VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

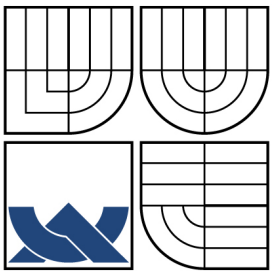
FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

NÁVRH INTERNETOVEJ STRÁNKY

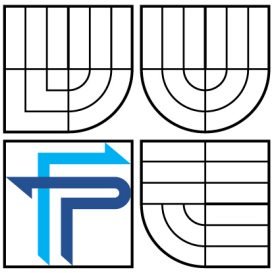
BAKALÁRSKA PRÁCA
BACHELOR'S THESIS

AUTOR PRÁCE VILIAM HUSÁR
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

NÁVRH INTERNETOVEJ STRÁNKY
WEBSITE DESIGN

BAKALÁRSKA PRÁCA
BACHELOR'S THESIS

AUTOR PRÁCE — VILIAM HUSÁR
AUTHOR

VEDÚCI PRÁCE — doc. Ing. Miloš Koch, CSc.
SUPERVISOR

BRNO 2011

ZADANIE PRÁCE

Abstrakt

Cieľom tejto práce je navrhnúť internetovú aplikáciu pre spoločnosť VHML, spol. s r.o., pomocou ktorej budú môcť zamestnanci tvoriť cenové ponuky a posielať ich zákazníkom. Táto aplikácia bude založená na nasledujúcich technológiách: PHP, MySQL, FLEX a LaTeX.

Kľučové slová

www, php, mvc, flash, action script, mysql, databáza, orm, flex, doctrine, latex

Abstract

The aim of this bachelor's thesis is to design a web application for company VHML, spol. s r.o., by help of which employees will be able to create quotations and pricelist and send them to customers. This application will based on following technologies: PHP, MySQL, FLEX and LaTeX.

Keywords

www, php, mvc, flash, action script, mysql, database, orm, flex, doctrine, latex

Bibliografická citácia VŠKP

HUSÁR, V. *Návrh internetovej stránky.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2011. 66 s. Vedúci bakalárskej práce doc. Ing. Miloš Koch, CSc.

Prehlásenie

Prehlasujem, že predložená bakalárska práca je pôvodná a spracoval som ju samostatne. Prehlasujem, že citácia použitých prameňov je úplna, že som vo svojej práci neporušil autorské práva (v zmysle Zákona č. 121/2000 Sb., o práve autorskom a o právach súvisiacich s právom autorským).

V Brne ....

(podpis autora)

Zoznam tabuliek

Zoznam obrázkov

Obsah

Úvod

Od 90. rokov 20.storočia, kedy internet začal prekračovať hranice akademickej pôdy, ubehlo ani nie 20 rokov a zrejme nikto si nepredstavoval, že sa dostane až do podoby, v ktorej ho poznáme dnes. Ale nebolo to prvá ani posledná technológia s tak rapídnym rozvojom.

Chronologicky si prešiel niekoľkými štádiami. Od čisto textovej podoby, ktorú si už súčasná generácia ani nepamätá, až do štádia, kedy obyčajná webová stránka môže predstatovať komplexnú plnohodnotnú aplikáciu. Stal sa súčasťou nášho každodenného života a to bez ohľadu na jeho pozitívne alebo negatívne stránky.

Svoje uplatnenie si veľmi rýchlo našiel aj v organizáciách, kde sa mu podarilo zbúrať mnoho bariér a stal sa neoceniteľným pracovným prostriedkom. Práve o možnostiach jeho využitia aj v mikro spoločnostiach by som rád venoval v tejto práci. Častokrát stačí naozaj málo a práca v spoločnosti sa stane efektívnejšia, rýchlejšia, ale hlavne príjemnejšia.

1 Teoretické východiská

V tejto kapitole predstavím hlavné technológie a nástroje, ktoré som pri spracovaní bakalárskej práce využil. Okrem stručného popisu a definícií, spomeniem ich výhody aj nevýhody, a pokúsim sa odôvodniť, prečo som si ich vybral. Nakoľko som ich využil väčší počet a dalo by sa o nich písať ďaleko za limit bakalárskej práce, mojím cieľom je poskytnúť čitateľovi aspoň základný obraz a dostatočné východisko pri porozumení praktickej časti.

