VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO EVIDENCI A TISK ZÁSILEK

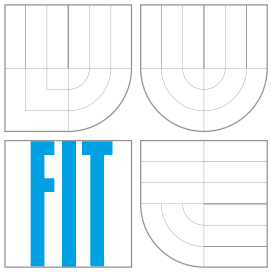
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK PETŘÍK

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO EVIDENCI A TISK ZÁSILEK

DELIVERY INFORMATION SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK PETŘÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. IVANA RUDOLFOVÁ
SUPERVISOR

BRNO 2008

Abstrakt

Cílem práce bylo vytvoření webové aplikace usnadňující dopravcům a přepravcům práci se zásilkami. Účelem aplikace je evidence zásilek a tisk etiket. Nejprve je poskytnut přehled technik používaných při vývoji webových aplikací. Dále je popsáno, jak funguje moderní zásilková přeprava. Pak je specifikována připravovaná webová aplikace. Na závěr je analyzován průběh implementace v C# ASP .NET a výsledky práce.

Klíčová slova

Tvorba webových aplikací, porovnání, PHP, Java, C#, FastCGI, AJAX, přeprava zásilek, tisk etiket.

Abstract

The field of this work was to create a web application easing shipper's and carrier's work with consignments. The purpouse of the application is evidence of packages combined with label printing. A prewiew of present-day web application development techniques is provided first. Recent package delivery systems are discussed then. The application specification is the next topic. Implementation process using C# ASP .NET and the results of this work is analyzed in the last chapters.

Keywords

Web application development, comparison, PHP, Java, C#, FastCGI, AJAX, package delivery, label printing.

Citace

Radek Petřík: Informační systém pro evidenci a tisk zásilek, diplomová práce, Brno, FIT VUT v Brně, 2008

Informační systém pro evidenci a tisk zásilek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Ivany Rudolfové. Informace z praxe o fungování firem poskytujících přepravu zásilek mi poskytl Bc. Martin Gürtler. Informace z praxe o technologii SEO mi poskytl Bc. Zdeněk Filipec. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Petřík
19.5.2008

Poděkování

Vedoucí práce Ing. Ivaně Rudolfové bych chtěl poděkovat především za trpělivost a rady, které mi dala. Určitě musím poděkovat Bc. Martinu Gürtlerovi za vyčerpávající popis fungování přepravy zásilek, bez něhož bych nemohl specifikovat aplikaci tak, aby byla pro přepravce a dopravce užitečná. Díky patří také Bc. Zdeňkovi Filipcovi, který mi poskytl informace o SEO z praxe, čímž mi ušetřil několik hodin práce. V neposlední řadě bych chtěl poděkovat rodině, která mi byla při psaní oporou a podržela mě, když jsem nevěděl, jak pokračovat.

Obsah

1 Úvod **4**

2 Tvorba webových aplikací **5**
2.1 Technologie používané na webu 5
2.1.1 Přenos informací z prohlížeče na server 5
2.1.2 JavaScript 6
2.1.3 Document Object Model 6
2.1.4 Rámce 6
2.1.5 Technika neviditelných rámců 7
2.1.6 AJAX 7
2.1.7 SEO 8
2.2 Platformy pro implementaci webových aplikací 8
2.2.1 Common Gateway Interface 9
2.2.2 Platformy Java 2 a .NET Framework 9
2.2.3 PHP 14
2.2.4 Ruby 17
2.2.5 FastCGI 17
2.2.6 Závěrečné hodnocení 18
2.3 Webové frameworky 18
2.3.1 Vlastnosti webových frameworků 18
2.3.2 Symfony 19
2.3.3 Zend framework 19
2.3.4 Ruby on Rails 19
2.3.5 MonoRail 20
2.3.6 Struts 20
2.3.7 JSF 20
2.3.8 ASP .NET 20
2.3.9 Hodnocení 20

3 Zásilková přeprava **21**
3.1 Před předáním zásilky dopravci 21
3.2 Předání zásilky dopravci 21
3.3 Doručení zásilky do depa dopravce 22
3.4 Přeprava zásilky do výstupního depa 22
3.5 Doručení zásilky příjemci 22

4 Specifikace webové aplikace pro evidenci a tisk zásilek **23**
4.1 Neformální specifikace . . . 23
4.1.1 Obecné zásady a součásti aplikace . . . 23
4.1.2 Uživatelé a přihlašování do systému . . . 24
4.1.3 Evidence zákazníků . . . 24
4.1.4 Evidence zásilek . . . 25
4.1.5 Import zásilek a zákazníků . . . 26
4.1.6 Tisk etiket . . . 26
4.1.7 Modul ukončení dne . . . 26
4.1.8 Firemní nastavení . . . 27
4.2 Diagramy případů použití . . . 28
4.2.1 Administrační část aplikace . . . 28
4.2.2 Uživatelská část aplikace . . . 28
4.3 Diagram entit a vztahů . . . 32
4.3.1 Vlastnosti používané ve více entitách . . . 32
4.3.2 Entita FIRMS . . . 32
4.3.3 Entita POVOLENE_MENY . . . 32
4.3.4 Entita NASTAVENI . . . 32
4.3.5 Entita ZASILKY . . . 33
4.3.6 Entita CISELNE_RADY . . . 33
4.3.7 Entita CISELNIKY_META . . . 34
4.3.8 Entita MAPOVANI_CISELNIKU a s ní související . . . 34
4.3.9 Entity IMPORT_ZASILEK a IMPORT_ZAKAZNIKU . . . 34
4.3.10 Entity určující typ importu . . . 35
4.3.11 Číselníky . . . 36

5 Implementace a hodnocení použitých nástrojů **37**
5.1 Knihovna libtools . . . 37
5.2 Autentizace a autorizace . . . 38
5.3 Tvorba uživatelského rozhraní . . . 39
5.3.1 DynamicTable . . . 39
5.3.2 AJAX . . . 40
5.4 Hodnocení ASP podle nabytých zkušeností . . . 40
5.5 Zdrojový kód a licence . . . 40

6 Závěr **41**

A Uživatelský manuál **45**
A.1 Úvod . . . 45
A.1.1 Obecné informace o aplikaci . . . 45
A.1.2 Informace o nápovědě . . . 45
A.2 Přehled funkcí aplikace . . . 45
A.2.1 Databáze zákazníků . . . 45
A.2.2 Databáze zásilek . . . 45
A.2.3 Vstup dat . . . 46
A.2.4 Importování dat . . . 46
A.3 Nápověda k formulářům . . . 49
A.3.1 Administrační část aplikace . . . 49

A.3.2 Zákazník 50
A.3.3 Zásilka 51
A.3.4 Nastavení 52
A.3.5 Ukončení dne 56

B Diagram entit a vztahů **58**

C CD se zdrojovým kódem **60**

Kapitola 1

Úvod

Dnes už určitě nikdo není schopen říct, kdy bylo poprvé přepraveno nějaké zboží z místa na místo za účelem zisku respektive směny. Dá se říct, že lidé přepravovali zboží vždy. Dlouho se však jednalo o individuální přepravu, která nebyla nijak organizovaná. Mezi první organizované poštovní služby, které však byly neveřejné, patří čínská (počátky v 16.-11. stol. př. n. l.), perská (550 - 333 př. n. l.) a římská (založena 27 př. n. l.). Základní principy se od té doby téměř nezměnily. Pošta a balíky se zbožím jsou stále přepravovány po předem připravených trasách mezi jednotlivými poštami a depy dopravce. Zvyšuje se však rychlost a kvalita přepravy. Ve zvyšování kvality služeb pomáhají společnostem poskytujícím zásilkovou přepravu informační systémy. Dopravcům umožňují automatické třídění zásilek podle adresy a také sledování, kde se zásilka právě nachází (služba sledování zásilek je často zpřístupněna i odesilateli a příjemci zásilky prostřednictvím internetu). Podrobněji tuto tematiku popíšu ve třetí kapitole. Informace o historii poštovní služby jsem čerpal z [2] a [21].

Obrovské možnosti, které dopravním společnostem přinášejí informační systémy však něco stojí. Je potřeba do nich vložit informace o každé zásilce, často včetně adresy příjemce a odesilatele, váhy zásilky, případně i její přibližné rozměry a podobně. Nejen s touto úlohou má dopravcům i jejich zákazníkům pomoci webová aplikace, jejíž tvorbou se zabývá tato práce. Specifikaci toho, co by měla aplikace umožňovat a její návrh uvedu ve čtvrté kapitole. Pátá kapitola se pak bude zabývat zajímavými částmi implementace.

Nejprve se však krátce zmíním o historii World Wide Web a technikách používaných při tvorbě webových stránek. A pak se budu hlouběji zabývat nejnovější trendy ve vývoji webových aplikací, které nyní zastřešují zkratky AJAX (Asynchronous JavaScript and XML) a SEO (Search Engine Optimization). Pokusím se také o co nejobjektivnější zhodnocení nejpoužívanějších jazyků a webových frameworků pro tvorbu webových aplikací.

Tato práce navazuje na stejnojmenný semestrální projekt. Rozšířil jsem v ní druhou kapitolu, upravil třetí kapitolu a rozšířil a upravil čtvrtou kapitolu podle nově nabytých vědomostí.

Kapitola 2

Tvorba webových aplikací

World Wide Web vynalezl v roce 1989 Tim Berners-Lee a v roce 1990 vytvořil první webový server a webového klienta (prohlížeč), který nazval WorldWideWeb a později přejmenoval na Nexus. Nexus sloužil zároveň jako editor a umožňoval jak tvorbu tak editaci a zobrazování dokumentů napsaných v HTML (Hypertext Markup Language). První verze HTML umožňovaly pouze vytváření dokumentů s formátovaným textem, vloženými obrázky a hypertextovými odkazy na jiné HTML dokumenty a zdroje identifikované pomocí jedinečných identifikátorů (URL). V jednoduchosti je síla a proto se také WWW začalo velmi rychle rozrůstat a s ním rostl i Internet, médium umožňující přístup k informacím dostupným na WWW většinou prostřednictvím HTTP (Hypertext Transfer Protocol). Časem vznikaly nové verze HTTP, HTML a samozřejmě i serverů a prohlížečů. Také vznikaly nástroje na tvorbu webových stránek, takzvané webové frameworky. V nestavovém HTTP protokolu se začala přenášet informace o stavu prostřednictvím POST metody, query string GET metody a cookies. Bylo umožněno vkládání krátkých programů v jazyce JavaScript do HTML dokumentů, byl vytvořen DOM (Dokument Object Model) a CSS (Cascading Style Sheets). Tím vznikly dynamické webové stránky, které měnily obsah a vzhled i bez komunikace se serverem. Díky zmíněným vylepšením mohly vznikat webové informační systémy a webové aplikace, na jejichž používání stačí mít v počítači webový prohlížeč a jejichž tvůrce či provozovatel se nemusí starat o to, aby všem uživatelům nahrál vylepšenou nebo opravenou verzi aplikace. Stačí změnit aplikaci na webovém serveru a projeví se to u všech uživatelů. Když k tomu ještě přidáme využívání neviditelných rámců, neviditelných plovoucích rámců a objekt XMLHttpRequest v JavaScriptu, dostáváme se k termínu AJAX (Asynchronous JavaScript and XML). Jedná se o souhrný název pro několik technologií, které se snaží o co největší interaktivnost stránek a posílání co nejmenšího množství dat mezi prohlížečem a serverem. Informace o historii World Wide Web jsem čerpal z [13] a navazujících dokumentů.

Nejprve krátce pohovořím o technikách, bez kterých by nebyl možný vznik AJAXu ani webových aplikací.

2.1 Technologie používané na webu

2.1.1 Přenos informací z prohlížeče na server

Pokud má webový prohlížeč fungovat jako uživatelské rozhraní webové aplikace, musí být umožněn přenos informací z prohlížeče na webový server. Metoda POST a query string metody GET toto umožňuje. Obě mětody mají téměř stejné možnosti, ale vzhledem k tomu, že query string se zobrazuje v adresovém řádku prohlížeče, používá se jen pro přenos krátkých

informací. Také je potřeba dbát na to, že podle specifikace HTTP (RFC 2616), patří GET mezi takzvané bezpečné metody (safe methods) a ty nemají provádět žádné akce, kromě zobrazování informací. To znamená především, že nesmí nic mazat ani měnit. Přenášení informací od klienta na server také umožnilo zachovávání informace o stavu relace, i když je HTTP nestavovým protokolem. Informace o stavu relace jsou uloženy na serveru, ten každé relaci přidělí číslo, které přidá do query string všech odkazů na stránce. Tak server pozná, které informace patří které relaci. Další variantu pro přenos čísla relace přinesly cookies (RFC 2965). Bohužel však kvůli chybám v implementaci prvních prohlížečů podporujících cookies někteří uživatelé této technice nedůvěřují a mají ji v prohlížeči vypnutou. Dnes už by cookies neměly být žádnou bezpečnostní hrozbou a i v minulosti hrozilo pouze, že bude někdo díky cookies sledovat, jak se uživatel pohybuje po webových stránkách. Z počátku se tyto metody HTTP používaly jen při odesílání formulářů. S příchodem JavaScriptu se však jejich využití značně rozšířilo. Informace o HTTP protokolu jsem čerpal z [15] a [19].

2.1.2 JavaScript

Původní název JavaScriptu byl LiveScript, vytvořil ho Brendanem Eichem z Netscape a poprvé byl implementován v Netscape Navigator 2.0, který vyšel v roce 1995. Jak napovídá název, JavaScript je nekompilovaný skriptovací jazyk, jehož syntaxe byla inspirována jazykem Java. Skripty v jazyce JavaScript jsou odesílány prohlížeči uvnitř HTML dokumentů uzavřené v tagu `script` a jsou prováděny v prohlížeči. V prohlížeči tak začalo být možné kontrolovat meze zadávaných hodnot ještě před odesláním na server. Vzhledem k tomu, že se taková kontrola dá jednoduše obejít vypnutím JavaScriptu, musí server meze kontrolovat znova, ale pokud má uživatel JavaScript zapnutý, nebudou špatně zadaná data na server odeslána a uživatel se okamžitě doví, že udělal chybu, a nebude muset čekat na odpověď serveru. Využití JavaScriptu je dnes mnohem širší, než popisuji v tomto jednoduchém příkladu.

2.1.3 Document Object Model

Schopnosti JavaScriptu asi nejvíce rozšířila standardizace DOM (Document Object Model, česky objektový model dokumentu) a jeho implementace do prohlížečů. DOM popisuje, jak má být objektově orientovaným způsobem popsán HTML dokument a tím vytváří pro programy a skripty rozhraní, které umožňuje měnit obsah a formátování HTML dokumentu. Díky DOM je tedy možné v závislosti na vstupu uživatele například nechat zčervenat popisek špatně vyplněného textového pole a zobrazit vedle něj rámeček popisující, jaké chyby se uživatel dopustil.

2.1.4 Rámce

Rámce představovaly další krok k technologii, které dnes říkáme AJAX. První prohlížeč, který umožňoval používání rámců, byl opět Netscape Navigator 2.0, ale standardizovány byly až v HTML 4.0. Rámce umožňují rozdělit okno prohlížeče na několik pevných částí, přičemž v každé z nich může být zobrazen jiný HTML dokument. Rámce se používaly na vložení pevného navigačního menu do jedné části okna prohlížeče a obsahu stránky do druhé části okna prohlížeče. Přičemž všechny odkazy z menu se zobrazovaly v části s obsahem. Od této techniky se později upustilo, protože komplikovala vytváření odkazů na stránku (neměnil se adresový řádek) a často bylo těžké zakomponovat do stránky nevzhledné okraje rámců. Načíst jinou stránku do rámce bylo možné i zavoláním funkce JavaScriptu.

2.1.5 Technika neviditelných rámců

JavaScriptem běžícím v jednom rámci je tedy možné změnit dokument zobrazený v jiném rámci. To přivedlo vývojáře webových stránek a aplikací na myšlenku, že by mohli takto přenášet informace mezi serverem a prohlížečem, aniž by musel být znovu načítán celý dokument. Stránka by tak mohla provádět na pozadí takové akce, jejichž výsledek nemusí být okamžitě znám, a uživatel by mezitím mohl pokračovat v práci.

Příkladem může být například kontrola existence vyplněné adresy v databázi. Webový server si totiž nemůže dovolit každému uživateli před vyplňováním formuláře odeslat celou databázi platných adres a musí tak adresu kontrolovat až po odeslání. Pro uživatele by však bylo příjemnější, kdyby se o neplatné adrese dověděl již při vyplňování a byl na to upozorněn. Právě toto umožňuje technika neviditelných rámců. Vývojář definuje ve skupině rámců jeden rámec, jehož výška nebo šířka je nastavena na 0. Takto vznikne neviditelný rámec, do něhož je načten HTML dokument s formulářem s neviditelnými poli. V příkladu, který jsem uvedl, by se do neviditelných polí formuláře vkládaly jednotlivé části adresy vyplňované uživatelem v hlavním rámci a v momentě, kdy by již byly vyplněny všechny povinné části adresy a uživatel by začal vyplňovat jiné pole, odeslal by JavaScript formulář z neviditelného rámce. Do neviditelného rámce by se tím načetl HTML dokument s JavaScriptem, který by informoval JavaScript hlavního rámce, že data jsou k dispozici. JavaScript hlavního rámce by si pak data vyzvedl a upozornil uživatele, že zadaná adresa nebyla nalezena v databázi.

Tato technika byla prvním modelem asynchronního požadavku a odpovědi a je také první technikou, kterou můžeme považovat za AJAX. Vylepšením této techniky je využít místo skupiny rámců plovoucí rámec, tag `iframe`. Zjednoduší se tak struktura stránky, protože plovoucí rámec se vkládá do dokumentu HTML, vylepší se vzhled stránky, protože prohlížeče i u rámců s nulovou výškou nebo šířkou zobrazují okraj, a zjednoduší se komunikace mezi rámci.

2.1.6 AJAX

Pokud jste se s termínem AJAX ještě nesetkali, pravděpodobně vás teď napadla otázka, co je to AJAX? AJAX znamená Asynchronous JavaScript and XML (asynchronní JavaScript a XML). A podle prvního o něm publikovaného článku [16] by se dalo říct, že se jedná o nový název pro známé techniky tvorby webových aplikací. AJAX skutečně nepřináší do hry žádné nové technologie, ale spíš nový a ucelený pohled na používané technologie. A i když se zdá, že to není nic zázračného, posunul AJAX známé technologie dál, rozhýbal vývojáře webových frameworků a tvůrce webových aplikací a začaly se objevovat nástroje na implementaci AJAXu do webových stránek.

Před tím než vznikl termín AJAX, nazývali tyto techniky ti, kdo je používali, například vzdáleně spouštěný JavaScript, vdálené volání procedur nebo dynamická aktualizace.

AJAX je určitý přístup ke komunikaci mezi prohlížečem a serverem. Je založen na přenosu malého množství informací k serveru a především od něj, aby interakce s uživateli byla co nejrychlejší. Ve zmíněném článku [16] Garrett zavádí termín engine AJAXu (doslovný překlad by byl motor AJAXu, ale volně bych ho nazval spíše jádro AJAXu). Jádro AJAXu je JavaScript, který je při komunikaci prohlížeče a serveru vložen mezi server a zobrazovací logiku v prohlížeči. Jestliže má být ve webové aplikaci něco změněno a je na to potřeba komunikace se serverem, ale není potřeba překreslení celé stránky, zavolá se funkce JavaScriptu, která odešle dotaz na server a nastaví, jaká funkce má být zavolána, až bude odpověď k dispozici. Jakmile server odpoví, zvolená funkce je spuštěna a provede

změny na stránce odpovídající obsahu odpovědi. Data vrácená serverem přitom mohou být v jakémkoli formátu srozumitelném pro jádro AJAXu. Jádro AJAXu si nejprve museli tvůrci webových aplikací celé tvořit sami. Dnes už je ve většině webových frameworků hlavní část jádra AJAXu naprogramovaná a tvůrce tak už jen přidá funkčnost, kterou potřebuje.

