.

[12] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* Slovenská technická univerzita v Bratislavě, 2000, iSBN 80-227-1377-5.

[13] Lacko, L.: *SQL Kapesní přehled.* CP Books, a.s., 2005, iSBN 80-251-0788-4.

[14] Lee, W.-M.: What Is ASP.NET [online]. `http://ondotnet.com/pub/a/dotnet/2005/09/19/what-is-asp-net.html`.

[15] Mařík, V.; Štěpánková, O.; Lažanský, J.; aj.: *Umělá inteligence (1).* Academia, nakladatelství AV ČR, 1993, iSBN 80-200-0496-3.

[16] Mička, P.: Dijkstrův algoritmus [online]. `http://www.algoritmy.net/article/5108/Dijkstruv-algoritmus`.

[17] Novotný, L.: Historie a vývoj jazyka Java [online].
http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xnovotn8.htm.

[18] stránky, W.: About MySQL [online]. http://www.mysql.com/about/.

[19] stránky, W.: PHP Function List [online]. http://cz.php.net/manual-lookup.php.

[20] Stěhule, P.: PostgreSQL [online]. http://www.postgres.cz/index.php/PostgreSQL.

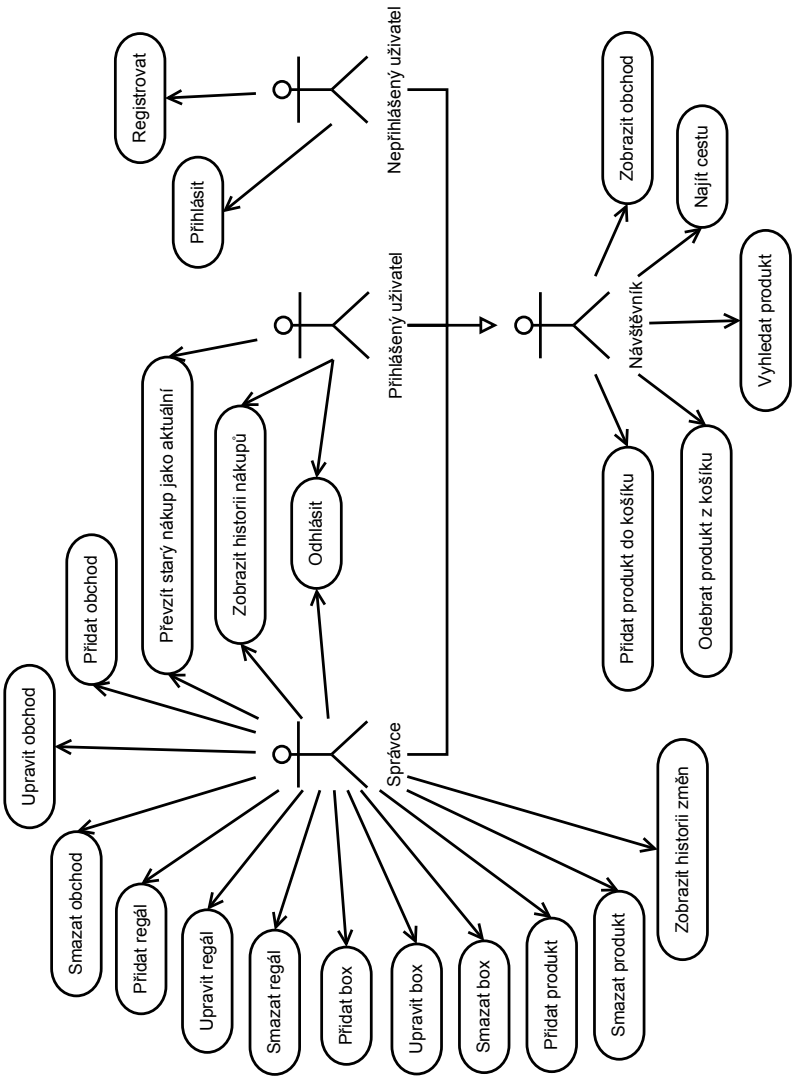
[21] Suehring, S.: *MySQL Bible*. Wiley Publishing, Inc., 2002, iSBN 0-7645-4932-4.

[22] Tim Converse, C. M., Joyce Park: *PHP5 and MySQL Bible*. Wiley Publishing, Inc., 2004, iSBN 0-7645-5746-7.

[23] Vašíček, Z.: Simulované žíhání [online].
http://feldocuments.googlecode.com/svn-history/r127/trunk/NS/msi.pdf.