1.1 Internetové aplikácie

Internetové aplikácie, resp. informačné systémy si vďaka svojim nesporným výhodám našli svoje miesto v mnohých organizáciách. Pod pojmom internetová aplikácia si môžme predstaviť akúkoľvek webovú stránku, ktorá užívateľovi poskytuje prostredie pre zobrazenie a prácu s dátami. (8, str. 4). Technológií, ktoré toto umožnujú, je nespočetné množstvo a záleží na vývojovom tíme, ktorú si vyberie. Každá z nich má svoje pre a proti, ktoré treba zhodnotiť a vybrať najlepšiu možnosť.

Použitím internetových aplikácií sa eliminujú nedostatky klasických aplikácií, ako napríklad potreba aktualizácie na strane klienta, správa aplikácie alebo rôzne platformy (Linux, Windows). Zaujímavý pohľad na to, kam až by internetové aplikácie mohli smerovať, poskytuje operačný systém Google Chrome OS[1], ktorý obsahuje len aplikácie priamo na internete. Tento koncept zatiaľ nepracuje so zložitými aplikáciami (strihanie videa, CAD aplikácie), ale tie využíva marginálna skupina užívateľov. Avšak toto smerovanie prináša svoje riziká v podobe centralizácie dát a výkonu, čo si vyžaduje zvýšenú bezpečnosť a v prípade výpadku služby je užívateľ bezmocný.

1.2 Relačné databázové systémy (RDBMS)

Na úvod si zadefinujme, čo je vlastne relačná databáza. „*Databáza je usporiadaná množina dát, ktorá je normálne uložená v jednom alebo niekoľkých dátových súboroch. Dáta sú štrukturované v tabuľkách s možnými referenciami medzi tabuľkami. Existencia týchto relácií viedla k pojmu relačné databázy.*“ (1, str. 34) Charakterizujú ju pojmy ako *tabuľka, záznam, pole, dotaz, index, kľúč* a iné. Oproti objektovo orientovaným databázam nám však neumožňuje priamo pracovať s objektami.

[1] `http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html`

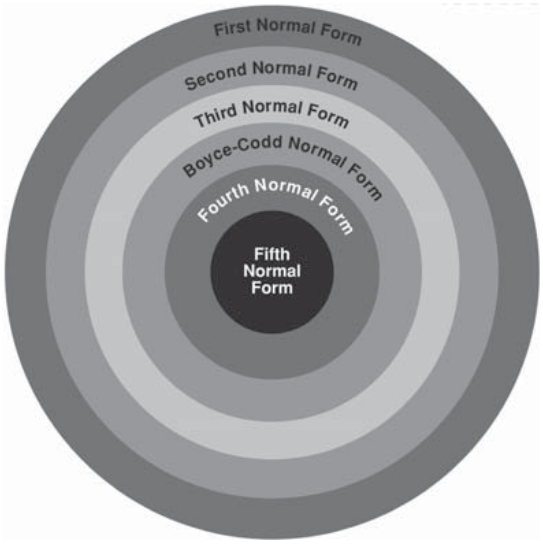
Pri výbere konkrétneho databázového systému máme na výber z mnohých riešení. Každé z nich má svoje výhody a nevýhody. Medzi tie najpoužívanejšie v súčasnosti patria:

- Oracle - momentálne jednotka na trhu RDBMS,
- Microsoft SQL Server - vynikajúca podpora Bussiness Intelligence - zložité databázové nástroje,
- MySQL - najpopulárnejšie open-source riešenie.

1.2.1 Normalizácia

Aby sme predišli základným chybám (redundancia dát a iné) pri návrhu databázového modelu, odporúča sa používať sadu niekoľkých základných pravidiel. Tieto pravidlá nazývame Normálové formy a ako prvý ich definoval vedec E. F. Codd. Keďže výskum stále pokračuje, v súčasnosti poznáme týchto 7 normálových foriem: (2, str. 206)

- prvá normálová forma (1NF),
- druhá normálová forma (2NF),
- tretia normálová forma (3NF),
- Boyce-Codd normálová forma (BCNF),
- štvrtá normálová forma (4NF),
- piata normálová forma (5NF),
- doména/kľúč normálová forma (DKNF).