XMLHttp

AJAX by ale ani zdaleka nebyl tak elegantním řešením, kdyby neexistoval ActiveX prvek nazvaný XMLHttp. Tento ActiveX je možné v Internet Exploreru použít z JavaScriptu a umožňuje vytvořit požadavek na HTTP server, který je zcela ovládán JavaScriptem. JavaScriptu XMLHttp zpřístupňuje nejen obsah odpovědi HTTP serveru, ale i hlavičky HTTP protokolu, umožňuje odeslat vlastní hlavičky i přečíst si přijaté hlavičky. Také JavaScript informuje, že spojení se serverem nebylo navázáno. Samozřejmostí je, že JavaScript může najednou čekat i na více odpovědí serveru. Později byl i v Mozzile implementován podobný objekt s názvem XMLHttpRequest, který se pak objevil i v dalších prohlížečích včetně Internet Explorer 7.

Informace o AJAXu a technologiích, které zastřešuje jsem čerpal z [30].

2.1.7 SEO

SEO (Search Engine Optimization, česky optimalizace pro vyhledávače) je důležité pro webové aplikace, do kterých chceme přilákat uživatele. Může se jednat o elektronické obchody nebo katalogy produktů. Pokud je SEO ve vytvářené webové aplikaci potřebné, musí se při výběru webového framewokru dát pozor na to, aby umožňoval vytvářet URL ve tvaru, který je potřeba na uplatnění SEO. Například je lepší, když URL vypadá takto:

```
http://www.knihy.cz/bozena_nemcova/babicka
```

než když má takovýto tvar:

```
http://www.knihy.cz/products.php?autor=bozena+nemcova&kniha=babicka
```

2.2 Platformy pro implementaci webových aplikací

Původně se tato kapitola měla jmenovat „Jazyky pro implementaci webových aplikací“, nakonec jsem se však rozhodl pro název zastřešující jak použitý jazyk, tak softwarové prostředky potřebné pro jeho používání (knihovny), překlad (překladače) a spouštění (virtuální stroje a JIT kompilery).

Na straně klienta existuje sice mnoho platforem, které se nazývají prohlížeče, ale jen jediný programovací jazyk, který je podporován ve všech nových prohlížečích, a tím je JavaScript. Volba jiného jazyka by proto nebyla kompatibilní se všemi prohlížeči, a proto se jinými možnostmi nebudu zabývat. Za to na straně serveru je obrovský výběr jazyků použitelných pro vývoj jakékoli webové aplikace. U jednotlivých jazyků se budu zaměřovat na schopnosti jazyka a možnosti, které vývojáři přináší jazyk samotný, a dále pak na způsob, jakým je aplikace napsaná v daném jazyce prováděna na serveru a podle toho zhruba zhodnotím zda daná platforma nevyužívá zbytečně mnoho systémových prostředků. Neprováděl jsem žádné výkonnostní testy žádné webové aplikace napsané v jednotlivých jazycích a to především proto, že hodně záleží na tom, jak je aplikace napsaná a o jakou aplikaci se jedná a testy by z toho důvodu nebyly věrohodné.

2.2.1 Common Gateway Interface

Není zcela správné zařadit CGI (Common Gateway Interface) do kapitoly o platformách, nechtěl jsem ho ale opomenout, když se jedná o historicky první prostředek umožňující vývoj webových aplikací, který se stále občas používá. CGI definuje rozhraní, které může libovolná aplikace spustitelná v daném prostředí (v daném operačním systému) použít pro komunikaci s webovým serverem podporujícím CGI. Aplikaci se pak obvykle říká CGI skript nebo CGI aplikace. Rozhraní je koncipováno tak, že pokud dorazí na server požadavek na určitou adresu, je tento požadavek poslán danému CGI skriptu, ten ho zpracuje, sestaví odpověď a tu server vrátí webovému prohlížeči. Z uvedeného popisu je zřejmé, že pro napsání CGI skriptu tím pádem může být použit libovolný programovací jazyk.

To, co jsem prozatím napsal, asi napovídá, že CGI je hojně používanou technikou na tvorbu webových aplikací. Pravdou však je, že se v dnešní době CGI používá jen minimálně. Důvodem je především pomalost rozhraní a špatná modularita CGI aplikací. Rozhraní je pomalé, protože se při každém požadavku musí CGI skript zavést do paměti, vytvořit výsledek a ukončit. Vhledem k tomu, že fáze vytvoření výsledku je obvykle velmi krátká, zpomaluje provádění znatelně zavádění skriptu do paměti a ukončování skriptu. Špatnou modularitou mám na mysli to, že není žádné rozhraní, které by se dalo obecně použít na poskládání CGI aplikace z různých samostatných komponent.

Informace o CGI jsem čerpal z [3] a [12].

2.2.2 Platformy Java 2 a .NET Framework

Platformy Java 2 a .NET Framework mají mnoho společného. Platformy Java 2 umožňují běh programů napsaných v jazyce Java. Programy jsou překládány do bajt kódu Javy. Bajt kód vzniklý při překladu je pak možné spustit na libovolném výpočetním sytému disponujícím odpovídajícím JRE (Java Runtime Environment). V JRE je bajt kód buď interpretován virtuálním strojem nebo dnes častěji překládán za běhu do strojového kódu pomocí JIT (just-in-time, česky právě včas) kompileru. Platformy Java 2 se liší především obsáhlostí knihoven podle cílového prostředí, ve kterém vytvořené programy poběží:

- J2ME (Java 2 Platform, Micro Edition) je určena pro vývoj aplikací pro malá zařízení, jako jsou mobilní telefony,
- J2SE (Java 2 Platform, Standart Edition) je určena pro vývoj aplikací pro osobní počítače a
- J2EE (Java 2 Platform, Enterprise Edition) je určena především pro vývoj aplikací s architekturou klient-server.

Platí zde přitom pravidlo, že na vyšší platformě (uvedené v seznamu níže) je možné používat vše, co bylo dostupné na nižší platformě (v seznamu výše). Pro vývoj webových aplikací je tedy určena platforma J2EE.

Platforma .NET Framework má jen dvě verze:

- .NET Compact Framework, který má stejný účel jako platforma J2ME, a
- .NET Framework, který zastřešuje schopnosti J2SE a J2EE.

Obě platformy obsahují CLR (Common Language Runtime), který plní stejný účel jako JRE v platformách Java 2. CLR umožňuje běh mezikódu s názvem CIL (Common Intermediate Language). Do CIL je možné přeložit hned několik programovacích jazyků, v této

práci se však budu zabývat jen jazykem C#, který byl vytvořen speciálně pro účely .NET Frameworku.

Jak J2EE, tak .NET Framework tedy vytvářejí mezikód překladem zdrojového kódu a provádějí jej pomocí JIT kompileru. To oběma umožňuje provádět optimalizace při překladu do mezikódu i při překladu do strojového kódu v čase běhu aplikace. JIT kompiler samozřejmě vytvořený strojový kód hned nezahazuje, ale snaží se jej použít znova při každém dalším průchodu danou částí kódu. Díky tomu, že platformy používají stejnou koncepci, budou na tom výkonnostně zřejmě velmi podobně.

Pohled do historie

Podle mého názoru bychom měli být rádi, že vznikl .NET Framework, protože tím vzniklo konkurenční prostředí, ve kterém dochází k rychlejšímu vývoji platforem ve snaze přežít a růst. Tvůrci platforem Java 2 přitom podle mého názoru postupují konzervativněji. V následujících odstavcích proto popíšu rozdíly mezi C# a Javou v historickém kontextu, abych dokázal předchozí tvrzení. Nebudu se zabývat možnostmi celých platforem, ale pouze možnostmi jazyků v kontextu těchto platforem. Java i jazyky přeložitelné do CIL jsou objektově orientované, přísně typové a umožňují používání jmenných prostorů, kterým se v Javě říká packages (balíčky) a v C# namespace (jmenný prostor). Balíčky se od jmenných prostorů liší pouze způsobem použití. Balíčky se umisťují do adresářové struktury shodné s hierarchickým členěním názvu balíčku. Na rozdíl od toho jmenné prostory nemají žádnou souvislost s adresářovou strukturou a platí u nich pouze nepovinná konvence pro vytváření názvu jmenného prostoru. Dalším rozdílem, který zde byl již od vzniku obou jazyků, je, že v Javě může jeden soubor obsahovat pouze jednu třídu, která bude dostupná i mimo tento soubor (nepočítám však vnořené třídy, o kterých bude řeč později), kdežto v C# jich může být i více. C# na rozdíl od Javy umožňuje vytváření vlastních hodnotových typů nazývaných struktury (struct), mutlidimenzionálních polí, delegátů (delegate, bezpečný a rychle pracující ukazatel na metodu objektu), událostí (event, spíš syntaktické usnadnění používání delegátů), předávání parametrů metod odkazem, přetěžování operátorů a vytváření metod, které nejsou virtuální (což je v C# implicitní, virtuální metody musí být označeny). Takovýchto kosmetických rozdílů je mnohem více a každý má své důvody, výhody i nevýhody. Dále se již nebudu zmiňovat o možnostech, které poskytují již od prvního vydání obě platformy.

První platformou Javy bylo JDK 1.0 vydané 23. ledna 1996. Další byla JDK 1.1 z 19. února 1997, ta přinesla do Javy zmíněné inner classes (které jsem nazval vnořené třídy). Kód vnořené třídy je celý zapsán do třídy vnější a její metody mají přístup k metodám a členským proměnným vnější třídy. Z toho jasně plyne, že instance vnitřní třídy musí být vytvořena v metodě nebo konstruktoru vnější třídy, aby se mohla odkazovat na metody a proměnné instance vnější třídy. V následujícím vydání Javy z 8. prosince 1998 bylo upuštěno od jedné platformy společné pro všechny cílové skupiny programů a vznikly platformy J2ME, J2SE a J2EE verze 1.2. Do Javy byla přidána reflexe, což je možnost v aplikaci za běhu procházet seznamy tříd, metod, proměnných a podobně. Verze 1.3 vydaná 8. května 2000 nepřinesla žádné významné změny jazyka.

Tím však pomalu končí existence Javy jako sólového hráče a 5. ledna 2002 je vydán .NET Framework 1.0. Ten má všechny zmíněné schopnosti Javy jen s tím rozdílem, že vnitřní třídy je možné vytvořit odkudkoli, takže se v nich automaticky nemůžou používat metody a proměnné vnější třídy a programátor si musí do konstruktoru předat instanci vnější třídy, pokud to potřebuje. Reflexe navíc umožňuje i vytváření nových tříd a metod

za běhu aplikace (tato problematika je velmi složitá a nebudu se jí dále zabývat). .NET Framework dále umožňuje ukládat ke třídám, metodám a členským proměnným metadata ve formě atributů. Díky tomu je možné automatické zpracování tříd například při serializaci, kdy se pomocí atributů označuje, které vlastnosti objektu mají být serializovány, a objekt provádějící serializaci pak nemusí o dané třídě nic vědět a vše potřebné mu sdělí metadata třídy. Tento přístup značně zpřehledňuje zdrojový kód aplikace.

Další verze Javy vydaná 6. února 2002 opět nepřináší žádné podstatné změny jazyka, výrazně však usnadňuje použití XML a regulárních výrazů. Tím dohání s měsíčním zpožděním první verzi .NET Frameworku. Následuje verze 1.1 .NET Frameworku vydaná 1. dubna 2003, která je také pro toto srovnání nedůležitá. 30. září 2004 pak bylo vydáno J2SE verze 5.0 (podle původního číslování verze 1.5). Ta přinesla mnoho užitečných změn. První z nich umožnila přidání metadat, kterým se v jazyce Java říká anotace. Rozdíl mezi anotacemi a atributy v C# je především v tom, že atributy jsou vždy dostupné za běhu aplikace a každý atribut je instancí třídy. Oproti tomu u anotací se nastavuje, zda mají být dostupné za běhu aplikace, a při vytváření anotací se vytváří jen rozhraní, takže je programátor může použít pouze jako kolekci klíčů a hodnot, která se na začátku nataví a není ji už možné změnit. Dále tato verze přinesla generické třídy, které vzdáleně připomínají šablony v C++, ale nemají tak široké použití. Slouží například k definování typu parametru a návratové hodnoty metody nebo k definování typu objektů vkládaných do kolekce. To umožnilo konečně typovou bezpečnost kolekcí. J2SE verze 5.0 také umožnila vyváření výčtových typů (enums), metod s proměnlivým počtem argumentů (vararg) a syntaxi podobnou foreach na průchod všech prvků kolekce, což vše umožňovala i první verze .NET Frameworku.

.NET Framework nezůstal dlouho pozadu a 7. listopadu 2005 vyšla verze 2.0, která také implementovala generické třídy a to ještě důmyslnějším způsobem, protože jejich podporu vložila přímo do CIL, na rozdíl od Javy, kde je kvůli zpětné kompatibilitě podpora pouze na úrovni překladače. Dále bylo do C# přidáno nové klíčové slovo yield, které usnadňuje vytváření enumerátorů (počeštil jsem anglické slovo enumerator). Myslím si, že toto klíčové slovo není moc známé, proto uvedu příklad:

```
public static IEnumerable<T> Where<T>(
    IEnumerable<T> source,
    Predicate<T> predicate
    )
{
    foreach (T item in source)
        if (predicate(item))
            yield return item;
}
```

Vytvořenou metodu lze použít například takto:

```
string[] test = "Ahoj, jak se mas?".Split(' ');
IEnumerable<string> result = Where(test, delegate(string item){
    return item.Length > 3;
});
foreach (string word in result)
{
    Console.WriteLine(word);
}
```

Výstupem tohoto příkladu budou řetězce „Ahoj,“ a „mas?“. Přičemž k porovnávání délek jednotlivých prvků pole dochází až v cyklu foreach, ve kterém se slova vypisují do konzole a nevzniká při tom žádné pomocné pole ani kolekce. Je to proto, že překladač metodu `Where<T>` upraví tak, že do její třídy podle těla metody vygeneruje třídu odvozenou od `IEnumerable<T>`. V této třídě je pak tělo metody `Where<T>` implementováno jako stavový automat a samotná metoda pak pouze vrací instanci této nové třídy. .NET Framework verze 2.0 také umožnil rozdělit implementaci třídy do více souborů. 6. listopadu 2006 pak byl vydán .NET Framework verze 3.0, který pouze přidal nové knihovny a zvýšil kooperaci s novými operačními systémy od firmy Microsoft. 11. listopadu 2006 vyšla zatím poslední J2SE verze 6.0, změny však pro mé srovnání nebyly důležité.

.NET Framework verze 3.5

Poslední .NET Framework verze 3.5 vyšel 19. listopadu 2007 a přinesl podstatné změny. Umožnil několik syntaktických kliček, které mohou v určitých situacích velmi usnadnit práci. První z nich je implicitní typování, které určitě ušetří spoustu napsaných znaků a zpřehlední kód. Vypadá takto:

```
var seznam = new Dictionary<string, Osoba>();
var retezec = "ahoj";
```

Zdálo by se, že je zde použito dynamické typování známé například z PHP nebo Pearlu. To však není pravda, C# zůstává přísně typovaným jazykem, v těchto případech je typ proměnných `seznam` a `retezec` určen podle typu přiřazované hodnoty a následující výrazy jsou proto zakázané a překladač bude hlásit chybu:

```
retezec = 1;
var nic = null; // zde není možné určit typ, proto je to chyba
```

Podoným usnadněním jsou inicializátory objektů, které umožňují při vytváření objektu nastavit jeho vlastnosti výčtem, například místo:

```
var osoba = new Osoba();
osoba.Jmeno = "Radek";
osoba.Prijmeni = "Petrik";
```

stačí napsat:

```
var osoba = new Osoba(){
  Jmeno = "Radek",
  Prijmeni = "Petrik"
}
```

Na stejném principu jsou založeny i inicializátory kolekcí, které jsou použitelné pro všechny kolekce implementující rozhraní `ICollection<T>`:

```
var seznam = new List<Osoba> {
  new Osoba(){Jmeno = "Radek", Prijmeni= "Petrik"},
  new Osoba(){Jmeno = "Jan", Prijmeni= "Novak"}
}
```

Dále bylo umožněno používání anonymních typů, kdy je vytvořen při překladu nový typ, který není pojmenován, například:

```
var osoba = new {
  Jmeno = "Radek",
  Prijmeni = "Petrik"
}
```

vytvoří nový typ, který má pouze dvě vlastnosti Jmeno a Prijmeni. Poslední čistě syntaktickou novinkou jsou lambda výrazy, které budu ilustrovat na kolekci seznam vytvořené výše, seřazení podle příjmení by se dalo udělat takto:

```
seznam.Sort( delegate(Osoba a, Osoba b) {
  return a.Prijmeni.CompareTo(b.Prijmeni);
};
```

což je teď možné elegantně zkrátit na:

```
seznam.Sort( (a,b) => a.Prijmeni.CompareTo(b.Prijmeni) );
```

U složitějších metod je možné za `=>` vložit blok příkazů, který už však musí obsahovat i klíčové slovo return pro vrácení hodnoty. Předposledním syntaktickým vylepšením je možnost využití rozšiřujících metod (extension methods). Tyto metody opět pouze usnadňují práci a umožňují napsat místo:

```
public static class OsobaRozsireni {
  public static int PorovnejPrijmeni(Osoba a, Osoba b)
  {
    retrun a.Prijmeni.CompareTo(b.Prijmeni);
  }
}
OsobaRozsireni.PorovnejPrijmeni(osobaA, osobaB);
```

následující kód:

```
public static class OsobaRozsireni {
  public static int PorovnejPrijmeni(this Osoba a, Osoba b)
  {
    retrun a.Prijmeni.CompareTo(b.Prijmeni);
  }
}
osobaA.PorovnejPrijmeni(osobaB);
```

Důležité při tom bylo přidání klíčového slova `this` před deklaraci parametru `osobaA`. Podmínkou funkčnosti je také, aby metoda PorovnejPrijmeni byla statickou metodou statické třídy. Tímto způsobem lze rozšířit i uzavřenou třídu (sealed class). Typem parametru `this` může být i interface, takže je možné snadno rozšířit všechny třídy implementující tento interface. Překladač přitom používá rozšiřující metodu pouze pokud nenalezne odpovídající metodu v rozširované třídě. Dále je potřeba upozornit, že volání rozšiřující metody je přímé a pokud bude stejná metoda přidána do rozšířené třídy a nedojde k opětovnému přeložení kódu, který používá rozšiřující metodu, bude stále volána. Tato praktika by se určitě neměla používat často, protože může mást a bez dobrého vývojového prostředí může být těžké dohledat, jakou metodu ve skutečnosti překladač zavolá. Název třídy s rozšířeními by asi měl obsahovat název rozšiřované třídy a účel rozšíření. Posledním syntaktickým vylepšením, ke kterému ty předchozí vedly, je Linq (Language Integrated Query, dotaz integrovaný do jazyka). Linq umožňuje v C# napsat něco, co už vůbec nevypadá jako C#:

```
var nalezeno = from o in seznam
               where o.Prijmeni == "Petrik"
               select o.Jmeno;
```

Po provedení tohoto řádku se do proměnné `nalezeno` uloží kolekce řetězců obsahující jména osob v `seznam`, které mají příjmení „Petrik“. Ve skutečnosti je tento řádek přeložen do něčeho podobného:

```
var nalezeno = seznam.Where(o => o.Prijmeni == "Petrik")
  .Select(o =>  o.Jmeno);
```

kde metody `Where` a `Select` můžou být rozšiřující metody. Linq nabízí i další možnosti jazyka SQL jako je `join`, `order by` a `group by`. Tímto přehledem jsem chtěl pouze nastínit nové možnosti, které tato syntaktická vylepšení přinášejí. Podrobnosti je možné nalézt ve článku [14], ze kterého jsem čerpal.