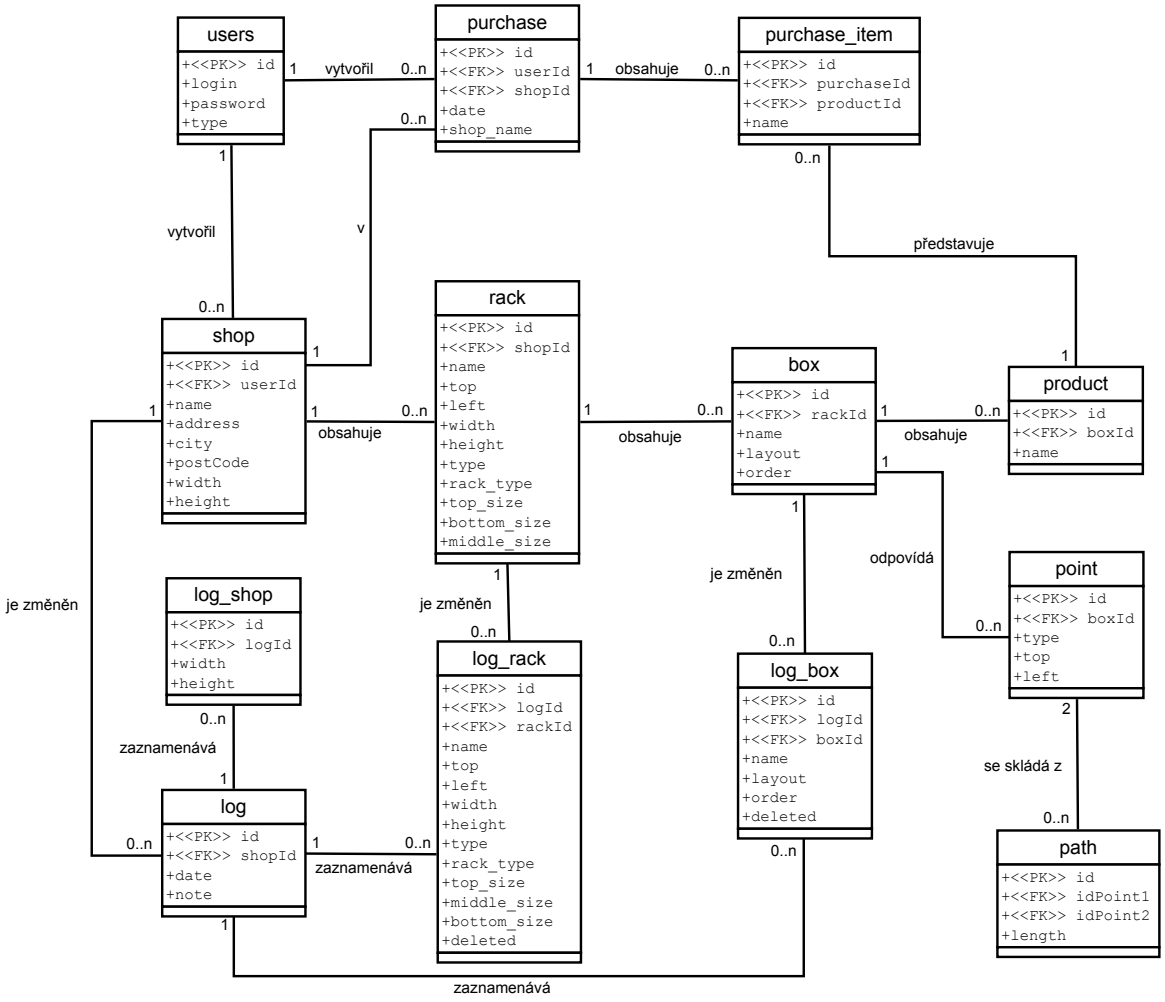
Příloha A

Kompletní Use-case diagram



Příloha B

Kompletní ER diagram



Příloha C

Obsah CD

- adresář `poradce` – Zdrojové soubory aplikace, pro spuštění nahrát obsah této složky do kořenového adresáře. Adresáře `/poradce/app/temp` a `/poradce/app/sessions` musí být zapisovatelné. V souboru `/poradce/app/config.ini` nastavit přístupové údaje k databázy *MySQL*.
- adresář `phpdoc` – Programová dokumentace
- adresář `tex` – Zdrojové soubory dokumentace
- `xkocou00.pdf` – Tato dokumentace v elektronické podobě
- `db.sql` – Export *MySQL* databáze