Obr. 1.1: Normálové formy (2, str. 207)

Každá z nich obsahuje sadu pravidiel, ktoré je potrebné naplniť, aby model spĺňal danú formu. Prvé tri z nich sú ľahko pochopiteľné a pri ich dodržaní by sme s naším modelom nemali mať žiadne problémy. Zriedkavo je vhodné použiť BCNF, prípadne 4NF. 5NF je ťažko implementovateľná a je zložité zhodnotiť, či relácie sú v 5NF. DKNF má čisto teoretický význam a jej cieľom nie je použitie v praxi. (2, str. 207)

Prvá normálová forma preložená do formy zrozumiteľnej pre „bežného človeka“ znamená: (1, str. 192)

- musíme eliminovať stĺpce s rovnakým obsahom,
- tabuľku musíme vytvoriť pre každú skupinu príslušných dát,
- každý záznam musí byť jednoznačne identifikovaný primárnym kľúčom.

Druhá normálová forma obsahuje tieto pravidlá: (1, str. 193)

- kedykoľvek sa opakujú obsahy niektorých stĺpcov v rôznych záznamoch, tabuľka by mala byť rozdelená na niekoľko tabuliek,
- tieto tabuľky musíme prepojiť cudzími kľúčmi.

Tretia normálová forma má len jedno pravidlo: (1, str. 195)

- stĺpce, ktoré nie sú priamo závislé na primárnom kľúči, musíme odstrániť (presunúť do samostatnej tabuľky).

Aj normalizácia má svoje pre a proti a nie vždy predstavuje najoptimálnejšie riešenie. V procese tvorby modelu je potrebné zvážiť, či nám naozaj pomôže alebo

spôsobí zbytočné komplikácie pri práci s dátami - tu platí, že čím vyššia normálová forma, tým zložitejšia práca s dátami na strane klienta. Ďalší faktor, ktorý v praxi môže výrazne ovplyvniť normalizáciu je samotný zákazník. Ako príklad by som spomenul projekt, kde klient vyslovene trval na tom, aby normalizačné formy neboli dodržané. Aj keď sme všetci len krútili hlavami, museli sme mu vyhovieť.

1.2.2 MySQL

MySQL predstavuje relačný databázový systém na báze klient/server. V oblasti internetových databáz je viac-menej štandardom. Za jeho úspechom stoja nulové náklady na obstaranie (GPL licencia), nízke hardvérové nároky a dostatočná stabilita a bezpečnosť pre väčšinu aplikácií. (1, str. 47) Minuloročná akvizícia spoločnosti Oracle, ktorá kúpila spoločnosť Sun Microsystems (pod jej krídlami bol MySQL vyvýjaný) vyvolala búrlivé diskusie s predzvesťou konca tohto RDBMS, ale Oracle sa zaviazal, že vo vývoji bude pokračovať a bežní užívatelia zmenu ani nepostrehli.

Ako každý RDBMS má aj MySQL svoje výhody a nevýhody - nedostatky. Pri tých treba zvážiť, či v konkrétnom riešení nebudú obmedzujúce a podľa toho sa rozhodnúť, ktorý zvoliť. Medzi rozšírené vlastnosti MySQL patria *vnorené dotazy, pohľady, procedúry, triggery, replikácie, transakcie, GIS funkcie, podpora ODBC, nezávislosť na platforme a iné.* (1, str. 36-37). Na druhej strane sú tu obmedzenia, ktoré neumožňujú použitie v náročných enterprise systémoch vyžadujúcich napr.: OLAP (viacdimenzionálne databázy), vlastné dátové typy, online zálohy.

1.3 Skriptovacie jazyky - PHP

História PHP siaha do roku 1995, keď Rasmus Lerdorf začal pracovať na nástroji PHP/FI (Personal Homepage Tools/Form Inspector), ktorý pozostával zo kolekcie skriptov jazyka PERL, ktoré boli schopné spracovávať dáta z odoslaných formulárov. Tento jazyk sa časom zmenil na nepoznanie. Vo verzii PHP 3 sa okrem nového názvu (Hypertext processor) k Rasmusovi pridali Andi Gutmans a Zeev Suraski, ktorí hľadali jazyk pre svoje riešenie internetových obchodov. Zároveň už PHP využívalo 50 000 serverov. PHP 4 úplne zmenilo spôsob, akým boli skripty interpretované. Kdežto PHP 3 spracovávalo skriptá priamo, PHP 4 najskôr skript preložilo do strojového kódu a ten bol následne spracovaný pomocou Zend Engine. Tým sa stal jazyk podstatne rýchlejší. Posledná dôležitá zmena bola verzia PHP 5, ktorá priniesla dostatočnú podporu objektovo orientovaného kódu, čím sa otvorili nové

možnosti a jazyk sa stal použiteľný aj pre rozsiahle projekty. Zjednodušene môžme fungovanie PHP popísať nasledovne:

- klient pošle svoju požiadavku na server - tá môže obsahovať premenné,
- server skript spracuje a vyhodnotí,
- na záver odošle klientovi odpoveď.