Informace o jazyce Java a CIL jsem čerpal z [23] a [26].

Zhodnocení

Podle mého názoru je mezikód, tím myslím CIL a bajt kód Javy, obou platforem velmi podobný a CIL pouze poskytuje navíc podporu pro generické třídy a nevirtuální metody. Obě tyto odlišnosti běh aplikací trochu urychlují, protože CLR nemusí při používání generických tříd stále provádět přetypování a při každém volání nevirtuální metody používat tabulku virtuálních metod, což JRE dělá. Výkonnostní rozdíly však budou pravděpodobně minimální.

Hůř je na tom Java ve srovnání překladačů, kdy překladač C# umožňuje použití množství syntaktických vylepšení, které třeba v případě klíčového slova yield ušetří programátorovi spoustu práce a při tom nemají žádný vliv na výkonnost aplikace. Dalo by se považovat za nevýhodu, že je těch syntaktických zvláštností tolik, ale stále platí, že kdo je nechce používat nebo jim nerozumí, nic ho k tomu nenutí, protože jsou to jen syntaktická vylepšení a je možné programovat i bez nich. C# tedy podle mého názoru určitě vede na poli komfortu při psaní.

Kde však C# jednoznačně zaostává je přenositelnost. Vůbec není snadné napsat složitější aplikaci tak, aby fungovala zároveň na několika operačních systémech či architekturách. Existuje několik projektů implementujících CLI[1]. Tyto platformy často implementují také knihovny, které jsou v .NET Frameworku a nejsou specifikované v CLI. Všechny jsou však oproti .NET Frameworku pozadu a většina má kompletně implementovanou teprve verzi 1.1. Oproti tomu i poslední verze platformy Java 2 jsou použitelné na většině podporovaných operačních systémů a architektur. Podle mého názoru je J2SE verze 6.0 srovnatelná s .NET Frameworkem verze 1.1 a navíc je podporována na více operačních systémech, takže je stále vítězem v přenositelnosti napsaného kódu.

2.2.3 PHP

PHP znamená „PHP: Hypertext Preprocesor“, jedná se tedy o rekurzivní akronym. PHP je skriptovací jazyk vkládaný do HTML. Je použitelné i mimo prostředí webového serveru, ale je maximálně uzpůsobené pro běh na webovém serveru a na dynamické generování webových stránek respektive na tvorbu webových aplikací.

[1] Common Language Infrastructure je norma pro vytváření platforem pro CLI, kterou dodržuje i .NET Framework

Syntaxe a možnosti jazyka

Syntaxe PHP je hodně podobná jazykům C a Java a některé prvky si vypůjčuje i z Pearlu. PHP skripty mohou být v HTML označeny několika způsoby:

- standardním způsobem `<?php PHP skript ?>`,
- zkráceným způsobem `<? PHP skript ?>`,
- ve stylu ASP `<% PHP skript %>`,
- v HTML skript standardu `<script language="PHP"> PHP skript </script>`

a platí, že vše mimo označený PHP skript je při průchodu programu danou částí odesláno jako výstup programu. V PHP skriptu je možné použít příkaz `include soubor.php;` respektive `include_once soubor.php`, který zpracuje soubor `soubor.php` stejně jako by byl vložen přímo do souboru, ze kterého je příkaz volán. Oddělení PHP skriptu od HTML je tedy možné, ale samozřejmě k němu programátor není nucen. V PHP není potřebná definice proměnných a je tedy možné uložit do proměnné hodnoty různých typů. Od PHP verze 4 lze v PHP vytvářet i třídy a jejich instance. V PHP 5 byl však objektový model zcela přepracován.

PHP je často vytýkáno, že neobsahuje jmenné prostory, má nekonzistentní názvy funkcí[2] a často existuje několik funkcí, které dělají naprosto stejnou činnost. Absence jmenných prostorů komplikuje v PHP vytváření modulů. PHP také neobsahuje žádné možnosti pro vkládání metadat do kódu, pro jejich ukládání se proto používají konfigurační soubory. Osobně mi v PHP chybí výčtový typ, který musím nahrazovat třídou. Vzhledem k tomu, že PHP je dynamicky typovaný jazyk, generické třídy nepotřebuje. Přes všechny nevýhody jazyka PHP je však hodně používaný. Podle mého názoru je to díky jeho jednoduchosti a díky podpoře na většině webových serverů. Pokud však programátor dodržuje určité zásady a místo jmenných prostorů používá třídy, je možné i v PHP vytvořit velmi přehlednou webovou aplikaci, jazyk PHP to však příliš neusnadňuje. Základní příkazy PHP neumožňují pohodlné programování složitých aplikací, je však možné používat různé moduly, které vývoj velmi usnadní. Více o modulech je možné najít v [28]. Informace o syntaxi jazyka jsem čerpal z [18].

Zend engine a urychlování PHP

Zend engine je jádro PHP, které vykonává PHP skripty. Zend engine definuje syntax jazyka a PHP definuje sémantiku. Zend engine pracuje tak, že nejprve provede převod PHP skriptu na symboly, pak jej zkompiluje do posloupnosti operačních kódů, které nakonec vykoná. Problém je, že toto provádí při každém volání skriptu, což prodlužuje dobu potřebnou na provedení skriptu. Toto by bylo výraznou nevýhodou PHP v souboji například proti .NET Framevorku nebo J2EE. Naštěstí existují tak zvané akcelerátory PHP, které vytvářejí code cache. Code cache ukládá pro každý skript seznam operačních kódů a pokud pro prováděný skript již existuje takový seznam, neprovádí se znova převod na symboly ani kompilace, ale rovnou se provedou operační kódy. Code cache takto vytváří kód, který je principielně velmi podobný bajt kódu Javy a CIL. Situaci jsem trochu zjednodušil a přesný popis je možné najít v [28] na stranách 267 - 274.

[2] slova v názvech jsou používána v různém pořadí a někdy je v názvech používán znak „podtržítko“ a jindy ne

Nesmím ovšem opomenout, že i když použiji code cache, stále budou pravděpodobně o něco rychlejší JRE a CLR, protože ty navíc používají JIT kompiler, který překládá mezikód do strojového kódu. Nepodařilo se mi nalézt žádný JIT kompiler pro PHP a podle článku [27] to vypadá, že se ani nikdo k jeho tvorbě nechystá a je doporučeno psát v jazyce C rozšíření (extensions) do PHP, které jsou určitě rychlejší než JIT kompiler. To je sice pravda, ale psát v PHP je řádově jednodušší než psát v C. Druhým aspektem je, o kolik by se provádění PHP mohlo urychlit, když by byl použit JIT kompiler? Určitě by výsledek nebyl lepší než u PHP přeloženého do strojového kódu ještě před spuštěním. Naštěstí existuje projekt RoadSend, který provádí překlad PHP do strojového kódu. Jeho autoři vytvořili také srovnání rychlosti PHP prováděného Zend enginem 4.4.2 bez code cache a strojového kódu vytvořeného jejich překladačem. Srovnání je možné najít na URL http://www.roadsend.com/home/index.php?rIDX=1&pageID=benchmarks (funkčnost odkazu jsem naposledy ověřoval 9. května 2008). Z testu je patrné, že strojový kód je zhruba 2x rychlejší. Použití code cache by v tomto případě výsledek moc nezměnilo, protože testovací příklady jsou krátké, spouštějí se jen jednou a probíhá v nich buď dlouhý cyklus nebo složitá rekurze. Dalším důkazem, že JIT kompiler pro PHP má význam, je projekt Quercus (informace o projektu jsem čerpal z [6]), který umožňuje spuštění PHP na JRE a jeho autoři slibují, že je 4x rychlejší než modul mod_php pro server Apache. Podobný je projekt Phalanger, který umožňuje překlad PHP do CIL. Výsledek prvního testu výkonnosti Phalangeru [24] vypadá velmi slušně, test ukazuje, že je téměř 2x rychlejší než PHP 5.1.6. Druhý test [22] už není pro Phalanger tak úspěšný, podle mě především proto, že pracuje pouze se syntaktickými obraty PHP, které zřejmě není snadné převést do CIL. I v druhém testu je však Phalanger rychlejší, rozdíl už však není tak velký a značně se liší výsledky jednotlivých částí testu. Více informací o tomto projektu je možné nalézt v jeho dokumentaci [25]. Oba zmíněné projekty převádějí PHP skripty do mezikódu, který je pak spouštěn v JIT kompileru a oba mají velmi dobré výsledky, což považuji za další důkaz toho, že JIT kompiler by běh PHP urychlil. Bylo by proto vhodné ho vytvořit a to, že neexistuje, je nevýhoda oproti .NET Frameworku a J2EE. Na tomto místě je však nutné zdůraznit, že „výpočetní výkon“ PHP sice zaostává za svými konkurenty, ale způsob jakým jsou v něm vytvářeny webové aplikace spotřebovává zpravidla méně výpočetního výkonu, a proto se svým konkurentům v celkovém pohledu pravděpodobně vyrovná. Nelze však říct, že toto platí ve všech situacích.

Projekt Phalanger

O výkonnosti projektu Phalanger jsem již hovořil. Protože mi tento projekt připadá velmi zajímavý, věnuji mu ještě pár vět. Phalanger obsahuje i prostředí pro vývoj aplikací v PHP, který je dostupný stejně jako celý projekt pod Microsoft Shared Source Permissive Licence. Phalanger umožňuje kombinovat PHP a ASP, kdy PHP může být použito jako code behind ASP stránky. Také řeší jeden z nedostatků jazyka PHP, umožňuje totiž použití jmenných prostorů. Phalanger také dává programátorovi možnost používat knihovny napsané v CIL místo knihoven napsaných v PHP. Nevýhodou tohoto projektu zůstává, že je plně kompatibilní pouze s CLR, s ostatními implementacemi CLI však kompatibilní není, což určitě v mnoha případech brání jeho použití.

Informace o tomto projektu jsem čerpal z jeho dokumentace [25] a stránek o projektu. Do jaké míry je Phalanger použitelný pro skutečný projekt nemůžu zhodnotit, protože jsem neměl příležitost si jej pořádně vyzkoušet. Rozhodně však je zajímavým projektem, který umožňuje běh přinejmenším fóra phpBB, a doufám, že popožene Zend v tvorbě JIT

kompileru a dalších vylepšení PHP. Sám pak může sloužit jako přestupní stanice pro ty, kteří chtějí začít tvořit webové aplikace v ASP. Varianta, že by se stal novou a hojně užívanou platformou pro provoz aplikací v PHP, je podle mého názoru nepravděpodobná.

2.2.4 Ruby

První verzi Ruby vytvořil Yukihiro Matsumoto v roce 1995. Smíchal v Ruby části svých oblíbených programovacích jazyků Perl, Smalltalk, Eiffel, Ada a Lisp a vytvořil z nich jazyk, který hledá rovnováhu mezi funkcionálním a imperativním programováním. Říká, že se snaží udělat Ruby přirozeným, ne jednoduchým, a přirovnává Ruby k lidskému tělu, protože taky vypadá na pohled jednoduše, ale je velmi složité uvnitř.

Ruby je objektový skriptovací programovací jazyk, ve kterém je vše chápáno jako objekt. Ruby, stejně jako například JavaScript, umožňuje měnit a přidávat třídy objektů za běhu. Kód Ruby je velmi dobře čitelný a většinou okamžitě pochopíte, co daný fragment kódu dělá, aniž byste Ruby znali. Je to opravdu zajímavý jazyk, který je možné použít i na tvorbu webových aplikací, což dokazuje framework Ruby on Rails, který bude popsán dále. Ruby zatím neumožňuje pro webové aplikace žádnou akceleraci v podobě cachování operačního kódu nebo jiného mezikódu. Po stránce schopností jazyka je Ruby velmi dobře vybaven.

Informace o Ruby jsem čerpal z [1].

2.2.5 FastCGI

Pochyboval jsem o tom, zda odstavec o CGI patří do této kapitoly. U FastCGI však nemám sebemenší pochybnost, že sem patří a že musím připomenout jeho existenci. FastCGI bylo vytvořeno v polovině devadesátých let a je následníkem pomalého CGI. Stejně jako u CGI se jedná o rozhraní pro komunikaci mezi webovým serverem a implementací webové aplikace. Abych podpořil své tvrzení, že toto staré a poměrně málo využívané rozhraní má místo i na dnešním trhu, vysvětlím nejprve historické důvody pro přehlížení FastCGI a vysvětlím výhody a nevýhody jeho používání oproti dnes běžně používaným technikám.

FastCGI přišlo na trh příliš pozdě na to, aby se mohlo uchytit. V době, kdy vznikalo FastCGI vznikaly i jiné způsoby, jak propojit webový server s aplikací. Tím byla především různá rozhraní samotných webových serverů umožňující spouštění aplikací přímo v procesu serveru. Příkladem může být rozhraní serveru Apache, které využívá například modul mod_php, sloužící k provádění PHP. FastCGI oproti tomu používá pro komunikaci mezi webovým serverem a aplikací buď TCP/IP spojení nebo roury, pokud jsou webový server i aplikace spuštěny na jednom počítači. V původním CGI se pro komunikaci používaly pouze roury a aplikace se vždy po odeslání odpovědi ukončila. Aplikace používající FastCGI rozhraní se neukončuje, ale po odeslání odpovědi čeká na další požadavek od webového severu.

Je pravdou, že komunikace v rámci jednoho procesu je zřejmě rychlejší, protože nevyžaduje žádnou komunikaci procesů a je velmi přímá. Má však i nevýhody. Modul má v operačním systému stejná práva jako webový server, což může být někdy na obtíž. Modul sdílí paměť s webovým serverem, a proto se může stát, že modul způsobí pád webového serveru kvůli špatné správě paměti. Další komplikací je, že každý proces webového serveru musí mít k dispozici knihovnu modulu a udržovat modul při životě, i když ho právě nepotřebuje, což zase zvyšuje nároky na paměť a částečně i na procesor. Modul nemůže běžet na jiném počítači než webový server, to zase komplikuje rozdělení úloh jedné aplikace na více serverů. Pokud havaruje aplikace, havaruje i webový server a naopak, pokud havaruje webový server, bude zastaven chod aplikace. Při změně konfigurace aplikace je nutné restartovat i webový

server. Další výhodou FastCGI je, že je implementováno na většině webových serverů a tudíž aplikace napsaná pro FastCGI je uplatnitelná ve více prostředích (jedinou podmínkou je, že v daném prostředí musí být spustitelná). Implementace FastCGI je při tom velmi jednoduchá, protože knihovny pro jeho použití jsou napsány pro většinu programovacích jazyků, včetně C# na platformě Mono, a například pro Ruby je to pravděpodobně jediný způsob, jak webovou aplikaci provozovat. Test [11] navíc ukazuje, že rychlost FastCGI je téměř stejná jako rychlost zmíněných modulů. Aplikace napsaná v PHP by tím pádem měla nejrychleji běžet zkompilovaná do strojového kódu pomocí RoadSend a provozovaná přes FastCGI rozhraní.

Informace o FastCGI jsem čerpal z [12].

2.2.6 Závěrečné hodnocení

O tomto tématu se dá napsat mnohem více, než jsem zde uvedl. Pozornost si zaslouží všechny zmíněné jazyky, protože všechny mají své kouzlo. Ať je to jednoduchost PHP, přirozenost Ruby, všestrannost Javy nebo propracovanost CIL. Ve všech zmíněných jazycích je možné napsat webovou aplikaci. Ruby se však moc nehodí v místech, kde je potřeba, aby aplikace zvládala velké množství dotazů.

2.3 Webové frameworky

Je sice možné napsat webovou aplikaci bez použití webového frameworku, ale proč objevovat Ameriku, když už byla dávno objevena. Webové frameworky obsahují nástroje, které umožňují vkládání hotových prvků do webové aplikace. Někdy bývá součástí frameworku i vývojové prostředí usnadňující vývoj aplikace. Nejprve specifikuji, čeho si u frameworků budu všímat, a pak několik vybraných frameworků popíšu podle zvolených kritérií.

2.3.1 Vlastnosti webových frameworků

U webových frameworků budu sledovat několik základních vlastností. Nebudu se zaměřovat na obsáhlost těchto frameworků, protože tu bych byl schopen objektivně posoudit pouze, kdybych s každým z nich delší dobu pracoval. Práci s frameworky, které mě zaujaly a vyzkoušel jsem si je, zhodnotím v závěru práce.

Návrhový vzor frameworku

Asi nejdůležitější vlastností frameworku je, jaký návrhový vzor používá. Společné pro všechny návrhové vzory je, že se snaží oddělit model dat od jejich zobrazení. Budu hovořit o následujících vzorech:

- MVC (Model-View-Controler, česky model-pohled-kontrolér). Pokud hovořím o tomto návrhovém vzoru, mám namysli takzvaný push-based MVC. V tomto modelu vyvolá uživatel akci kontroléru, ten provede odpovídající akci v modelu dat a naplní pohled informacemi, které pak pohled zobrazí uživateli.
- Komponentový vzor (component-based). Také se jedná o druh MVC, ale tentokrát o pull-based MVC. O akci uživatele je v něm informován pohled a ten pak získává data pro zobrazení z různých kontrolérů.

- Post-back model. Jedná se o model využívající post-back. Při post-back se nemění URL stránky a data jsou vrácena tomu pohledu, na němž byla uživatelem provedena akce. Z vrácených dat se pak rekonstruují komponenty pohledu, které následně vyvolávají události. Na události reaguje aplikace a provádí podle nich změny v modelu dat a nakonec znovu zobrazí pohled.

Při tvorbě tohoto rozdělení jsem se inspiroval v [29], [17] a [20].

Budoucnost frameworku

Pokud se má vytvářená aplikace dále vyvíjet, je potřeba také zhodnotit, zda framework přežije vytvářenou aplikaci nebo zanikne dříve než webová aplikace. Určitě by měla být v posledním půl roce vydána nová verze frameworku nebo by se na ní mělo alespoň pracovat. Další zárukou v tomto ohledu může být velká organizace spolupracující na vývoji frameworku nebo velká organizace využívající framework.

Další vlastnosti

Také budu hodnotit, zda framework podporuje AJAX a mapování objektů na relační databázi a umožňuje SEO.

2.3.2 Symfony

Jazykem tohoto frameworku je PHP 5. Používá MVC vzor. Framework je využíván mnoha organizacemi, byl v něm vytvořen například i Yahoo! Bookmarks, takže nepředpokládám, že by v dohledné době zanikl. Framework podporuje AJAX, mapování objektů na relační databázi a má propracovaný systém na podporu tvorby URL pro SEO. Adresa URL je pomocí pravidel mapována na interní URI adresu a naopak. Další informace o frameworku je možné najít v [9].

2.3.3 Zend framework

Jazykem tohoto frameworku je opět PHP 5. Také používá MVC vzor. Za rok 2007 byly vydány 4 nové verze frameworku. Jedná se o nový framework, takže určitě nezanikne v nejbližší době, jeho dlouhodobou existenci však momentálně zaručuje pouze fakt, že ho vytváří Zend Technologies. Framework podporuje AJAX, mapování objektů na relační databázi a URL používané v tomto frameworku je použitelné pro SEO. Další informace o frameworku je možné najít v [10].