Z toho plynie dôležitý fakt a to, že PHP je skriptovací jazyk na strane servera, čo znamená, že bežný návštevník o ňom nemusí ani vedieť. Ako príklad scriptovacieho jazyka na strane návštevníka spomeniem JavaScript.

1.3.1 OOP vs. procedurálne programovanie

„Hlavný rozdiel medzi objektovo orientovaným a funkčným programovaním je v podstate jednoduchý: v objektovo orientovanom programovaní sú dáta a kód spojené do jednej entity, ktorej sa obvykle hovorí objekt. Objektovo orientované aplikácie sú zvyčajne rozdelené do mnohých menších objektov, ktoré medzi sebou komunikujú. Každý objekt je väčšinou entitou určitého problému, ktorý je nezávislý na ostatných a je vybavený vlastnosťami a metódami. Vlastnosti sú dáta objektu. To obvykle znamená, že premenná patrí objektu. Metódy, predovšetkým pre tých, čo majú skúsenosti s funkčným, resp. procedurálnym programovaním, sú v podstate funkcie, ktoré môžu pracovať s dátami objektu.“ (4, str. 82) Pokúsim sa túto definíciu vysvetliť na jednoduchom príklade. Predstavme si, že máme užívateľa, ktorý má meno a priezvisko. V praxi nás bude zaujímať jeho celé meno.

```
<?php
// procedurálny spôsob
$uzivatel = array(
  'meno' => 'Peter',
  'priezvisko' => 'Novák'
);

function cele_meno($uzivatel)
{
  return sprintf('%s %s', $uzivatel['meno'], $uzivatel['priezvisko']);
}

echo cele_meno($uzivatel);

// OOP spôsob
class Uzivatel
```

```
{
  protected $_meno;
  protected $_priezvisko;

  public function __construct($meno, $priezvisko)
  {
    $this->meno = $meno;
    $this->priezvisko = $priezvisko;
  }

  public function celeMeno()
  {
    return sprintf('%s %s', $this->_meno, $this->_priezvisko);
  }
}

$petr = new Uzivatel('peter', 'novak');

echo $peter->celeMeno();
```

1.3.2 Objektovo orientované PHP 5

Uspokojivá podpora pre objektovo orientované programovanie prišla až vo verzii PHP 5. Prvý pokus bol aj pri verzii PHP 3 (verzia PHP 4 kvôli spätnej kompatibilite zásadné zmeny nepriniesla), ale ten mal značné nedostatky. Na vysvetlenie kompletnej syntaxe jazyka PHP by sme potrebovali priestor, ktorý nemáme, a preto uvediem len niektoré konštrukcie, ktoré priamo súvisia s OOP.

Trieda, objekt, konštruktor. Základ OOP v PHP predstavujú práve tieto tri pojmy. Trieda, resp. deklarácia triedy začína kľúčovým slovom `class` nasleduje definícia konštant, vlastností a metód. Konštruktor predstavuje špeciálnu metódu `__construct()`, ktorá sa volá pri vytvorení novej instancie objektu. V príklade vyššie sa ako parametre konštruktoru posielajú meno a priezvisko.

Magické metódy. Tieto metódy sa začínaju dvojitým podtržítkom - `__metoda()` - a preto sa nedoporučuje používať toto pomenovanie pre vlastné metódy. Každá z nich má svoj špecifický význam pri práci s objektom. Hneď prvou a najpoužívanejšou magickou metódou je samotný konštruktor. Ďalšou skupinou sú metódy `__call()`, `__get()` a `__set()`. Tieto metódy sú volané v prípade, ak voláme nedefinovanú metódu, prípadne pri pokuse získať alebo nastaviť hodnotu nedefinovanej vlastnosti. Ich použitie popisuje nasledujúci príklad, v ktorom upravíme