2.3.4 Ruby on Rails

Jazykem tohoto frameworku je Ruby. Používá MVC vzor. Každý den je upravován zdrojový kód frameworku. Vzhledem k aktivitě komunity věřím v dlouhou životnost tohoto frameworku. Podporuje AJAX, automatické mapování objektů na relační databázi a URL používané v tomto frameworku je použitelné pro SEO. Další informace o frameworku je možné najít v [7].

2.3.5 MonoRail

Jazykem tohoto frameworku může být libovolný jazyk přeložitelný do CIL. Framework byl inspirován Ruby on Rails a umožňuje použití stejných principů pod .NET Frameworkem. To znamená, že používá také MVC vzor. Téměř každý den je k dispozici nový build. Opět můžu konstatovat, že věřím v dlouhou životnost tohoto frameworku. Podporuje AJAX, automatické mapování objektů na relační databázi a URL používané v tomto frameworku je použitelné pro SEO. Další informace o frameworku je možné najít v [5].

2.3.6 Struts

Jazykem tohoto frameworku je Java. Používá MVC vzor. Ročně vychází několik verzí tohoto frameworku. Vzhledem aktivitě komunity věřím, že bude tento framework ještě dlouho vyvíjen. Podporuje AJAX, mapování objektů na relační databázi a URL používané v tomto frameworku je použitelné pro SEO. Další informace o frameworku je možné najít v [8].

2.3.7 JSF

Jazykem tohoto frameworku je Java. Používá komponentový vzor. JSF vytváří Sun Microsystems, což je podle mého názoru dostatečná záruka, že bude framework dále vyvíjen. Podporuje AJAX, mapování objektů na relační databázi je umožněno například ve spolupráci s Hibernate a URL používané v tomto frameworku je použitelné pro SEO. Další informace o frameworku je možné najít v [4].

2.3.8 ASP .NET

Jazykem tohoto frameworku může být libovolný jazyk přeložitelný do CIL. Používá postback model. Tento framework je vyvíjen Microsoftem, což je opět záruka, že jeho vývoj nebude ukončen. Podporuje AJAX, mapování objektů na relační databázi je umožněno a je možné vytvořit snadno modul, který bude URL v aplikaci upravovat tak, aby bylo použitelné pro SEO. Informace o tomto frameworku jsem čerpal z [20].

2.3.9 Hodnocení

Popisované vlastnosti všech zmiňovaných frameworků ukazují na jejich použitelnost pro tvorbu webových aplikací. Tento pohled byl však velmi povrchní a dokud se všemi frameworky nebudu mít zkušenosti, nemůžu s jistotou říct, zda je skutečně výhodné a vhodné v nich vytvářet webové aplikace. Také jsem nehodnotil vývojová prostředí pro tvorbu v daných frameworcích, zde by určitě nad ostatními vyniklo JSF a ASP .NET.

Jsem rád, že tento přehled ukázal, že vytvořit webovou aplikaci v ASP .NET není krok špatným směrem. Můžu tedy s čistým svědomím dodržet zadání diplomové práce.

Kapitola 3

Zásilková přeprava

Pro pochopení toho, jak funguje moderní zásilková přeprava, je nejlepší projít si životní cyklus zásilky od jejího zabalení u odesilatele až po její přijetí příjemcem. Předem upozorňuji čtenáře, že popis, který bude následovat, se netýká žádné konkrétní firmy poskytující zásilkovou přepravu, ale je poskládaný z úkonů, které se obvykle provádějí a s malými odchylkami odpovídá postupům ve všech firmách.

V textu budu používat terminologii používanou i v zákonech zabývajících se touto problematikou. Přepravce je ten, kdo si objednal dopravu, jinak řečeno ten, kdo si nechává věc či náklad za úplatu přepravit. Dopravce je pak ten, kdo přepravu zajistí většinou vlastními prostředky, ale není to pravidlem. Je důležité odlišit přepravce od termínu odesilatel, protože není nutné, aby přepravce byl i odesilatelem. Když mluvím o odesilateli, mám na mysli buď zpáteční adresu nebo toho, kdo předává zásilku dopravci.

3.1 Před předáním zásilky dopravci

K předání zásilky dopravci může dojít buď přímo v provozovně či bydlišti odesilatele, nebo v provozovně dopravce. Pokud má dojít k předání u odesilatele, musí být tento s dopravcem předem dohodnut, kdy k převzetí dojde. Předání zásilky je pak naplánováno do informačního systému a bude automaticky vytištěno odpovědnému řidiči na jeho soupisku zásilek k vyzvednutí. Řidiči vždy zásilky zároveň rozvážejí i přijímají, proto bývá obvykle soupiska zásilek jen jedna a obsahuje jak zásilky, které je potřeba vyzvednout, tak ty, které musí být rozvezeny.

3.2 Předání zásilky dopravci

Po převzetí zásilky dopravcem již musí být zásilka označena etiketou s čárovým kódem, adresami a případně dalšími kontaktní údaji odesilatele a příjemce. Etikety si většinou vyzvedává v potřebném množství odesilatel u dopravce. Takovýmto sériím etiket se pak říká etiketní řada a každá tato etiketní řada patří konkrétnímu odesilateli. Druhá možnost je, že odesilatel má software poskytnutý přepravcem, umožňující mu vytisknout si etikety z jemu přidělené etiketní řady, když je potřebuje. Třetí a poslední varianta je, že etiketu pro danou zásilku doveze přepravce na místo převzetí zásilky. Při převzetí načte řidič dopravce čárový kód etikety zásilky a tím je zásilka obvykle poprvé zavedena do informačního systému dopravce. Před načtením etikety může být zásilka v informačním systému dopravce jedině v případě, že při telefonické domluvě operátor dopravce zadal do systému číslo etikety

a údaje, které bude etiketa obsahovat, vytiskl ji a byla doručena odesilateli. Řidič je po načtení etikety upozorněn čtečkou na úkony, které je potřeba provést při převzetí daného typu zásilky (například zaplacení poplatku). Pokud byly etikety vytištěny pomocí software dopravce, obdrží řidič od odesilatele také datové médium s údaji o zásilkách, které odesílá. Většinou řidič vystaví nebo obdrží a potvrdí přejímací doklady zásilek.

3.3 Doručení zásilky do depa dopravce

Řidič doveze vyzvednuté a nerozvezené zásilky do depa dopravce a při jejich vykládání opět načítá čtečkou čárových kódů etikety zásilek. Většina dovezených zásilek v tuto chvíli nemá do informačního systému zaneseny údaje z etikety a proto jsou po vykládce tyto údaje v informačním systému kontrolovány a doplňovány. Přitom se případně může provádět i vážení a měření zásilek. Následuje třídění zásilek podle tak zvaných výstupních směrů. Výstupní směr určuje, na které depo má zásilka dále putovat, a zaměstnanci, kteří provádějí třídění, jej zjistí z informačního systému.

3.4 Přeprava zásilky do výstupního depa

Pokud zásilka není ve výstupním depu, objeví se podle výstupního směru na tak zvané noční nebo systémové soupisce a je podle ní přepravena na další depo. Zásilka tak putuje mezi depy dopravce po předem stanovené trase, kterou pro každou cílovou adresu určuje informační systém. Při každém naložení i vyložení zásilky provádí řidič načtení čárových kódů z etikety zásilky, díky čemuž má dopravce celou dobu přehled o tom, kde se zásilka právě nachází, a v případě ztráty zásilky je také možné dohledat viníka. Služba sledování pohybu zásilky je často zpřístupněna i odesilateli a příjemci přes jednoduchou webovou aplikaci. Takto zásilka putuje až do výstupního depa.

3.5 Doručení zásilky příjemci

Ve výstupním depu je zásilka opět vytištěna na soupisku řidiče, který má zásilku doručit příjemci. Ten při nakládce opět načte čárový kód zásilky a zaveze ji příjemci. Při doručení musí naposledy načíst etiketu zásilky a je upozorněn čtečkou na platbu dobírky nebo převzetí zpětné zásilky. Pokud příjemce zásilku nepřevezme, uloží řidič do čtečky důvod odmítnutí a zaveze zásilku na depo, kde je pak uložena do klece nebo poslána na cestu zpět k odesilateli.

Kapitola 4

Specifikace webové aplikace pro evidenci a tisk zásilek

Specifikace byla vytvářena ve spolupráci s konzultantem diplomové práce a dále upřesňována během implementace jednotlivých částí aplikace. V průběhu práce bylo do specifikace také přidáno několik nových požadavků, což způsobilo, že byl vývoj aplikace inkrementální. V této kapitole se pokusím co nejobecněji popsat všechny požadavky. Nejprve uvedu neformální specifikaci a další detaily, pak objasním na diagramech případů použití (use case diagrams) a diagramu entit a vztahů (entity relationship diagram).

4.1 Neformální specifikace

Aplikace má pomáhat jak dopravci, tak přepravci. Umožnit přepravci tisk etiket v jeho provozních prostorách s možností využití komunikace s jeho informačním systémem tak, aby bylo co možná nejméně ruční práce, a tím i docházelo k vyšší kvalitě práce na úseku přípravy zásilek k přepravě. Důležitou podmínkou k tomu, aby měli dopravci o takovou aplikaci zájem je, že její používání nesmí zatěžovat dopravce a instalace aplikace musí probíhat pokud možno s malou nebo žádnou účastí zaměstnanců dopravce. Jediným úkonem, který nebude probíhat automaticky, by mělo být vytvoření uživatelského účtu pro daného zákazníka. Aplikace by také měla být snadno přizpůsobitelná pro připojení k informačnímu systému libovolného dopravce a musí umět automaticky exportovat vytištěné etikety do tohoto informačního systému.

4.1.1 Obecné zásady a součásti aplikace

Aplikace poběží na serveru dopravce. Budou ji používat zaměstnanci přepravce. Uživatelé budou aplikaci spouštět přes webový prohlížeč. Na straně klienta bude nutná jen automatická instalace některých komponent, například pro tisk etiket. Veškerá data budou uchována na databázovém serveru dopravce. Aplikace bude konfigurovatelná a modulární, aby ji bylo snadné upravit podle potřeb dopravců i přepravců.

Aplikace bude s uživateli komunikovat v českém jazyce. Všechny zobrazované seznamy bude možné filtrovat a řadit podle zvoleného sloupce. Aplikace bude obsahovat tyto součásti:

- evidence zásilek,
- evidence zákazníků,

- import zásilek a zákazníků,
- tisk etiket,
- modul ukončení dne,
- nastavení.

4.1.2 Uživatelé a přihlašování do systému

Aplikace bude rozdělena na dvě části, administrační a uživatelskou. Každý přepravce bude mít jeden přístup do administrační části, kde bude moct vytvářet a upravovat uživatelské přístupy pro svou firmu. Některé firmy mívají více poboček, proto součástí přihlášení do administrační části bude i volba pobočky a administrátor dané firmy bude přihlášen až po výběru pobočky.

Administrátor bude moct upravovat uživatelské přístupy v evidenci uživatelů, která bude umožňovat zadávání nových a editaci a výmaz existujících uživatelů. Každý uživatel bude mít povinně tyto atributy:

- jméno,
- příjmení,
- přihlašovací jméno,
- přihlašovací heslo.

Aby firmy nebyly v přihlašovacích jménech omezeny pouze na uživatelská jména, která ještě nebyla v žádné jiné firmě či pobočce použita, bude součástí přihlášení uživatele i zadání identifikátoru pobočky.

4.1.3 Evidence zákazníků

Evidence zákazníků bude umožňovat vytváření, editaci a výmaz zákazníků. U adresy a PSČ budou stejné možnosti kontroly, jako v evidenci zásilek. Zákazník bude mít tyto atributy:

- ID,
- adresa zákazníka (zdroj pro adresu příjemce):
  - název,
  - ulice + č.p.,
  - město,
  - PSČ,
  - země;
- měna dobírky,
- e-mail,
- poznámka,
- kontaktní osoba,
- telefon,
- telefon pro SMS.

4.1.4 Evidence zásilek

Evidence zásilek bude umožňovat vytváření, editaci a výmaz zásilek. Pokud bude zásilka již odeslaná do informačního systému dopravce, nebude ji již možné ani editovat ani smazat. Pokud bude mít zásilka vytištěnou etiketu, nebude ji už možné editovat.

Aplikace bude umožňovat kontrolu PSČ, pokud dopravce dodá databázi korektních PSČ. Bude možné zadat i PSČ, které v databázi nebude. Takové PSČ bude pro informaci obsluhy zvýrazněno. Zásilky budou mít následující atributy:

- číslo zásilky,
- adresa příjemce:
  - název,
  - ulice + č.p.,
  - město,
  - PSČ,
  - země;
- dobírka,
- měna dobírky,
- variabilní symbol,
- kontaktní osoba,
- telefon,
- telefon pro SMS,
- e-mail,
- datum a čas doručení od,
- čas doručení do,
- datum odeslání dopravci.

Číslo zásilky se bude generovat automaticky z etiketní řady podle typu zásilky až při tisku etikety. Implementované typy zásilek:

- normální balík,
- normální balík - dobírka,
- soukromá adresa,
- soukromá adresa - dobírka,
- expresní balík,
- expresní balík - dobírka,
- export,

- export - dobírka.

Při zadávání zásilek bude možné zadat počet kusů. Po uložení zásilky bude vytvořen zadaný počet stejných zásilek. Pokud bude zadána dobírková částka, zůstane tato pouze u první zásilky, ostatní takto vytvořené zásilky budou mít částku 0 Kč.

Informace o příjemci je možné do zásilky doplnit výběrem zákazníka. Doplní se adresa příjemce, měna dobírky, variabilní symbol, e-mail, kontaktní osoba, telefon a telefon pro SMS. Mezi zásilkou a zákazníkem se neukládá vazba, pouze se zkopírují hodnoty ze zákazníka.

4.1.5 Import zásilek a zákazníků

Aby aplikace mohla spolupracovat i s jinými aplikacemi přepravce, musí být umožněn import zásilek a zákazníků v těchto formátech:

- textový soubor s pevnou šířkou sloupců,
- textový soubor s oddělovači,
- DBF soubor.

4.1.6 Tisk etiket

Tisk etiket bude probíhat z evidence zásilek. Nejprve se vygenerují čísla zásilek z etiketních řad. Bude následovat volba typu tiskárny, přičemž implicitně bude nastavena ta tiskárna, která byla zvolena v nastavení. Pak budou etikety vytištěny a v případě, že se jedná o zpětnou zásilku, bude vytištěna i zpětná etiketa.

4.1.7 Modul ukončení dne

Modul se bude skládat z několika součástí, které bude zákazník obvykle spouštět každý den. Činnosti těchto součástí na sebe budou navazovat.

Přenos zásilek

Po spuštění se zobrazí seznam neodeslaných zásilek z vybraného dne, přičemž implicitně bude zvolen aktuální den, a všechny budou označeny. Zrušením označení bude možné zásilku z odesílaných dat vyjmout. Potvrzením odeslání se všechny označené zásilky odešlou do informačního systému dopravce. U každé odeslané zásilky se pak doplní datum a čas odeslání.

Sestavy

V této části bude zákazníkovi umožněno vytisknout si přejímací doklady zásilek.

Přejímací doklad všech zásilek bude obsahovat informace o všech zásilkách odeslaných do informačního systému dopravce za aktuální den. Přejímací doklad dobírkových zásilek bude obsahovat informace o všech dobírkových zásilkách odeslaných do informačního systému dopravce za aktuální den. Obě sestavy budou obsahovat tyto položky:

- číslo zásilky,
- adresa příjemce:

  - název,
  - ulice + č.p.,
  - město,
  - PSČ,
  - země;
- dobírka,
- variabilní symbol,
- zákaznická reference,
- poznámka.

Pod nimiž bude uvedeno následující shrnutí:

- počet zásilek,
- celková dobírková částka.

Odeslání e-mailů zákazníkům

V tomto kroku bude moct obsluha odeslat e-mail všem zákazníkům přepravce, jejichž zásilky již byly odeslány dopravci. E-mail bude obsahovat čísla všech zásilek a součet plateb za dobírkové zásilky. Odeslání e-mailu bude probíhat přes SMTP server dopravce.

4.1.8 Firemní nastavení

Firemní nastavení bude přístupné pouze pro administrátory. Mělo by umožňovat nastavování následujících položek:

- povolení odesílání e-mailů pro zákazníky,
- povolení importu zásilek,
- nastavení tiskárny pro etikety,
- nastavení hlavní adresy odesílatele:
  - název,
  - ulice + č.p.,
  - PSČ,
  - město,
  - poznámka;
- nastavení adresy odesílatele pro zpětné etikety:
  - název,
  - ulice + č.p.,
  - PSČ,
  - město,

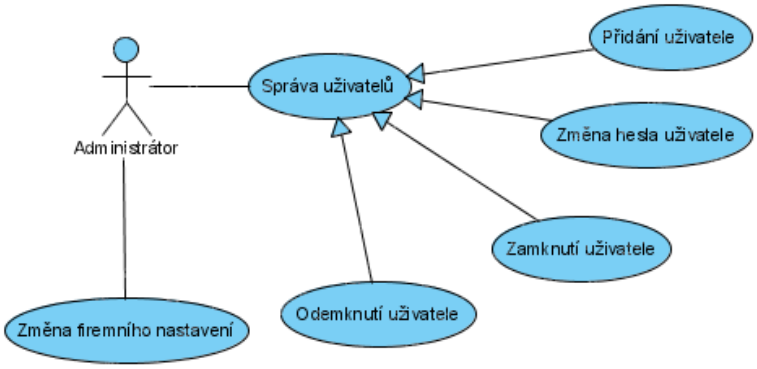
  - poznámka;
- nastavení e-mailu odesilatele,
- změna hesla.

4.2 Diagramy případů použití

Podle uvedené specifikace jsem vytvořil diagramy případů použití, které jsme dále upravovali. Zde uvedu pouze finální podobu, která je také jediná, která byla vytvořena i v digitální podobě a zachovala se.

4.2.1 Administrační část aplikace

Bylo dohodnuto, že administrační část bude v této verzi aplikace zcela oddělena od části uživatelské, aby zatím nebylo nutné v žádné z částí vytvářet infrastrukturu pro vytváření uživatelských rolí. Toto rozhodnutí bylo učiněno ve snaze usnadnit vývoj aplikace, protože ještě není jisté, zda bude někdy nasazena do skutečného provozu. Proto jsou také případy použití administrační části aplikace zcela odděleny od případů použití uživatelské části aplikace.



Obrázek 4.1: Diagram případů použití administrátora

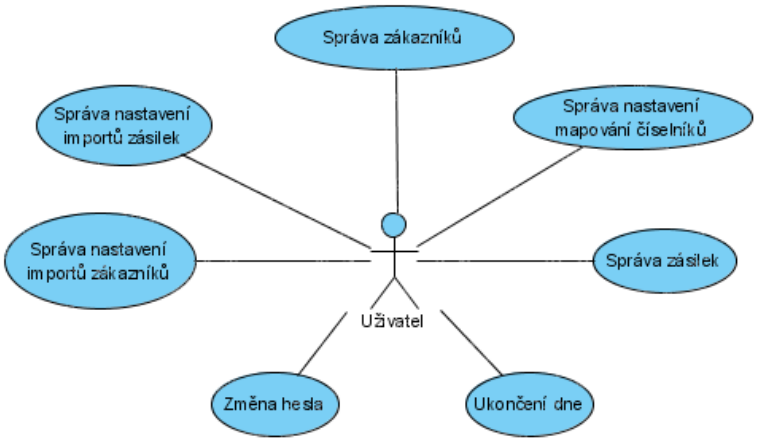
Obrázek 4.1 ukazuje diagram případů použití administrátora. Pokud je ve firemním nastavení povoleno odesílání E-mailů zákazníkům, musí případ použití „Změna firemního nastavení“ ještě obsahovat kontrolu, zda je zadána E-mailová adresa odesilatele. Ve správě uživatelů bude pro úplnost za jménem uživatele vždy vypsána i identifikace firmy.

4.2.2 Uživatelská část aplikace

Případy použití uživatelské části aplikace jsem pro lepší orientaci dekomponoval do několika diagramů. Základním je diagram obsahující i aktéra „Uživatel“ na obrázku 4.2. Všechny případy použití kromě případu použití „Změna hesla“ budou specifikovány v dále.

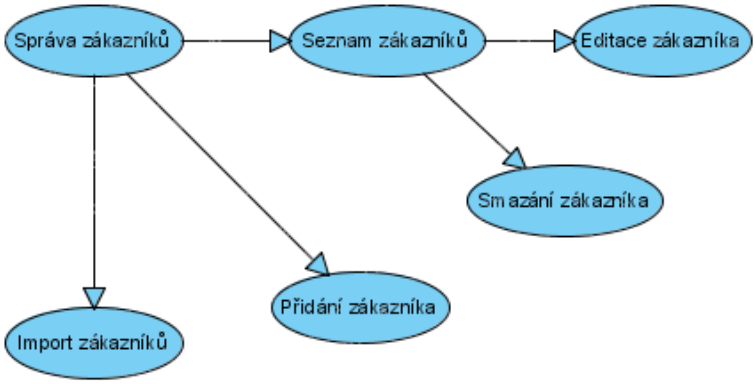
Případ použití „Správa zákazníků“

Případ použití „Správa zákazníků“ v sobě nenese žádná úskalí a jeho dekompozici ukazuje obrázek 4.3. Je potřeba dbát na to, že podle obecných pravidel neformální specifikace musí



Obrázek 4.2: Diagram případů použití uživatele

seznam zákazníků také umožňovat řazení a filtrování položek.



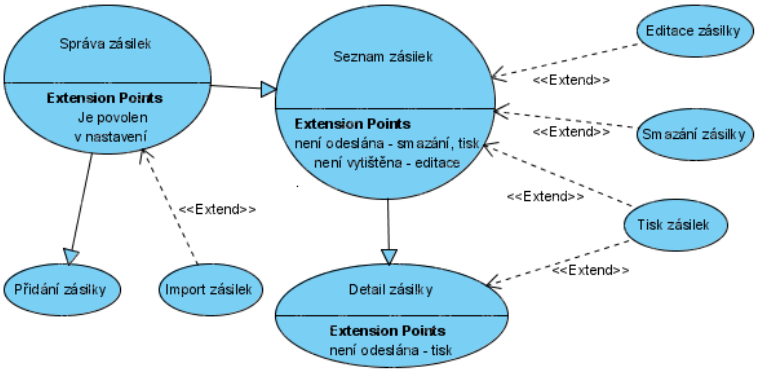
Obrázek 4.3: Dekompozice případu použití „Správa zákazníků“

Případ použití „Správa zásilek“

Dekompozice případu použití „Správa zásilek“ je komplikovanější, což dokládá i obrázek 4.4. Aby bylo možné importovat zásilky, musí to být povoleno v nastavení. Editace zásilek je u vytištěných zásilek zakázána. Smazání zásilky je zakázáno u odeslaných zásilek. Stejně tak je u odeslaných zásilek zakázán tisk, kterým je myšleno tisk etikety zásilky.

Případ použití „Správa nastavení mapování číselníků“

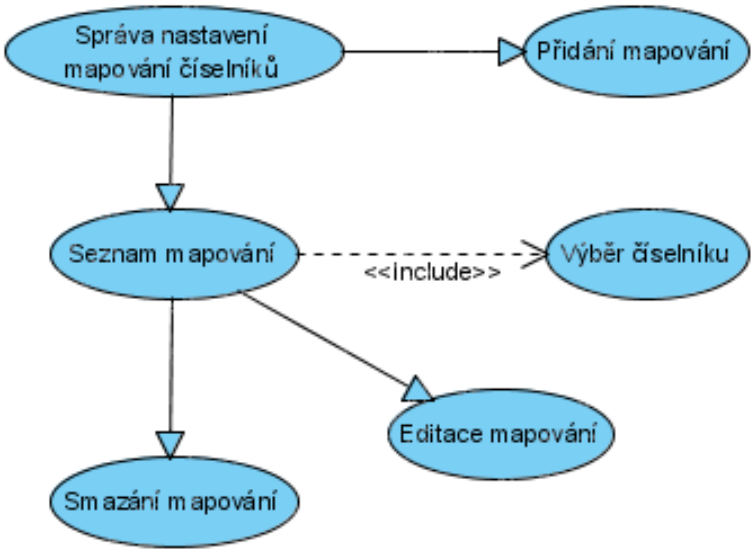
Dosud jsem se ještě nezmínil o mapování číselníků. Na problém mapování číselníků jsem narazil až při návrhu tříd pro import zákazníků a zásilek. Mapování číselníků řeší problém, že v databázi přepravce může být jiná reprezentace významově stejné hodnoty číselníku než v databázi dopravce. Například stát Česká republika v číselníku států může být u dopravce reprezentován hodnotou `CZ` a u přepravce hodnotou `ČR`. Prvním řešením, které se nabízelo bylo donutit přepravce, aby si výstupy upravil tak, ať vyhovují, což je pravděpodobně časté



Obrázek 4.4: Dekompozice případu použití „Správa zásilek“

řešení tohoto problému. Já jsem se rozhodl přidělat přepravci co možná nejméně práce a vyjít mu vstříc, proto jsem vytvořil mapování číselníku. Mapování číselníku tedy umožňuje mapovat hodnoty databáze přepravce na význam v databázi dopravce. Přepravce při tom není zatěžován ani hodnotou v databázi dopravce a je mu zobrazována přímo její textová reprezentace.

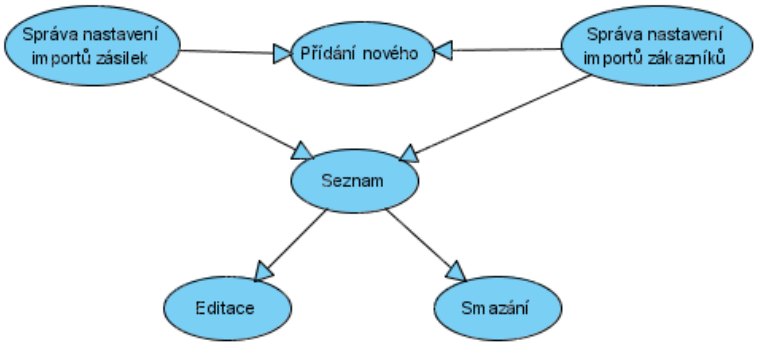
Dekompozice případu použití „Správa nastavení mapování číselníků“ je na obrázku 4.5. Nastavení mapování jsou rozdělena podle číselníků, ke kterým náleží a pro každý číselník může být více nastavení. Uživatel si pak při nastavování importu může vybrat, které mapování pro který sloupec použije.



Obrázek 4.5: Dekompozice případu použití „Správa nastavení mapování číselníků“

Případy použití „Správa nastavení importu zásilek“ a „Správa nastavení importu zákazníků“

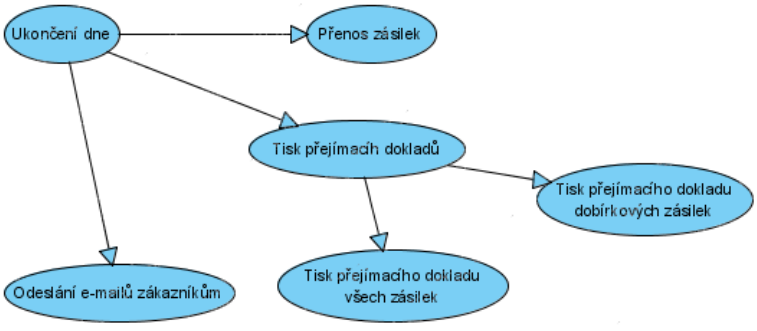
Pro případy použití „Správa nastavení importu zásilek“ a „Správa nastavení importu zákazníků“ jsem vytvořil společnou dekompozici, která je na obrázku 4.6. Případy použití „Přidání nového“ a „Editace“ musí obsahovat také výběr typu importu a jeho nastavení. Nastavení může být uloženo více a při importu si uživatel může vybrat, které nastavení použije. Nastavení bude obsahovat také výběr uložených mapování číselníku pro sloupce, u kterých je to potřeba.



Obrázek 4.6: Dekompozice případů použití „Správa nastavení importu zásilek“ a „Správa nastavení importu zákazníků“

Případ použití „Ukončení dne“

Případ použití „Ukončení dne“ měl být původně proveden ve stylu průvodce (wizard). Nakonec bylo rozhodnuto, že bude lepší, když budou jednotlivé akce na sobě nezávislé a uživatel je bude moct vykonávat i v jiném pořadí, než je doporučeno v návodu. Dekompozice tohoto případu použití je na obrázku 4.7.



Obrázek 4.7: Dekompozice případu použití „Ukončení dne“

4.3 Diagram entit a vztahů

Diagram entit a vztahů, který teď budu popisovat, naleznete v dodatku B. Diagram přesně odpovídá databázi aplikace a entity v něm jsou rozmístěny tak, aby byly související entity blízko sebe. Nyní se pokusím objasnit význam vlastností a entit, u kterých není zřejmý.

Pro některé entity jsem použil české názvy, protože jsou snadněji zapamatovatelné. Nejsem si však jistý, zda to bylo dobré řešení.

4.3.1 Vlastnosti používané ve více entitách

Následující vlastnosti mají stejný význam pro všechny entity, které je obsahují. Pokud je vlastnost LOCKED nastavena na true (1), znamená to, že byla daná entita zablokována. Pokud je nastavena na true (1) vlastnost REJECT znamená to, že je daná entita zakázána. Vlastnost CHYBY obsahuje výčet chyb, ke kterým došlo při importu entity ze souboru.

4.3.2 Entita FIRMS

Entita FIRMS představuje pobočku firmy. Je to jen ukázka, jak může vypadat složitější reprezentace pobočky pomocí dvou parametrů:

- CUSTOMER_ID nesoucí číslo zákazníka dopravce (tedy přepravce) a
- CUST_DEP_ID obsahující číslo depa, pod které přepravce spadá.

Vzhledem k tomu, že data pro jednotlivé zákazníky dopravce musí být oddělena, má téměř každá entita v diagramu cizí klíč FIR_ID odkazující se na tuto entitu.

4.3.3 Entita POVOLENE_MENY

Entita POVOLENE_MENY nese informaci o tom, které měny jsou povolené pro dobírky pro danou pobočku. Tato entita byla zavedena proto, že se ukázalo, že někteří dopravci podmiňují používání zahraničních měn pro dobírky určitými pravidly. Tato entita tedy slouží k tomu, aby mohla být synchronizována povolení mezi databází dopravce a databází aplikace pro tisk etiket.

4.3.4 Entita NASTAVENI

Entita NASTAVENI slouží k ukládání nastavení pro jednotlivé pobočky. Vlastnosti s prefixem POVOLEN umožňují zakázat danou funkčnost. Konkrétně odesílání informačních E-mailů příjemcům zásilek a importování zásilek ze souboru. Vlastnosti s prefixem DATAMAX se používají pro nastavení tiskáren Datamax. Prefix HL znamená hlavní adresa. Ta bude na etiketách použita jako adresa odesilatele. Prefix ZP znamená zpětná adresa a ta bude vytištěna jako adresa příjemce na tak zvaných zpětných zásilkách. Zpětná zásilka je přibalena k odesílané zásilce a po doručení příjemci je příjemcem odeslána na zvolenou zpětnou adresu[1]. Vlastnost EMAIL_ODESILATELE je vyplněna při odesílání informačních E-mailů jako adresa odesilatele. Tato vlastnost musí být vyplněna, pokud je povoleno odesílání informačních E-mailů.

[1] většinou se jedná o fakturu, kterou příjemce před odesláním podepíše

4.3.5 Entita ZASILKY

Entita ZASILKY uchovává informace o připravovaných i vytištěných zásilkách. Než vysvětlím význam vlastností této entity, musím vysvětlit operace se zásilkami, které byly po aplikaci vyžadovány.

Vytváření zásilek mělo umožnit vytvoření několika stejných zásilek najednou. Tyto zásilky měly být samostatně editovatelné. V případě, že se jedná o zásilku s dobírkou, měly být i svázány dohromady jako jeden celek, aby bylo možné při tisku etiket vytisknout i o kolikátou zásilku z jakého celkového množství se jedná. Zatím uvedené podmínky by ukazovaly na řešení vytvořit entitě vlastnost počet kusů a v případě jakékoli změny jednoho z kusů jej z této sady vyjmout. Problém je však v tom, že určité změny jsou pro jednotlivé zásilky ze sady povoleny, aniž by došlo k vyloučení ze sady. Konkrétně dojde k vyloučení ze sady, pokud se změní variabilní symbol, ulice, město nebo PSČ příjemce a v ostatních případech zůstane zásilka v sadě. Tím pádem bylo nutné zásilky svázat na úrovni databáze, což jsem provedl přidáním vlastnosti MASTER_ID, která je cizím klíčem entity ZASILKY a odkazuje se na první zásilku ze sady. Entita obsahuje ještě jeden odkaz na sebe a tím je ZPETNA_OD_ID. Tento odkaz se používá pouze u zpětných zásilek a odkazuje se na zásilku, do které má být zpětná zásilka přibalena. Jedná se o vztah 1:1 (zásilka nesmí obsahovat více než jednu zpětnou zásilku). Umístění odkazu do zpětné zásilky jsem volil proto, abych mohl zadat pravidlo na smazání zpětné zásilky zároveň se zásilkou, ke které měla být připojena.

Vlastnost CODE_TYPE nese typ zásilky vybraný z číselníku TYP_ZASILKY. Vlastnost ZAKAZNIK_ID je pro budoucí použití a momentálně se pouze odkazuje na entitu ZAKAZNIK, která byla použita jako šablona pro tuto zásilku. Vlastnost EMAIL obsahuje adresu elektronické pošty, na kterou bude doručen informační E-mail v případě, že obsluha po odeslání zásilky do databáze dopravce tuto volbu vybere. Vlastnosti DATUM_DOR_OD a DATUM_DOR_DO slouží pro vymezení dne a denní doby, ve které si přepravce přeje, aby byla zásilka doručena. Datum obou vlastností musí být shodné a odpovídat dopravcem stanoveným pravidlům. Čas uložený v těchto vlastnostech pak stanovuje časové rozmezí doručení zásilky ve vybraném dni. Vlastnost VYTISKNUTA se nastavuje na true (1) co nejpozději před vytištěním a vlastnost DATUM_ODS_DB se nastavuje na čas odeslání zásilky do databáze dopravce.

4.3.6 Entita CISELNE_RADY

Entita CISELNE_RADY umožňuje využití zaběhnutého systému pro přidělování čísel zásilek, které používá dopravce pro identifikaci zásilek. Ukládají se do ní informace o řadách těchto čísel zásilek ve formě použitelné pro snadné automatické doplňování ze systému dopravce. Tato entita se bude u různých dopravců lišit svým obsahem, podle toho, jakým způsobem dopravce řady přiděluje. K přidělování čísel těchto řad pak bude sloužit databázová procedura volaná z aplikace, která má přesně dané rozhraní.

V tomto případě obsahuje entita identifikaci pobočky tak, jak ji používá dopravce (vlastnosti CUSTOMER_ID a CUST_DEP_ID), typ zásilky, ke kterému se vztahuje (vlastnost CODE_TYPE), rozsah čísel v řadě (od čísla zadaného v PACKAGE_ID_FROM do čísla v PACKAGE_ID_TO) a poslední použité číslo zásilky (vlastnost LAST_USED).

4.3.7 Entita CISELNIKY_META

Entita CISELNIKY_META obsahuje metainformace o číselnících. Vytvořil jsem ji pro usnadnění práce s mapováním číselníků a hlavně pro umístění všech potřebných informací o číselnících na jedno místo, což usnadňuje jejich modifikaci. Každý číselník má svůj řádek v této entitě.

Vlastnost NAZEV obsahuje název, který je používán v aplikaci pro identifikaci číselníku. Vlastnost NAZEV_TABULKY obsahuje název tabulky v databázi, ve které jsou hodnoty číselníku uloženy. Vlastnost ZOBRAZOVANY_NAZEV obsahuje název číselníku, který je v aplikaci zobrazován uživateli. Vlastnosti SLOUPEC_ID a SLOUPEC_HODNOTA se odkazují na názvy sloupců v tabulce číselníku. Přičemž SLOUPEC_ID obsahuje název sloupce s primárním klíčem (například „CZ“) a SLOUPEC_HODNOTA obsahuje název sloupce s textovou reprezentací významu této hodnoty (například „Česká Republika“). Vlastnosti HAS_LOCKED a HAS_REJECT jsou nastaveny na true (1) u těch číselníků, které obsahují ve svých tabulkách i sloupec LOCKED respektive REJECT s významem, který jsem popisoval v oddíle 4.3.1.

4.3.8 Entita MAPOVANI_CISELNIKU a s ní související

Entita MAPOVANI_CISELNIKU slouží k ukládání mapování číselníků. Vlastnost CIS_META_ID se odkazuje na metainformace k číselníku, který mapuje. Vlastnost NAZEV obsahuje zobrazovaný název mapování. Vlastnost POPIS může obsahovat uživatelský popis mapování. Jednotlivé prvky mapování jsou pak uloženy v entitě PRVKY_MAPOVANI_CISELNIKU. V té jsou pouze tyto vlastnosti:

- HODNOTA_VE_ZDROJI_DAT, obsahující hodnotu v importovaném souboru,
- HODNOTA_V_DATABAZI, obsahující hodnotu klíče číselníku v databázi aplikace, a
- MAPOVANI_CISELNIKU_ID odkazující se na MAPOVANI_CISELNIKU.

4.3.9 Entity IMPORT_ZASILEK a IMPORT_ZAKAZNIKU

Entity IMPORT_ZASILEK a IMPORT_ZAKAZNIKU obsahují nastavení importu zásilek, respektive zákazníků. Jak jsem již uvedl, každá pobočka může mít více těchto nastavení. Pokud je vlastnost IMPORT_S_CHYBOU natavena na true (1), jsou importovány i řádky, při jejichž importu došlo k nějaké chybě a do sloupce CHYBA je u nich uložen popis vzniklé chyby. Vlastnosti s postfixem CISLO_SLOUPCE obsahují číslo sloupce v importovaném souboru, ze kterého se má načíst hodnota do sloupce v cílové tabulce (ZASILKA, respektive ZAKAZNIK). Hodnota se načítá do sloupce se stejným názvem, jako je začátek názvu vlastnosti před zmíněným postfixem. Obsah vlastnosti s prefixem DEFAULT je do tohoto sloupce cílové tabulky uložen, pokud dojde při jeho překladu k chybě, na vstupu chybí nebo pro něj nebyla definována vlastnost CISLO_SLOUPCE (je nastavena na -1). Vlastnosti s postfixem MAPOVANI_CISELNIKU_ID jsou pouze pro sloupce obsahující hodnotu některého číselníku a odkazují na mapování, které se má při importu použít.