predchádzajúcu triedu Uzivatel. K často využívaním magickým metódam patrí ešte `__toString()`, ktorá sa volá, ak sa pokúsime vypísať priamo objekt: `echo $objekt`.

```
<?php
class Uzivatel
{
  protected $_data = array(
    'meno' => null,
    'priezvisko' => null
  );

  public function __construct($meno, $priezvisko)
  {
    $this->meno = $meno;
    $this->priezvisko = $priezvisko;
  }

  public function __set($vlastnost, $hodnota)
  {
    if (array_key_exists($vlastnost, $this->_data))
    {
      $this->_data[$var] = $val;
    }
  }

  public function __get($vlastnost)
  {
    if (array_key_exists($vlastnost, $this->_data))
    {
      return $this->_data[$vlastnost];
    }
  }

  public function __toString()
  {
    return sprintf('%s %s', $this->meno, $this->priezvisko);
  }
}

$peter = new Uzivatel('peter', 'novak');

echo $pepter->meno; // -> peter
echo $peter; // -> peter novak
```

Aj keď sa môže zdať, že tento pristúp je zložitejší, jeho uplatnenie popíšem pri objektovo relačnom mapovaní, kde je v ešte prepracovanejšej forme použití. Samozrejme

existujú aj ďaľšie magické metódy, ale tie pri tejto práci nemajú priamy význam.

Modifikátory prístupu. To, že všetky spomínané charakteristiky OOP implementácie v PHP sa vzájomne dopĺňajú, dokazujú modifikátory prístupu. „*Kľúčovým vzorom objektovo orientovaného programovanie je zapuzdrenie a ochrana prístupu k vlastnostiam objektu (často označovanými ako členské premenné). Väčšina bežne používaných programovacích jazykov má k dispozíci tri hlavné modifikátory prístupu: public, protected a private.*“(4, str. 86) Každý z nich upravuje prístup k objektu nasledovne: (4, str. 86)

- `public` - k verejným metódam a vlastnostiam máju prístup všetky objekty,
- `private` - privátne je možné volať len v rámci konkrétnej triedy a to pomocou automatickej premennej `$this`, alebo `self`,
- `protected` - chránené metódy a vlastnosti sú okrem objektov svojej triedy prístupné aj objektom všetkých odvodených tried.

Výnimky. „*Ošetrenie výnimiek je jedným z najproblematickejších aspektov vývoja softvéru. Náročné je už rozhodnutie, čo robiť ak k chybe dôjde, ale náročné je taktiež nájdenie všetkých miest, kde k chybám môže dôjsť. Takéto miesta je potom potrebné zabezpečiť kódom pre overenie chýb. ... Ešte obtiažnejšie je vytvoriť kód, ktorý chybu napraví a umožní pokračovanie v behu programu.*“(4, str. 100) K zachytávaniu výnimiek sa používa konštrukcia `try { ... } catch { ... }`. Pri práci s výnimkami si však veľa ľudí neuvedomuje, že PHP výnimky nepreposiela ďalej, čo znamená, že ak sa výnimka nezachytí priamo pri volaní danej metódy, ale pri volaní metódy, ktorá volá metódu s výnimkou, tá nebude zachytená. Na tento fakt zabúda aj odborná literatúra a pre lepšie pochopenie uvediem príklad.

```
<?php
class Priklad
{
  public function metodaA()
  {
    throw new Exception();
  }

  // metóda B len volá metódu
  public function metodaB()
  {
    $this->metodaA();
  }
```

```
}

$test = new Priklad();

// tento blok výnimku zachytí
try
{
  $test->metodaA();
}
catch (Exception $e)
{
  echo 'chyba';
}

// tento blok výnimku nezachytí
try
{
  $test->metodaB();
}
catch (Exception $e)
{
  echo 'chyba';
}
```

Aj toto je jednou z príčin, prečo práca s výnimkami patrí k mimoriadne obtiažnym.

Statické metódy a vlastnosti. Trieda môže taktiež obsahovať statické vlastnosti a metódy. Statické vlastnosti nepatria instanciám triedy (objektom), ale samotnej triede. Môžme ich považovať za globálne vlastnosti dostupné pre všetky instancie triedy. Na rovnakom princípe fungujú aj statické metódy. Pri tých si treba dať pozor, že nie sú nijak spojené s objektom danej triedy a nie je možné v nich použiť premennú `$this`. Statické metódy a vlastnosti definujeme kľúčovým slovom `static` a pristupujeme k nim buď `NazovTriedy::$vlastnost, NazovTriedy::metoda()`, alebo `self::$vlastnost, self::metoda()` v rámci triedy. `self` je ekvivalent `$this` pre prístup k triede.

Polymorfizmus. „*Téma polymorfizmu je v objektovo orientovanom programovaní pravdepodobne najdôležitejšia. Pomocou tried a ich delenia sa dajú jednoducho popisovať situácie v skutočnom živote. Triedy a ich instancie nie sú tak len kolekciami dát a funkcií. Vďaka existencii tried sa môžu projekty ľahko a spoľahlivo rozrastať, pretože prvé objekty vznikají na už preverených základoch. Hotový kód je znovu používaný predovšetkým vďaka existencii mechanizmu delenia.*“(4, str. 92) Polymorfizmus

nám umožňuje rozširovať a upravovať už hotové triedy a zmeniť ich tak v náš obraz veľmi jednoducho. V PHP reprezentuje polymorfizmus kľúčové slovo `extends`.

```
<?php
class Clovek
{
}

class Zena extends Clovek
{
}
```

Význam OOP. Na záver tohto stručného sprievodcu OOP v PHP uvediem niekoľko osobných postrehov. PHP je jazyk relatívne jednoduchý, čomu zodpovedá jeho veľká popularita. Tá jednak prispieva k stálemu rozvoju, no na druhej strane vzniká aj veľké množstvo nekvalitného kódu, čo vytvára nie veľmi pozitívny obraz u odbornej verejnosti. Nakoľko tento jazyk nemá zatiaľ veľkú podporu ani v školstve, väčšina programátorov sú samouci. To by problém nebol, ale svoje vedomosti získavajú prevažne z internetu a nie odbornej literatúry, čo vedie k zlým návykom už od začiatku a tie sa potom len ťažko odstraňujú. Mal som možnosť byť súčasťou prijímacieho konania PHP programátorov a hodnotiť ich riešenia testovacieho zadania. Bol som až prekvapený, že takmer nikto nedokázal pomerne jednoduché zadanie vypracovať podľa našich predstáv. Práve nesprávne pochopenie základných princípov OOP bolo najčastejšou chybou uchádzačov.

1.4 MVC architektúra - Zend Framework

V prípade, že sa rozhodneme programovaniu venovať naplno, časom zistíme, že pri zložitejších problémoch si nevystačíme len s perfektnou znalosťou syntaxe vybraného jazyka, aj keď tá tvorí pevný základ našich schopností a v žiadnom prípade ju nesmieme podceniť alebo zanedbať.

1.4.1 Návrhové vzory

„Pri návrhu softvéru sa nevyhnete opakovaniu určítých programovacích vzorov. Niektoré z nich boli komunitou návrhárov vyriešené a pre tieto vzory existuje všeobecne prijateľné riešenie. Takýmto stále sa opakujúcim problémom hovoríme bežne ***návrhové vzory****. Výhodou ich znalosti a využívanie je nielen úspora času, ale taktiež*

spoločný jazyk pri návrhu softvéru.“ (4, str. 116) K tým najčastejšie používaným patria **Singleton (Jedináčik)** a **Factories (Vzor výrobnej triedy)**.

Vzor Singleton umožnuje prístup k jedinej instancii triedy naprieč celou aplikáciou. Svoje uplatnenie si našiel napríklad pri uchovaní databázového pripojenia a jeho implementácia je pomerne jednoduchá.

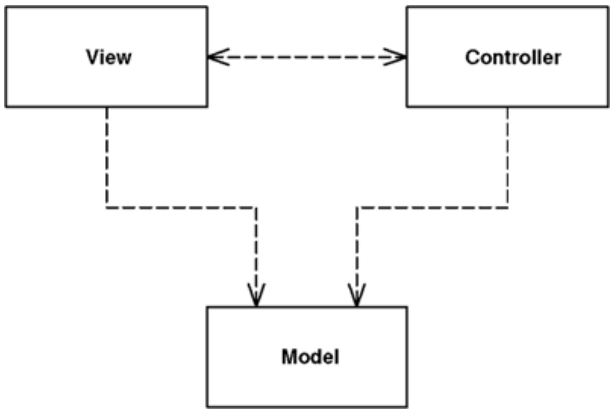
```
<?php
class DatabazovePripojenie
{
  protected static $_instance == NULL;

  public static function getInstance()
  {
    if (self::$_instance === NULL)
    {
      self::$_instance = new self();
    }

    return self::$_instance;

  // nasledujú metódy pre prácu s pripojením
  ...
}
```

Takto môžme k pripojeniu pristupovať v celej aplikácii pomocu statickej metódy `DatabazovePripojenie::getInstance()`.