Z posledních tří vlastností IMPORT_DBF_ID, IMPORT_S_ODDELOVACI_ID a IMPORT_S_PEVNOU_SIRKOU_SLOUPCE_ID musí mít hodnotu jinou než null právě jedna.

4.3.10 Entity určující typ importu

IMPORT_DBF jen označuje, že se bude importovat ze souboru DBF a nemá zatím žádné používané vlastnosti. IMPORT_S_ODDELOVACI a IMPORT_S_PEVNOU_SIRKOU_SLOUPCE se používají pro importy z textových souborů. Oba obsahují

- vlastnost ENCODING, do které se ukládá identifikátor kódování používaný v .NET,
- vlastnost ODDELOVAC_RADKU, která říká, jakým znakem jsou odděleny řádky ve vstupním souboru (obvykle znak nového řádku),
- vlastnost OREZAT_BILE_ZNAKY, kterou je potřeba nastavit na true (1), pokud mají být po načtení sloupce ze začátku a konce textu, který obsahuje, ořezány bílé znaky a
- vlastnost VYNECHAT_PRVNI_RADEK, která, nastavena na true (1), způsobí vynechání prvního řádku, což je výhodné, pokud první řádek například obsahuje názvy sloupců.

IMPORT_S_ODDELOVACI má pak ještě

- vlastnost ODDELOVAC_SLOUPCU, která obsahuje znak, použitý ve vstupním souboru pro oddělení sloupců,
- vlastnost UNIKOVY_ZNAK, ve které může být uveden znak, který ruší význam speciálního znaku za ním a vloží ho na výstup (aby mohl být na výstup vložen únikový znak, musí být na vstupu dvakrát za sebou),
- vlastnost UVOZOVACI_ZNAK, který, pokud je zadán, není předáván na výstup, ale jakékoli znaky, které jsou mezi dvěma uvozovacími znaky, jsou předány na výstup včetně případného uvozovacího znaku a
- vlastnost DVE_UVOZOVKY_JAKO_JEDNA, kterou je možné nastavit na true (1), což způsobí, že dvě za sebou následující uvozovky budou na výstup vypsány jako jedna a nebudou mít svůj speciální význam.

IMPORT_S_PEVNOU_SIRKOU_SLOUCPE obsahuje navíc

- vlastnost SIRKA_SLOUPCE_DEFAULT a SIRKA_MEZERY_DEFAULT, které definují šířku sloupce a mezery za ním ve vstupním souboru pro všechny sloupce, jejichž šířka a mezera není definována,
- vlastnost POVOLIT_KONEC_RADKU_VE_SLOUPCI, která musí být nastavena na true (1), pokud má být umožněno importovat ze souboru, ve kterém je znak konce řádku uvnitř některého ze sloupců.

Pro IMPORT_S_PEVNOU_SIRKOU_SLOUCPE bude také často definováno několik entit IMPORT_SIRKY_SLOUPCU obsahující informaci o šířkách sloupce a mezery pro sloupce, jejichž parametry neodpovídají implicitním.

Více o významu těchto položek i s příklady je možné nalézt v dodatku A.

4.3.11 Číselníky

- Vlastnost EU v číselníku ZEME je určena pro budoucí použití.
- Vlastnost COD v číselníku MENY určuje, zda daná měna může být povolena pro dobírku.
- Vlastnost COD v číselníku TYP_ZASILKY určuje, zda je daný typ zásilky dobírkový, to znamená, zda má být při předání příjemci požadována dobírka.
- Vlastnost OUTLAND v témže číselníku označuje, zda je daný typ zásilky určen pro přepravu do zahraničí (hodnota true (1)) nebo zda slouží pouze pro vnitrostátní přepravu.
- Vlastnost POST v číselníku PSC obsahuje název pošty a je určena pro budoucí použití.
- Číselník PORTY obsahuje seznam portů podporovaných pro tisk.
- Číselník TISKARNY obsahuje seznam tiskáren podporovaných pro tisk, přičemž tiskárny s ACTIVEX nastaveným na true (1) budou obslouženy ACTIVEX prvkem na stránce.

Kapitola 5

Implementace a hodnocení použitých nástrojů

Vzhledem k tomu, že jsem před psaním této diplomové práce neznal ASP .NET 2.0, nepouštěl jsem se předem do návrhu tříd uživatelského rozhraní a vytvářel jsem nejprve perzistentní třídy. Pro rozhraní s databází a jako základ persistentních tříd jsem musel použít knihovnu libtools, která je licencovaná, a proto není součástí odevzdaného zdrojového kódu. V průběhu vytváření persistentních tříd jsem se postupně učil práci s ASP. Když už byla většina persistentních tříd hotova a měl jsem nastudovány základy ASP, začal jsem hledat možnosti autentizace a autorizace v ASP. Rozhodl jsem se držet se standardního přístupu k autorizaci a vytvořit vlastní implementaci standardního rozhraní autentizace, aby bylo možné mít v databázi uložené uživatele libovolným způsobem. Jakmile bylo jádro autentizačního modulu hotovo, implementoval jsem třídy sloužící k importu dat z textových souborů a souborů *DBASE 4*. Pak jsem začal postupně vytvářet uživatelské rozhraní podle připravených diagramů případů použití a při tom jsem pokračoval v implementaci ostatních součástí. Vytvořil jsem několik vlastních serverových komponent, které mě sice stály mnoho sil, ale do budoucna už budu vědět, jak na to, a vytváření dalších komponent mi bude šetřit práci. Tento způsob vývoje aplikace jsem zvolil hlavně proto, abych mohl pružně reagovat na změny požadavků na aplikaci, které nebyly zpočátku zcela přesně specifikovány.

V následujících podkapitolách se budu zabývat podrobněji jednotlivými fázemi implementace. Vzhledem ke komplikovanosti problematiky počítám, že má čtenář alespoň základní znalosti ASP, pokud chce všem popisovaným věcem porozumět. Snažil jsem se však, aby i neznalý problematiky získal alespoň nějakou představu o ASP.

5.1 Knihovna libtools

Naučit se dělat s pomocí této knihovny základní operace bylo jednoduché. Hned při prvním seznámení se mi nelíbilo, že knihovna nepoužívá atributy, čímž vzniká redundance kódu, protože názvy sloupců v databázi musíte zadávat na více místech nebo si pro ně vytvořit proměnnou a tu pak všude používat. Tvůrce knihovny mi sdělil, že atributy nejsou používány záměrně, aby knihovna pracovala rychleji, protože zjišťování atributů je pomalé. S tím jsem nesouhlasil, protože se jednak údaje z metadat mohly klidně načítat pouze jednou a uložit do vhodné, rychle pracující struktury, a jednak i kdyby se toto neudělalo, .NET si zjištěné atributy automaticky ukládá na určitou dobu do cache. V případě webové aplikace by se jen těžko mohlo stát, že by byla tato cache vyprázdněna. Nakonec jsem se

rozhodnul s tímto nedostatkem nic nedělat, protože vytvoření obalu tříd, který by umožnil použití atributů, by mi zabralo více času než poprat se s redundancí.

Postupně jsem libtools při vytváření tříd zkoumal hlouběji a zjistil jsem, že nemá podporu transakcí takovou, jak bych si představoval a nepracuje nejlépe s identitama[1]. Vytvořil jsem proto obal původních tříd, který umožnil automatické nahrávání primárních klíčů do nově uložených objektů a používání transakcí tak, aby bylo možné zavoláním jedné metody provést změnu ve více tabulkách a nemuset se při tom starat o transakce. Zároveň obal umožňuje, aby byla transakce pro práci s databází nastavena jen jednou, a všechny metody pracující s databází ji používají, dokud není odpojena nebo ukončena.

Později jsem zjistil, že je na tom tato knivona poměrně špatně i co se týče výkonnosti, protože pro načítání dat z databáze nepoužívá `DbDataReader`, který je v C# nejryhlejší v přístupu k databázi, ale `DbDataAdapter` v kombinaci s `DataTable`, což je mnohem náročnější než používat `DbDataReader` a podle mého názoru použití `DbDataAdapter` ani v případě implementace perzistentní třídy nedává smysl. Vysvětlím proč. O tom, že `DbDataReader` je rychlejší svědčí i to, že `DbDataAdapter` podle všeho používá na provedení dotazu `DbDataReader`. Navíc `DbDataAdapter` vytváří `DataTable`, což je tabulka přijatých záznamů. Libtools pak pracují tak, že používají zpožděné načítání dat z takto vytvořené `DataTable`, i když by se data mohla místo kopírování do `DataTable` nahrát z `DbDataReader`u přímo do pezistentních objektů. Tento problém se však týká pouze načítání záznamů, s tím, jak je implementován `insert`, `update` a `delete`, souhlasím. Jen si myslím, že když už se používá `DataTable` pro načítání dat z databáze, mohl si tvůrce možná ušetřit trochu práce, kdyby ji použil i pro ukládání a změny v databázi. Nemusel by pak vytvářet vlastní třídu pro sestavování SQL dotazů, protože toto `DbDataAdapter` v kombinaci s `DataTable` zvládá.

S knihovnou libtools jsem tedy nebyl moc spokojen, ale kvůli tomu, jak je implementována, a ne kvůli tomu, že by šlo o nějaký nedostatek .NETu.

5.2 Autentizace a autorizace

Jak jsem se již zmiňoval, problém autorizace byl eliminován tím, že zatím nejsou požadovány žádné role pro uživatele a aplikace je fyzicky rozdělena na dvě části. Autorizace tím pádem spočívá pouze v ověření, zda je uživatel přihlášen. Pokud je při psaní webové aplikace v ASP použit jeden z vestavěných autentizačních modulů nebo se ze základní abstraktní třídy `MembershipProvider` odvodí vlastní varianta používající k přihlášení pouze jméno a heslo uživatele, ASP programátorovi ušetří práci. Lze pak použít všechny stadartní komponenty, jako je například

- `Login` (slouží pro přihlášení) a
- `LoginName` (zobrazuje jméno přihlášeného uživatele)

a také nastavování přístupu k jednotlivým stránkám přímo v konfiguračním souboru `Web.config`. Další výhodou z tohoto přístupu ja také to, že ho zná snad každý, kdo vyvíjí apliakce v ASP a není pak problém se v aplikaci vyznat.

Já jsem však potřeboval umožnit i přihlašování nejen pomocí uživatelského jména a hesla, ale i pomocí identifikace pobočky. Také jsem se chtěl vyhnout používání sessions, protože kladou podle mého názoru nemalé nároky na webový server. Pro ten je mnohem

[1] tento název se v MSSQL používá pro primární klíč s automatickou inkrementací

snažší používat pro ověřování autorizace šifrované cookies. Ty v kombinaci s šifrovaným spojením přinášejí podle mého názoru dostatečnou úroveň zabezpečení. V ASP je standardně využívána autorizace pomocí cookies, které umožňují vložit kromě jména a platnosti přihlášení do šifrované cookie i další textový údaj. Bohužel ačkoli třída pro správu těchto cookies `FormsAuthenticationTicket` umožňuje používání uživatelských dat v cookies, abstraktní modul `MembershipProvider` ani komponenta `Login` s nimi neumí pracovat. Na ostatní funkčnost však jejich použití nemá vliv. Musel jsem tedy kromě implementace a rošíření `MembershipProvider` rozšířit také komponentu `Login`, aby bylo vše funkční. Svou základní třídu pro autentizaci a správu uživatelů jsem nazval `ExtMembershipProvider` a odvodil z ní jednu další pro každou část aplikace. Moje třída používá jako sklad informací o uživatelích databázi, kde je přístup ke všemu prováděn prostřednictvím procedur. Názvy procedur a jejich parametrů jsou definovány ve `Web.config`, takže by mělo být snadné třídu použít kdekoli, kde jsou informace o uživatelích v databázi. V dodatku B je také seznam procedur, ve kterém všechny procedury začínající `UA` (zkratka pro user authentication) jsou ve `Web.config` napojené na `ExtMembershipProvider`. Do uživatelských dat pak ukládám primární klíč pobočky a uživatele pro usnadnění přístupu k databázi. Primární klíče se však v uživatelském rozhraní nikde nezobrazují.

5.3 Tvorba uživatelského rozhraní

Tvorba uživatelského rozhraní v ASP je poměrně snadná. ASP umožňuje vytvořit takzvanou *master page* (vzorovou stránku), která slouží jako šablona pro ostatní stránky (jsou v ní vyznačena místa pro obsah, na které se pak jednotlivé stránky odkazují). Úpravu vzhledu zase usnadňují *templates* (šablony). Výběr komponent uživatelského rozhraní je sice bohatý, ale přesto jsem nakonec dvě vlastní komponenty vytvořil. První byla jen jednoduchým rozšířením `DropDownList` umožňující navíc automatické přidání prvků, které jsem nazval `null` (reprezentuje hodnotu `null` v databázi) a `delete` (indikuje, že má být položka smazána). Druhou komponentou byla dynamická tabulka.

5.3.1 DynamicTable

Komponentu jsem nazval `DynamicTable`. Vytvářel jsem ji kvůli snazšímu a rychlejšímu zadávání hodnot do mapování číselníku a nastavení importu s pevnou šířkou sloupce. Tabulka slouží pro hromadnou editaci hodnot, proto nemá moc smysl v ní zobrazovat čistý needitovatelný text. V aplikaci tedy obsahuje výhradně pole pro zadávání textu (která jsou po načtení zaplněna aktuálními hodnotami) a roletová menu. Poslední řádek této tabulky je vždy prázdný a když dostane focus, je pomocí javascriptu zkopírován a vytvořen znova jako další řádek tabulky. Při kopírování se upravují identifikátory jednotlivých prvků, aby bylo ASP schopno při post-backu tyto prvky rekonstruovat. Říkám rekonstruovat i když tyto prvky při vytváření stránky neexistovaly a to zcela záměrně. Chci tím zdůraznit to, že tabulka pracuje tak, aby si ASP myslelo, že prvky na stránce už byly, protože pokud by tomu tak nebylo, stránka by hlásila chybu, protože by nechtěla načíst data do komponent, které nevytvořila. Aby se toto dalo provést musel jsem pro dynamickou tabulku vytvořit lehce upravené komponenty, které potlačují hlášení o tom, že původně na stránce nebyly. Je to sice obcházení bezpečnostního pravidla, ale pokud se tyto komponenty budou používat pouze v `DynamicTable` nebezpečí nehrozí.

5.3.2 AJAX

Při vytváření `DynamicTable` jsem použil několik funkcí z AJAXové knihovny vytvořené pro .NET, ale ty neměly s AJAXem nic spolecného. Komponenty z `AjaxControlToolkit` jsem začal vkládat až ke konci, kdy bylo hotové vše potřebné pro chod aplikace a mi už nezbývalo moc času, takže jich je jen pár a ani jedna z nich nevyužívá asynchronní přístup. Je škoda, že mi to nevyšlo, protože jsem si o principech a využití AJAXu stihl hodně načíst a vůbec jsem to nepoužil. Pravděpodobně se pokusím přidat ještě alespoň `UpdatePanel` do formuláře pro editaci a vytváření zásilek. Pokud se ukáže, že tato verze má své uplatnění a bude se dále vyvíjet, určitě se bude jedním z mých cílů vytvořit co nejvíce prvků využívajících AJAX, protože zvyšují interaktivitu aplikace a usnadňují její používání.

5.4 Hodnocení ASP podle nabytých zkušeností

Myslím, že musí být pro každého snadné naučit se používat připravené komponenty a infrastrukturu ASP. Pokud však chcete vytvořit vlastní komponenty a vlastní infrastrukturu postavenou na té, co je v ASP, musíte pochopit mnoho vnitřních procesů. Po zkušenostech s programováním aplikací pro Windows jsem myslel, že tvorba komponent pro ASP bude snazší. Byla to naivní představa. Opomenul jsem totiž to, že model klient-server je složitější než model běžné aplikace. V ASP je situace také komplikována tím, že se komponenty snaží tvářit, jakoby k žádné komunikaci mezi klientem a serverem nedocházelo, a to prostřednictvím zmíněného *post-back* modelu a *viewstate* mechanizmu. *Viewstate* umožňuje při *post-backu* rekonstruovat původní komponenty, které pak podle toho vyvolávají události a informují o změnách svého obsahu stejně jako ovládací prvky normální aplikace.

MSSQL je pro ASP .NET určitě nejlepší volba, protože oba produkty jsou vytvářeny tak, aby spolu dobře fungovaly.

5.5 Zdrojový kód a licence

V dodatku C jsou uloženy všechny soubory se zdrojovými kódy, které jsem vytvořil. Jen několik souborů v adresáři `ISBussines` není moje práce. Při implementaci však byla použita knihovna ComponentOne a libtools, které jsou licencované a nemohl jsem je proto přiložit. Z tohoto důvodu je přiložený zdrojový kód nepřeložitelný. Také není přiložen rozpracovaný prvek ActiveX, sloužící k tisku na speciálních tiskárnách, který jsem také nevytvářel a navíc ještě není kompletní. Funkční aplikace bude nejméně do obhajoby této diplomové práce k dispozici na internetu. Její administrační část bude na URL `http://www.digipartners.cz/TiskEtiketAdmin/` a uživatelská část na URL `http://www.digipartners.cz/TiskEtiket/`. Přihlašovací údaje jsou k dispozici na přiloženém CD.

Kapitola 6

Závěr

Zpočátku jsem nevěřil svému konzultantovi, že tato aplikace je pro programátora neznalého ASP prací na dva a půl měsíce na plný úvazek. Nakonec se ukázalo, že nebyl daleko od pravdy. Vytvořit aplikaci bylo tedy náročnější, než jsem očekával, a proto jsem nestihl udělat uživatelské rozhraní podle svých původních představ. Aplikace je však funkční a splňuje všechny požadavky. Chybí pouze dokončit ActiveX prvek sloužící pro výstup na tiskárny etiket, to však nebyl můj úkol.

Nenapadají mě žádné další funkce, které by bylo rozumné do aplikace, vzhledem k jejímu účelu, přidat. Bylo by však vhodné vylepšit uživatelské rozhraní, optimalizovat perzistentní třídy (pokud možno nepoužívat libtools) a upravit aplikaci tak, aby velké množství zásilek příliš nezatěžovalo server (zatím se počítá s nasazením pouze pro malé přepravce, pokud mezi nimi bude zájem).

Aplikace ještě nebyla nasazena do zkušebního provozu. Moje práce je již sice hotova, bohužel však chybí zmíněný ActiveX. Druhým problémem je, že se zkušebním provozem musí souhlasit především dopravce. Ten zatím pouze přislíbil, že se chce na tomto podílet, ale aplikaci ještě pro zkušební provoz neschválil.

Dalším cílem této práce bylo, vytvořit stručný přehled technologií, jazyků a frameworků používaných pro vývoj webových aplikací. Tento úkol mě velmi obohatil, ale vzhledem k požadovanému rozsahu této části jsem se mu již nechtěl v textu více věnovat. Po vyzkoušení práce se třemi webovými frameworky se ukázalo, že je složitější se naučit pracovat s JSF a ASP než s frameworkem Symphony. JSF a ASP však přinášejí výhody snazšího vývoje aplikací pro ty, kteří je již znají. Při zkoušení JSF jsem narazil na problémy se serverem Tomcat, který občas přestal reagovat nebo se v něm neprojevily provedené změny a musel jsem jej restartovat. To však mohlo být nějakou chybou v konfiguraci. U ASP jsem žádné problémy se serverem neměl.

Literatura

[1] *About Ruby*. [online], Dokument dostupný na URL http://www.ruby-lang.org/en/about/ (29. prosince 2008).