Pri tvorbe nových objektov je niekedy potrebné, hlavne v zložitejších aplikáciách, kontrolovať tento proces. Napríklad, v rámci kontextu celej aplikácie chceme použiť na niektorých miestach vlastné triedy a to bez úprav samotného kódu. Práve v tejto fáze sa nám hodí vzor výrobnej triedy, ktorý nám dáva kontrolu nad týmto procesom.

1.4.2 Model - View - Controller

K najpopulárnejším návrhovým vzorom v oblasti návrhu webových aplikácii patrí **MVC: Model - View - Controller**.



Obr. 1.2: Návrhový vzor MVC (5, Kapitola 14)

„*MVC berie do úvahy tri role. Model je objekt, ktorý reprezentuje informácie o doméne. Je to nevizuálny objekt obsahujúci všetky dáta a správanie. V jeho najčistejšej objektovo orientovanej forme je Model objekt v Doménovom modeli. Pohľad (View) reprezentuje zobrazenie Modelu v užívateľskom prostredí. Napríklad, ak je náš Model objekt Zákazník, Pohľad môže byť HTML stránka s informáciami o Modeli. Pohľad je len o zobrazovaní informácií, akékoľvek zmeny týchto informácií sú spracované tretím členom MVC trojice: Controllerom. Controller spracúva vstupy od užívateľa, manipuluje s Modelom a patrične upravuje Pohľad.*“ (5, Kapitola 14)

Pomocou tohto prístupu jednoznačne oddelíme aplikačnú logiku (Controller), biznis logiku (Model) a samotnú prezentáciu (View). Tým sa náš kód stane čistejším, prehľadnejším a s orientáciou v ňom nebude mať problém ani nezainteresovaný človek, ktorý sa na samotnom vývoji nepodieľal.

Implementáciu MVC v jazyku PHP poskytuje množstvo Frameworkov, pomocou ktorých sa vývoj stáva dynamický, s jasnými pravidlami a niekede sa stretávame až s pojmom „rapid application development - rýchly vývoj aplikácií“. Okrem samotného MVC tieto Frameworky väčšinou poskytujú bohatú knižnicu komponentov, ktoré vývoj ešte viac urýchľujú a pri tých známejších máme k dispozícii aj bohatú komunitu vývojárov a užívateľov. Na českom internete si svoju popularitu získal Framework **Nette**[2], hlavne pre svoju jednoduchosť a lokálnu komunitu. Z tých komplexnejších sú to napríklad **symfony**[3] a **Zend Framework**[4]. Osobne preferujem a používam framework symfony, ale ako neskôr vysvetlím, pre tento projekt nebol vhodný a preto dostal prednosť Zend Framework. A práve výber správneho, resp.

[2] `http://nette.org/cs/`

[3] `http://www.symfony-project.org/`

[4] `http://framework.zend.com/`

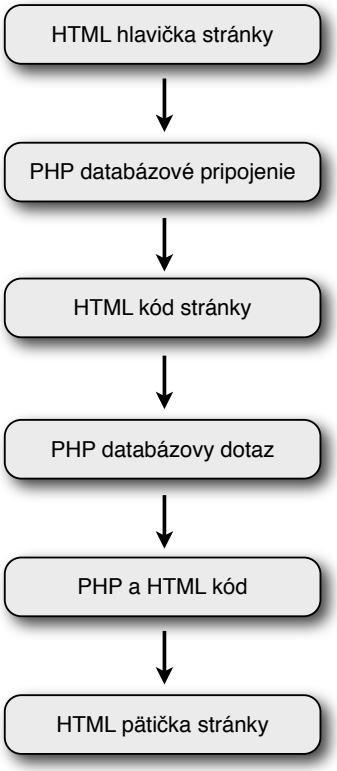
nesprávneho frameworku môže sposobiť zbytočné komplikácie pri vývoji aplikácie. Je úplne zbytočné použiť robustný framework ako napríklad symfony na jednoduché CMS, prípadne prezentačné stránky. Tu sa hodí prirovnanie kanónom na vrabca.