[2] *China's Postal Service: A Brief History*. [online], Dokument dostupný na URL http://www.chinapost.com.cn/upuxx/cywl/eng/2.htm (3. ledna 2008).

[3] *The Common Gateway Interface*. [online], Dokument dostupný na URL http://hoohoo.ncsa.uiuc.edu/cgi/overview.html (29. prosince 2007).

[4] *JavaServer Faces Technology*. [online], Dokument dostupný na URL http://java.sun.com/javaee/javaserverfaces/ (16. května 2008).

[5] *MonoRail*. [online], Dokument dostupný na URL http://www.castleproject.org/monorail/ (16. května 2008).

[6] *quercus: php in java*. [online], Dokument dostupný na URL http://www.caucho.com/resin/doc/quercus.xtp (9. května 2008).

[7] *Ruby on Rails*. [online], Dokument dostupný na URL http://www.rubyonrails.org/ (16. května 2008).

[8] *Struts*. [online], Dokument dostupný na URL http://struts.apache.org/ (16. května 2008).

[9] *symfony Open-Source PHP Web Framework*. [online], Dokument dostupný na URL http://www.symfony-project.org/ (16. května 2008).

[10] *Zend Framework*. [online], Dokument dostupný na URL http://framework.zend.com/ (16. května 2008).

[11] *mod_php, LightTPD, FastCGI - What's fastest?* 1. srpna 2006, [online], Dokument dostupný na URL http://www.gnegg.ch/2006/08/mod_php-lighttpd-fastcgi-whats-fastest/ (9. května 2008).

[12] *FastCGI: A High-Performance Web Server Interface*. duben 1996, [online], Dokument dostupný na URL http://www.fastcgi.com/devkit/doc/fastcgi-whitepaper/fastcgi.htm (9. května 2008).

[13] BERNERS-LEE, T.: *The WorldWideWeb browser*. [online], Dokument dostupný na URL http://www.w3.org/People/Berners-Lee/WorldWideWeb.html (3. ledna 2008).

[14] BOX, D.; HEJLSBERG, A.: *LINQ: .NET Language-Integrated Query*. únor 2007, [online], Dokument dostupný na URL http://msdn.microsoft.com/en-gb/library/bb308959.aspx (8. května 2008).

[15] FIELDING, R.; GETTYS, J.; MOGUL, J.; aj.: *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. 1. září 2004, [online], Dokument dostupný na URL http://www.w3.org/Protocols/rfc2616/rfc2616.html (3. ledna 2008).

[16] GARRETT, J. J.: *AJAX: A New Approach to Web Applications*. 18. února 2005, [online], Dokument dostupný na URL http://www.adaptivepath.com/ideas/essays/archives/000385.php (29. prosince 2007).

[17] GREER, D.: *Interactive Application Architecture Patterns*. 25. srpna 2007, [online], Dokument dostupný na URL http://ctrl-shift-b.blogspot.com/2007/08/interactive-application-architecture.html (3. ledna 2008).

[18] KREJČÍ, L.: *PHP kapesní přehled*. Brno: Computer Press, a.s., 2006, ISBN 80-251-0808-2.

[19] KRISTOL, D.; MONTULLI, L.; aj.: *RFC 2965: HTTP State Management Mechanism*. říjen 2000, [online], Dokument dostupný na URL http://tools.ietf.org/html/rfc2965 (3. ledna 2008).

[20] MacDONALD, M.; SZPUSZTA, M.: *ASP.NET 2.0 a C# - tvorba dynamických stránek PROFESIONÁLNĚ*. Brno: Zoner Press, 2006, ISBN 80-86815-38-2.

[21] MARTINEZ, E.: *History of the Postal Service*. [online], Dokument dostupný na URL http://stamps.about.com/od/history/a/Postal_History.htm (3. ledna 2008).

[22] MATOUŠEK, T.: *Micro-benchmarks of the latest release*. 20. září 2006, [online], Dokument dostupný na URL http://www.codeplex.com/Phalanger/Wiki/View.aspx?title=The%20latest%20release&referringTitle=Benchmarks (9. května 2008).

[23] MELOAN, S.: Articles: Getting Started with Java 2: Tools and Configuration. In *Sun Developer Network*, prosinec 1998, [online], Dokument dostupný na URL http://java.sun.com/developer/technicalArticles/Programming/GettingStarted/index.html (3. ledna 2008).

[24] PROŠEK, L.: *phpBB: Creating communities twice as fast*. 4. září 2006, [online], Dokument dostupný na URL http://www.codeplex.com/Phalanger/Wiki/View.aspx?title=phpBB%3A%20Creating%20communities%20twice%20as%20fast&version=4 (9. května 2008).

[25] PROŠEK, L.; MATOUŠEK, T.: *Phalanger: Integrating PHP with CLR*. červenec 2006, [online], Dokument dostupný na URL http://php-compiler.net/lib/exe/fetch.php?id=documentation&cache=cache&media=integrating_php_with_clr.pdf (9. května 2008).

[26] ROBINSON, S.; ALLEN, K. S.; CONES, O.; aj.: *C# Programujeme profesionálně*. Brno: Computer Press, a.s., 2003, ISBN 80-251-0085-5.

[27] SANTOS, J.: *In Response to JIT Compilation*. 26. dubna 2007, [online], Dokument dostupný na URL http://www.santosj.name/writing-projects/in-response-to-jit-compilation/ (9. května 2008).

[28] SKLAR, D.: *PHP 5 - moduly, rozšíření a akcelerátory*. Brno: Zoner Press, 2005, ISBN 80-86815-19-6.

[29] THOMPSON, K.: *Clarification on MVC Pull and MVC Push*. 29. října 2003, [online], Dokument dostupný na URL http://www.theserverside.com/patterns/thread.tss?thread_id=22143 (3. ledna 2008).

[30] ZAKAS, N. C.; McPEAK, J.; FAWCETT, J.: *AJAX Profesionálně*. Brno: Zoner Press, 2007, ISBN 978-80-86815-77-0.

Dodatek A

Uživatelský manuál

A.1 Úvod

A.1.1 Obecné informace o aplikaci

Aplikace slouží k uchovávání databáze klientů a zásilek, tištění etiket na zásilky, odesílání informací o odesílaných zásilkách do databáze dopravce, tištění přejímacích dokladů a informování příjemců zásilek o expedici.

A.1.2 Informace o nápovědě

Nápověda je rozdělena na dvě části. V této části jsou obecně popsány funkce aplikace a nastíněno používání aplikace. Druhá část se pak zabývá podrobným popisem funkčnosti jednotlivých formulářů dostupných z menu aplikace.

A.2 Přehled funkcí aplikace

Před tím, než je možné aplikaci plně využívat, je nutné nastavit firemní nastavení a vytvořit uživatele aplikace v administrační části. Přístup do administrační části je možný pouze se znalostí firemního kódu a hesla do aplikace.

A.2.1 Databáze zákazníků

Databáze zákazníků slouží ke snazšímu vytváření zásilek. Jediné přímé využití informací o zákaznících v aplikaci je pro zkopírování údajů o zákazníkovi do nově vytvářené zásilky. Samozřejmě je možné využít databázi zákazníků i jinak, protože umožnuje vyhledávání, třídění, zadávaní poznámek a ukládání kontaktních údajů. Údaje do databáze zákazníků je možné také importovat odesláním souboru v jednom z podporovaných formátů.

A.2.2 Databáze zásilek

Databáze zásilek udržuje informace o připravovaných i odeslaných zásilkách. Umožňuje jejich tisk a odesílání do databáze dopravce. K odeslaným zásilkám je pak možné vytisknout přejímací doklady a informovat příjemce zásilek o expedici. Přejímací doklady jsou vždy tištěny pro zásilky odeslané do databáze dopravce v den tisku přejímacích dokladů. Totéž platí i pro informování příjemců. Není nijak ošetřeno, aby příjemci nebyli o dané zásilce informováni dvakrát nebo aby nebyl dvakrát vytištěn přejímací doklad. Očekává se, že

na konci pracovního dne jsou všechny připravené zásilky obsluhou aplikace odeslány do databáze dopravce, vytištěn potřebný přejímací doklad a odeslány informace příjemcům.

A.2.3 Vstup dat

Aplikace umožnuje zadávání zákazníků i zásilek do databáze buď ručně nebo poloautomaticky. Při ručním zadávání jsou všechny údaje zadávány do formuláře pro vložení zákazníka či zásilky. Více informací o ručním zadávání dat naleznete v nápovědě o jednotlivých formulářích:

- formulář pro vytvoření a editaci zákazníka,
- tabulka zákazníků,
- formulář pro vytvoření, editaci a tisk zásilky,
- tabulka zásilek.

Poloautomatickým vkládáním je myšleno importovaní údaju ze souborů v jednom z následujících formátů:

- soubor ve formátu DBASE4 (přípona *.dbf),
- tabulka s oddělovači (obvykle přípona *.csv),
- tabulka s pevnou šířkou sloupců (obvykle přípona *.txt).

A.2.4 Importování dat

Před importem dat je nutné import nastavit. Nejprve je potřeba vytvořit mapování pro číselníky.

Nastavení mapování číselníku

Mapování číselníku obsahuje informace o tom, jaký význam mají hodnoty ve vstupním souboru. Mapování se vytváří pro číselník typů zásilek, měn a zemí. Mapování není nutné nastavovat, pokud odpovídá přednastavenému mapování. Přednastavené mapování typů zásilek má název "-Typ zásilky-" a obsahuje tyto hodnoty:

Hodnota v importovaném souboru	Význam v číselníku
1	Normální balík
Normální balík	Normální balík
2	Normální balík - dobírka
Normální balík - dobírka	Normální balík - dobírka
13	Soukromá adresa
Soukromá adresa	Soukromá adresa
14	Soukromá adresa - dobírka
Soukromá adresa - dobírka	Soukromá adresa - dobírka
7	Express
Express	Express
8	Express - dobírka
Express - dobírka	Express - dobírka
9	Export
Export	Export
10	Export - dobírka
Export - dobírka	Export - dobírka

Přednastavené mapování měn má název "-Kód měny-" a obsahuje tyto hodnoty:

Hodnota v importovaném souboru	Význam v číselníku
CZK	Česká koruna
Česká koruna	Česká koruna
EUR	Euro
Euro	Euro
SKK	Slovenská koruna
Slovenská koruna	Slovenská koruna
USD	Americký dolar
Americký dolar	Americký dolar

Přednastavené mapování zemí má název "-Kód země-" a obsahuje tyto hodnoty:

Hodnota v importovaném souboru	Význam v číselníku
AT	Rakousko
Rakousko	Rakousko
BE	Belgie
Belgie	Belgie
CZ	Česká republika
Česká republika	Česká republika
DE	Německo
Německo	Německo
DK	Dánsko
Dánsko	Dánsko
EE	Estonsko
Estonsko	Estonsko
ES	Španělsko
Španělsko	Španělsko
FI	Finsko
Finsko	Finsko
FR	Francie
Francie	Francie
GB	Velká Britanie
Velká Britanie	Velká Britanie
HU	Maďarsko
Maďarsko	Maďarsko
CH	Švýcarsko
Švýcarsko	Švýcarsko
IE	Irsko
Irsko	Irsko
IT	Itálie
Itálie	Itálie
LT	Litva
Litva	Litva
LV	Lotyšsko
Lotyšsko	Lotyšsko
NL	Nizozemí
Nizozemí	Nizozemí
NO	Norsko
Norsko	Norsko
PL	Polsko
Polsko	Polsko
PT	Portugalsko
Portugalsko	Portugalsko
SE	Švédsko
Švédsko	Švédsko
SI	Slovinsko
Slovinsko	Slovinsko

Hodnota v importovaném souboru	Význam v číselníku
SK	Slovensko
Slovensko	Slovensko
US	USA
USA	USA

Pokud tedy hodnoty v importovaném souboru odpovídají hodnotám uvedeným v tabulkách nalevo a jejich význam hodnotám uvedeným napravo, není potřeba vytvářet žádné mapování číselníku. Pokud ne, více o vytváření mapování se dozvíte v nápovědě k formuláři pro vytvoření a editaci mapování číselníku. Mapování číselníku je po nastavení možné použít v libovolném nastavení importu.

Nastavení importu

Dalším krokem je nastavení samotného importu. To se provádí zvlášť pro zákazníky a zvlášť pro zásilky a v obou případech se dělí na dvě části. Nastavení formátu vstupních dat a nastavení, který sloupec ze vstupního souboru má být namapován na kterou položku zákazníka či zásilky. Podrobnější informace o nastavení naleznete v nápovědě k formuláři pro vytvoření a editaci nastavení importu zákazníka a importu zásilky.

Import dat

Jakmile je nastavem import dat, je možné toto nastavení opakovaně používat pro importování dat ve zvoleném formátu. Po importu jsou všechna data vložena do databáze a je zobrazen formulář se všemi importovanými záznamy a chybami, ke kterým při importu došlo. Importované záznamy je možné v tomto formuláři smazat a upravovat. Více o importu naleznete v nápovědě k formuláři pro import zákazníků a zásilek.

A.3 Nápověda k formulářům

Tato část nápovědy je členěna stejně jako menu aplikace. Pokud se ve formuláři objeví ve výběru položka -nezadáno-, její zvolení znamená, že do databáze nebude vložena hodnota. Pokud se ve formuláři objeví ve výběru položka -zrušit-, její zvolení způsobí neuložení celého řádku do databáze. Pokud se nepodaří provést některou z požadovaných akcí, zobrazí se poblíž tlačítka, které ji vyvolalo, červeně text informující o vzniklé chybě. Pokud je akce úspěšná, dojde buď k přesměrování nebo je o tom uživatel informován zobrazením textu poblíž tlačítka, které akci vyvolalo.

A.3.1 Administrační část aplikace

Administrační část aplikace TiskEtiket slouží k vytváření uživatelů aplikace a zadání firemního nastavení.

Firemní nastavení

V části Parametry tiskárny pro etikety se nastavují parametry tisku na tiskáně datamax - zatím neimplementováno.

V části Povolení lze obsluze povolit či zakázat odesílání E-mailů s informacemi o číslech zásilek a dobírkové částce příjemcům zásilek a importování zásilek z datových souborů.

V části Adresa odesílatele 1 (hlavní adresa) se vyplňuje adresa odesilatele, která bude vytištěna na etiketách.

V části Adresa odesílatele 2 (zpětná adresa) se vyplňuje adresa, která bude vytištěna jako adresa příjemce na zpětných zásilkách.

V části Email se vyplňuje E-mailová adresa, která bude používána jako adresa odesilatele v E-mailech odesílaných příjemcům zásilek.

Správa uživatelů

Tento formulář se automaticky zobrazí po přihlášení do administrační části aplikace. Je možné se k němu také dostat kliknutím na položku v menu. Formulář slouží k vytváření uživatelských účtů pro firmu. Všechny účty mají stejná práva. Každý účet má pouze uživatelské jméno a heslo, které je možné v kdykoli změnit. Vytvořený účet není možné smazat, pouze zamknout a zabránit tak přihlášení do aplikace. Pro přidání nebo změnu hesla uživatele stačí pouze vyplnit uživatelské jméno a heslo a stisknout příslušné tlačítko (přidat uživatele nebo změnit heslo). Nad ovládacími prvky pro přidávání uživatelů a změnu hesla se zobrazuje seznam všech uživatelů, kteří již byli vytvořeni.

A.3.2 Zákazník

Nový zákazník

Formulář slouží ke vložení nového zákazníka do databáze. Vložení se provede po vyplnění údajů kliknutím na tlačítko Vložit. PSČ zákazníka je možné buď vybrat z výběru nebo vepsat do pole Vlastní PSČ. Pole Vlastní PSČ má přednost před výběrem, proto bude v případě vyplnění obou hodnot uložena hodnota z pole Vlastní PSČ. Pokud nechcete PSČ uložit do databáze, musíte nechat prázdné pole Vlastní PSČ a ve výběru zvolit hodnotu -nezadáno-.Pokud je ve firemním nastavení povoleno odesílání E-mailů, pak bude po provedení akce Odeslat mail zákazníkům odeslán zákazníkovi informační E-mail o expedovaných zásilkách na adresu vyplněnou v poli E-mail. Pole Poznámka pro tisk na etiketě se zobrazuje pouze v přejímacích dokladech. Pole Poznámka zatím nemá v aplikaci využití a slouží čistě pro vaše účely. Pole Telefon pro SMS slouží čistě pro vaše účely.

Seznam zákazníků

Formulář slouží k vyhledávání, zobrazování a úpravě údajů o zákaznících. Umožňuje řazení podle libovolného sloupce a filtrování zobrazovaných záznamů. Řazení se aplikuje po kliknutí na název sloupce v tabulce zákazníků. Opětovným kliknutím na stejný sloupec se obrátí směr řazení a třetím kliknutím se řazení zruší. Filtrování se nastavuje v části Filtr. Při filtrování se zobrazují pouze ty řádky splňující všechny zadané podmínky. Filtry Firma, Ulice a Město vyhledávají zadaný text uvnitř příslušného sloupce. Filtr PSČ vyžaduje přesnou shodu celého PSČ (mezery jsou při porovnávání ignorovány). Filtr se aplikuje až po kliknutí na tlačítko Zobraz. Při provádění akcí v rámci formuláře (listování, řazení) zůstává ve filtru hodnota, kterou jste do něj naposledy vepsali. Pokud jste tedy nestiskli tlačítko Zobraz, nejsou ve filtru vyplněny hodnoty, podle kterých je tabulka filtrována. Každého zákazníka je možné upravit nebo smazat z databáze kliknutím na příslušné tlačítko v tabulce.

Detail zákazníka Do tohoto formuláře se dostanete ze seznamu zákazníků. Slouží k zobrazení údajů o zákazníkovi a jejich úpravám. Upravené údaje je možné uložit kliknutím

na tlačítko Ulož. Kliknutím na tlačítko Zpět se vrátíte do seznamu zákazníků bez uložení změn. Více o položkách v tomto formuláři naleznete v nápovědě k formuláři Nový zákazník.

Import zákazníků

Formulář slouží k importu údajů o zákaznících ze souboru v jednom z podporovaných formátů. Obecné informace o importování naleznete v první části nápovědy. Podrobnější informace o nastavení importů naleznete v popisu jednotlivých formulářů. Pokud máte import zákazníků nastaven, stačí zvolit správné nastavení v poli Vyberte nastavení, zvolit soubor s importovanými daty v poli Vybraný soubor a kliknout na tlačítko Vlož. Vstupní soubor bude zpracován a data vložená do databáze budou zobrazena v tabulce pod formulářem včetně výpisu chyb, ke kterým v průběhu importování došlo. Importované záznamy je pak možné mazat a editovat. Každý záznam, který po editaci uložíte, je považován za korektní a proto je z výpisu vyjmut. Každý záznam, který nesmažete zůstane v databázi. Formulář také umožňuje zobrazení zvoleného nastavení kliknutím na tlačítko Detail a vytvoření nového nastavení klinutím na tlačítko Nové. POZOR!! Pokud použijete jedno ze zmíněných dvou tlačítek, nedostanete se již k seznamu všech importovaných záznamů a budete je muset pracněji kontrolovat v seznamu zákazníků.

A.3.3 Zásilka

Nová zásilka

Formulář slouží ke vložení nové zásilky do databáze. Vložení se provede po vyplnění údajů kliknutím na tlačítko Vložit. Pole Doplnit adresu podle stávajícího zákazníka umožňuje vybrat si jednoho ze zákazníků uložených v databázi a kliknutím na tlačítko Zvolit vyplnit všechny aplikovatelné údaje o zákazníkovi do nově vytvářené zísilky. Aplikovatelné jsou údaje se shodným názvem. PSČ zásilky je možné buď vybrat z výběru nebo vepsat do pole Vlastní PSČ. Pole Vlastní PSČ má přednost před výběrem, proto bude v případě vyplnění obou hodnot uložena hodnota z pole Vlastní PSČ. Pokud nechcete PSČ uložit do datbáze, musíte nechat prázdné pole Vlastní PSČ a ve výběru zvolit hodnotu -nezadáno-. Po zvolení typu zásilky je formulář překreslen a zobrazí se pole odpovídající zvolenému typu zásilky. Vyplníte-li počet kusů jiný než 1, vytvoří se do databáze zvolený počet zásilek. Pole Telefon pro SMS slouží čistě pro vaše účely. Pokud je ve firemním nastavení povoleno odesílání E-mailů, pak bude po provedení akce Odeslat mail zákazníkům odeslán příjemci informační E-mail o expedovaných zásilkách na adresu vyplněnou v poli E-mail. Pole Poznámka pro tisk na etiketě se zobrazuje pouze v přejímacích dokladech. Pole Zákaznická reference slouží pro vložení vašeho čísla zásilky (jedná se o nepovinnou hodnotu). Do pole Datum a čas doručení můžete vyplnit, v jakou dobu by zásilka měla být doručena. Musí to však být v pracovní den mezi 8:00 a 21:00 a časový interval pro doručení musí být alespoň 2 hodiny.

Seznam zásilek

Formulář slouží k vyhledávání, zobrazování, úpravě údajů a tisku zásilek. Umožňuje řazení podle libovolného sloupce a filtrování zobrazovaných záznamů. Řazení se aplikuje po kliknutí na název sloupce v tabulce zákazníků. Opětovným kliknutím na stejný sloupec se obrátí směr řazení a třetím kliknutím se řazení zruší. Filtrování se nastavuje v části Filtr. Při filtrování se zobrazují pouze ty řádky splňující všechny zadané podmínky. Filtr Číslo zásilky i PSČ vyžadují přesnou shodu celého sloupce. Filtr Vytvořeno od do zobrazuje jen zásilky

vytvořené ve zvoleném intervalu. Filtr se aplikuje až po kliknutí na tlačítko Zobraz. Při provádění akcí v rámci formuláře (listování, řazení) zůstává ve filtru hodnota, kterou jste do něj naposledy vepsali. Pokud jste tedy nestiskli tlačítko Zobraz, nejsou ve filtru vyplněny hodnoty, podle kterých je tabulka filtrována. Každou zásilku je možné upravit nebo smazat z databáze kliknutím na příslušné tlačítko v tabulce. Zaškrtávací políčka nalevo v tabulce slouží k výběru zásilek, pro které chcete vytisknout etikety. Po vybrání všech zásilek k tisku klikněte na tlačítko Vytisknout vybrané a bude zobrazen formulář pro tisk zásilek.

Detail zásilky Do tohoto formuláře se dostanete ze seznamu zásilek. Slouží ke zobrazení údajů o zásilce, jejich úpravám a tisku etikety zásilky. Upravené údaje je možné uložit kliknutím na tlačítko Ulož. Kliknutím na tlačítko Zpět se vrátíte do seznamu zásilek bez uložení změn. Pokud chcete zásilku vytisknout, zvolte z výběru vedle tlačítka Tisk požadovanou tiskárnu a klikněte na Tisk. Více o položkách v tomto formuláři naleznete v nápovědě k formuláři Nový zákazník.

Tisk výběru Formulář slouží k tisku etiket pro více zásilek. Umožňuje zkontrolovat, zda opravdu vytisknout etikety všem zvoleným zásilkám (seznam se zobrazuje pod částí Výběr tiskárny) a zvolit požadovanou tiskárnu z výběru. Tisk se provede kliknutím na tlačítko Tisk.

Import zásilek

Formulář slouží k importu údajů o zásilkách ze souboru v jednom z podporovaných formátů. Obecné informace o importování naleznete v první části nápovědy. Podrobnější informace o nastavení importů naleznete v popisu jednotlivých formulářů. Pokud máte import zásilek nastaven, stačí zvolit správné nastavení v poli Vyberte nastavení, zvolit soubor s importovanými daty v poli Vybraný soubor a kliknout na tlačítko Vlož. Vstupní soubor bude zpracován a data vložená do databáze budou zobrazena v tabulce pod formulářem včetně výpisu chyb, ke kterým v průběhu importování došlo. Importované záznamy je pak možné mazat a editovat. Každý záznam, který po editaci uložíte, je považován za korektní a proto je z výpisu vyjmut. Každý záznam, který nesmažete zůstane v databázi. Formulář také umožňuje zobrazení zvoleného nastavení kliknutím na tlačítko Detail a vytvoření nového nastavení klinutím na tlačítko Nové. POZOR!! Pokud použijete jedno ze zmíněných dvou tlačítek, nedostanete se již k seznamu všech importovaných záznamů a budete je muset pracněji kontrolovat v seznamu zásilek.

A.3.4 Nastavení

Nové nastavení importu zákazníků

Formulář slouží k vytvoření nového nastavení importu zákazníků. Dříve, než začnete nastavení vyplňovat, doporučuji přečíst si obecné informace o importech v první části nápovědy. Do pole Název nastavení vyplňte název, který jednoznačně identifikuje dané nastavení, protože tento název se jako jediný zobrazuje ve výběru nastavení. V části Typ importovaného souboru nejprve zvolte typ souboru z výběru a pak teprve vyplňujte ostatní údaje, protože po změně typu souboru se zobrazí jiné položky k nastavení. Pokud zašktnete Importovat i záznamy s chybou, budou importovány i záznamy, při jejichž importu došlo k některé z následujících chyb:

- některý ze sloupců nebyl ve vstupním souboru nalezen,
- hodnotu v některém ze sloupců se nepodařilo převést na správný typ.

O těchto chybách sice budete informováni, ale budete pak muset chybné záznamy buď smazat nebo odstranit, což může být při větším množství záznamů zdlouhavé.

Nastavení importu souboru s pevnou šířkou sloupce Import souboru s pevnou šířkou sloupce očekává na vstupu textový soubor, ve kterém je použit oddělovač na oddělení řádků a sloupce mají pevný počet znaků. V poli Encoding se volí kódování vstupního souboru. Do pole Šířka sloupce defaultní je vhodné zadat nejčastější šířku sloupce. Ta pak bude doplněna jako šířka všech sloupců, u kterých není šířka zadána v tabulce ve spodní části Parametrů souboru s pevnou šířkou sloupce. Pole Šířka mezery defaultní má stejný význam jako předchozí pole, ale určuje šířku mezery mezi sloupci, která se nemá počítat jako obsah žádného z nich (obvykle žádná mezera mezi sloupci není a bývá proto vyplněna hodnota 0). Pokud není zaškrtnuto pole Povolit konec řádku uprostřed sloupce, zakáže se import řádků, u nihž řádek končí uprostřed sloupce. Například když má řádek 100 znaků a je nastaveno, že každý sloupec má 30 znaků, nebude řádek importován, protože poslední sloupec má 10 znaků místo 30ti. V poli Oddělovač řádků je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny řádky nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. Pokud je zaškrtnuto pole Ořezat bílé znaky, budou ořezány bílé znaky (mezera, tabulátor) na začátku a na konci textu každého sloupce. Pokud je zaškrtnuto pole Vynechat první řádek, bude první řádek ze vstupního souboru vynechán. Tato volba se používá, pokud první řádek obsahuje názvy sloupců. V tabulce dole se vyplňuje šířka sloupce a mezery u sloupců, jejichž parametry se neshodují s defaultní šířkou sloupce a nebo defaultní šířkou mezery.

Nastavení importu souboru s oddělovači Import souboru s oddělovači očekává na vstupu textový soubor, ve kterém je použit oddělovač na oddělení sloupců a jiný oddělovač na oddělení řádků. V poli Encoding se volí kódování vstupního souboru. V poli Oddělovač sloupců je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny sloupce nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. V poli Oddělovač řádků je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny řádky nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. V poli Únikový znak můžete zadat znak, který ruší speciální význam následujícího znaku a vypíše ho do výsledku. Pokud je potřeba vypsat únikový znak, musí být v souboru dvakrát za sebou. Pokud například zvolím jako únikový znak ’
’ a jako oddělovač sloupců ’;’, pak dvojice znaků ’
;’ neukončí sloupec ale vypíše do daného sloupce ’;’. V poli Uvozovací znak můžete zadat znak který je použit jako uvození řetězce. Řetězec začíná a končí uvozovacím znakem. V řetězci je zrušen speciální význam všech znaků kromě uvozovacího, který řetězec ukončuje. Pokud je zaškrtnuto pole Dva uvozovací znaky jako jeden jsou dva uvozovací znaky za sebou dány na výstup jako jeden uvozovací znak (toto chování je obvyklé při výstupu z tabulkových procesorů).Pokud je zaškrtnuto pole Ořezat bílé znaky, budou ořezány bílé znaky (mezera, tabulátor) na začátku a na konci textu každého sloupce. Pokud je zaškrtnuto pole Vynechat první řádek bude první řádek ze vstupního souboru vynechán. Tato volba se používá, pokud první řádek obsahuje názvy sloupců.

Nastavení importu souboru ve formátu DBF Import souboru ve formátu DBF očekává na vstupu soubor ve formátu DBASE4. Pro tento druh importu zatím nejsou žádná nastavení.

Společné nastavení pro všechny typy souboru Pod specifickým nastavením pro různé typy vstupních souborů je nastavení společné. Zde se vyplňují čísla sloupců, na kterých jsou údaje k nalezení ve vstupním souboru, defaultní hodnoty, které jsou do databáze vloženy, pokud není zadáno číslo sloupce nebo dojde k chybě při importu daného sloupce, a mapování, která se mají použít pro číselníky.

Seznam nastavení importu zákazníků

Formulář obsahuje seznam všech vytvořených nastavení importu zákazníků. Umožňuje jejich úpravu (kliknutím na tlačítko Upravit) a výmaz (kliknutím na tlačítko Smazat).

Detail nastavení importu zákazníků Do tohoto formuláře se dostanete ze seznamu nastavení importu zákazníků. Slouží ke zobrazení nastavení importu zákazníků a jeho úpravám. Upravené nastavení je možné uložit kliknutím na tlačítko Ulož. Kliknutím na tlačítko Zpět se vrátíte do seznamu nastavení importu zákazníků bez uložení změn. Více o položkách v tomto formuláři naleznete v nápovědě k formuláři Nové nastavení importu zákazníků.

Nové nastavení importu zásilek

Formulář slouží k vytvoření nového nastavení importu zásilek. Dříve, než začnete nastavení vyplňovat, doporučuji přečíst si obecné informace o importech v první části nápovědy. Do pole Název nastavení vyplňte název, který jednoznačně identifikuje dané nastavení, protože tento název se jako jediný zobrazuje ve výběru nastavení. V části Typ importovaného souboru nejprve zvolte typ souboru z výběru a pak teprve vyplňujte ostatní údaje, protože po změně typu souboru se zobrazí jiné položky k nastavení. Pokud zašktnete Importovat i záznamy s chybou, budou importovány i záznamy, při jejichž importu došlo k některé z následujících chyb:

- některý ze sloupců nebyl ve vstupním souboru nalezen,
- hodnotu v některém ze sloupců se nepodařilo převést na správný typ.

O těchto chybách sice budete informováni, ale budete pak muset chybné záznamy buď smazat nebo odstranit, což může být při větším množství záznamů zdlouhavé.

Nastavení importu souboru s pevnou šířkou sloupce Import souboru s pevnou šířkou sloupce očekává na vstupu textový soubor, ve kterém je použit oddělovač na oddělení řádků a sloupce mají pevný počet znaků. V poli Encoding se volí kódování vstupního souboru. Do pole Šířka sloupce defaultní je vhodné zadat nejčastější šířku sloupce. Ta pak bude doplněna jako šířka všech sloupců, u kterých není šířka zadána v tabulce ve spodní části Parametrů souboru s pevnou šířkou sloupce. Pole Šířka mezery defaultní má stejný význam jako předchozí pole, ale určuje šířku mezery mezi sloupci, která se nemá počítat jako obsah žádného z nich (obvykle žádná mezera mezi sloupci není a bývá proto vyplněna hodnota 0).Pokud není zaškrtnuto pole Povolit konec řádku uprostřed sloupce, zakáže se import řádků, u nihž řádek končí uprostřed sloupce. Například když má řádek 100 znaků

a je nastaveno, že každý sloupec má 30 znaků, nebude řádek importován, protože poslední sloupec má 10 znaků místo 30ti. V poli Oddělovač řádků je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny řádky nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. Pokud je zaškrtnuto pole Ořezat bílé znaky budou ořezány bílé znaky (mezera, tabulátor) na začátku a na konci textu každého sloupce. Pokud je zaškrtnuto pole Vynechat první řádek bude první řádek ze vstupního souboru vynechán. Tato volba se používá, pokud první řádek obsahuje názvy sloupců. V tabulce dole se vyplňuje šířka sloupce a mezery u sloupců, jejichž parametry se neshodují s defaultní šířkou sloupce a nebo defaultní šířkou mezery.

Nastavení importu souboru s oddělovači Import souboru s oddělovači očekává na vstupu textový soubor, ve kterém je použit oddělovač na oddělení sloupců a jiný oddělovač na oddělení řádků. V poli Encoding se volí kódování vstupního souboru. V poli Oddělovač sloupců je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny sloupce nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. V poli Oddělovač řádků je potřeba buď vybrat znak, kterým jsou ve vstupním souboru odděleny řádky nebo zvolit hodnotu Tisknutelný znak ve vedlejším poli a oddělovač zadat do prázdného pole napravo. V poli Únikový znak můžete zadat znak, který ruší speciální význam následujícího znaku a vypíše ho do výsledku. Pokud je potřeba vypsat únikový znak, musí být v souboru dvakrát za sebou. Pokud například zvolím jako únikový znak ’
’ a jako oddělovač sloupců ’;’, pak dvojice znaků ’
;’ neukončí sloupec ale vypíše do daného sloupce ’;’. V poli Uvozovací znak můžete zadat znak který je použit jako uvození řetězce. Řetězec začíná a končí uvozovacím znakem. V řetězci je zrušen speciální význam všech znaků kromě uvozovacího, který řetězec ukončuje. Pokud je zaškrtnuto pole Dva uvozovací znaky jako jeden, jsou dva uvozovací znaky za sebou dány na výstup jako jeden uvozovací znak (toto chování je obvyklé při výstupu z tabulkových procesorů). Pokud je zaškrtnuto pole Ořezat bílé znaky budou ořezány bílé znaky (mezera, tabulátor) na začátku a na konci textu každého sloupce. Pokud je zaškrtnuto pole Vynechat první řádek bude první řádek ze vstupního souboru vynechán. Tato volba se používá, pokud první řádek obsahuje názvy sloupců.

Nastavení importu souboru ve formátu DBF Import souboru ve formátu DBF očekává na vstupu soubor ve formátu DBASE4. Pro tento druh importu zatím nejsou žádná nastavení.

Společné nastavení pro všechny typy souboru Pod specifickým nastavením pro různé typy vstupních souborů je nastavení společné. Zde se vyplňují čísla sloupců, na který jsou údaje k nalezení ve vstupním souboru, defaultní hodnoty, které jsou do databáze vloženy, pokud není zadáno číslo sloupce nebo dojde k chybě při importu daného sloupce, a mapování, která se mají použít pro číselníky.

Seznam nastavení importu zásilek

Formulář obsahuje seznam všech vytvořených nastavení importu zásilek. Umožňuje jejich úpravu (kliknutím na tlačítko Upravit) a výmaz (kliknutím na tlačítko Smazat).

Detail nastavení importu zásilek Do tohoto formuláře se dostanete ze seznamu nastavení importu zásilek. Slouží ke zobrazení nastavení importu zásilek a jeho úpravám. Upravené nastavení je možné uložit kliknutím na tlačítko Ulož. Kliknutím na tlačítko Zpět se vrátíte do seznamu nastavení importu zásilek bez uložení změn. Více o položkách v tomto formuláři naleznete v nápovědě k formuláři Nové nastavení importu zásilek.

Nové mapování číselníku

Formulář slouží k vytvoření nového mapování číselníku. Dříve, než začnete nastavení vyplňovat, doporučuji přečíst si obecné informace o importech v první části nápovědy. Do pole Název nového mapování číselníku vyplňte název, který jednoznačně identifikuje dané nastavení, protože tento název se jako jediný zobrazuje ve výběru nastavení. Pole Popis mapování číselníku slouží jen jako krátká poznámka, která se zobrazuje pouze v detailu mapovani ciselniku. V části Výběr číselníku nejprve zvolte číselník z výběru a pak teprve vyplňujte část Vyplnění příslušných hodnot k mapování číselníku, protože po změně číselníku bude zmíněná část vymazána. V části Vyplnění příslušných hodnot k mapování číselníku se vyplňují jednotlivé hodnoty ve vstupním souboru a jim náležící význam. Věnujte prosím pozornost poznámce na začátku této části nápovědy.

Seznam mapování číselníků

Formulář obsahuje seznam všech vytvořených mapování číselníků pro číselník zvolený ve výběru Výběr číselníku. Umožňuje jejich úpravu (kliknutím na tlačítko Upravit) a výmaz (kliknutím na tlačítko Smazat).

Detail mapování číselníku Do tohoto formuláře se dostanete ze seznamu mapování číselníků. Slouží ke zobrazení mapování číselníku a jeho úpravám. Upravené mapování je možné uložit kliknutím na tlačítko Ulož. Kliknutím na tlačítko Zpět se vrátíte do seznamu mapování číselníků bez uložení změn. Více o položkách v tomto formuláři naleznete v nápovědě k formuláři Nové mapování číselníku.

Změna hesla

Tento formulář umožňuje změnu hesla přihlášeného uživatele. Pro kontrolu je potřeba vyplnit Staré heslo: a dvakrát Nové heslo:. Heslo je změněno po kliknutí na tlačítko Změnit heslo.

A.3.5 Ukončení dne

Tato část menu obsahuje úkony, které jsou obvykle prováděny na konci pracovního dne.

Přenos zásilek

Formulář slouží k výběru zásilek k přenosu do databáze dopravce. Kalendáře v části Výběr dne vytištění určují, kdy byly vytištěny zásilky, které se mají zobrazit v tabulce pro výběr zásilek na odeslání. Po vybrání zásilek jsou tyto odeslány kliknutím na tlačítko Odeslat označené do databáze dopravce.

Sestavy - Přejímací doklad všech balíků

Kliknutím na tuto položku menu se vygeneruje přejímací doklad všech zásilek přenesených dnes do databáze dopravce.

Sestavy - Přejímací doklad dobírkových balíků

Kliknutím na tuto položku menu se vygeneruje přejímací doklad všech dobírkových zásilek přenesených dnes do databáze dopravce.

Odeslat mail zákazníkům

Kliknutím na tuto položku menu se odešlou E-maily všem příjemcům zásilek, jejichž zásilka byla dnes přenesena do databáze dopravce.

Dodatek B

Diagram entit a vztahů

Zde bude umístěn diagram.

Dodatek C

CD se zdrojovým kódem

K diplomové práci je přiloženo i CD se zdrojovým kódem aplikace a touto prací v digitální podobě. Aplikaci není možné přeložit, protože jsem k ní nemohl přiložit licencovanou knihovnu pro vytváření pdf sestav ComponentOne a knihovnu libtools použitou pro persistenci dat.