1.4.3 Zend Framework

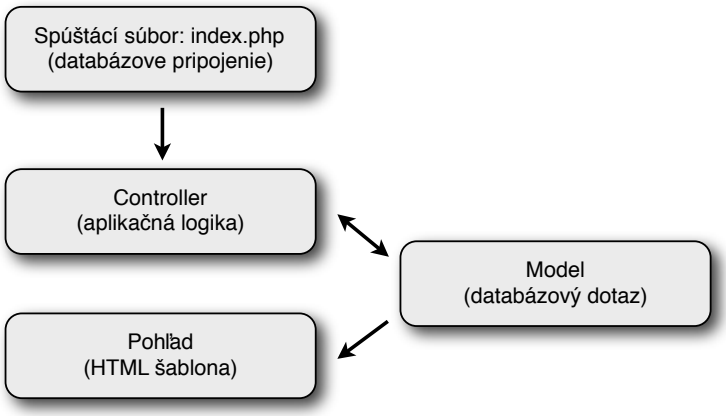
„*Zend Framework je PHP knižnica pre tvorbu webových PHP aplikácií. Jednotlivé komponenty spoločne poskytujú plnohodnotný framework so všetkými komponentami potrebnými k tvorbe moderných, ľahko udržateľných aplikácií.*“ (6, str. 9)

Za týmto frameworkom, ako už názov napovedá, stojí spoločnosť Zend, ktorá sa podieľa aj na vývoji samotného PHP. Jednou z jeho výhod predstavuje možnosť využívať jednotlivé komponenty samostatne, bez nutnosti postavenia jadra aplikácie na tomto frameworku. Asi najväčší nedostatok predstavuje slabá podpora pre prácu s Modelom. Na prácu s ním využíva návrhové vzory **Table Data Gateway** (5, Kapitola 10) , a **Row Data Gateway** (5, Kapitola 10). Tieto návrhové vzory poskytujú len základné rozhranie pre prácu s dátami (CRUD). Tento nedostatok môže byť eliminovaný použitím niektorej z dostupných ORM knižníc (Doctrine, Propel).

Zend Framework je postavený na návrhovom vzore MVC a tu by bolo vhodné ilustrovať odlišnosť štruktúry klasickej PHP stránky a aplikácie využívajúcej tento návrhový vzor.



Obr. 1.3: Klasická štruktúra PHP súboru bez použitia MVC. (6, str. 4)



Obr. 1.4: Štruktúra pri využití MVC. (6, str. 5)

Práve bohatá knižnica hotových komponentov rozhodla, že svoju aplikáciu postavím práve na Zend Frameworku a nie na symfony. Hlavnú úlohu zohrala podpora pre webovú službu AMF a podpora PDF. Predtým ako popíšem niektoré, pre prácu významné komponenty, považujem za dôležité oboznámenie s adresárovou štruktúrou projektov využívajúcich Zend Framework. Práve tá je pri každom frameworku špecifická a jasne definuje usporiadanie kódu.

zložka	**popis**
`<project>`	koreňová zložka projektu
`/application`	obsahuje samotnú aplikáciu a všetky súčasti MVC
`/configs`	konfiguračné súbory a nastavenia
`/controllers`	triedy Controllerov
`/services`	služby
`/models`	modely
`/views`	HTML šablony
`/data`	táto zložka poskytuje priestor na uskladnenie aplikačných dát
`/docs`	dokumentácia k aplikácii
`/library`	adresár pre spoločné knižnice aplikácie
`/Zend`	knižnica Zend Frameworku
`/public`	jediný verejný adresár, obsahuje index.php, ktorý spúšťa aplikáciu
`/tests`	priestor pre aplikačné testy

Tabuľka 1.1: Adresárová štruktúra Zend Framework projektu

A na záver spomínané kľúčové komponenty Zend Frameworku, ktoré boli využité v riešení tejto práce.

Zend_Mail. Aj keď PHP obsahuje podporu pre posielanie emailových správ a každý PHP programátor sa už s funkciou `mail()` stretol, podpora pre protokoly ako SMTP je závísla od kompilácie konkrétnej verzie na servery a aj posielanie správ obsahujúcich viac častí, prípadne súborových príloh, už nie je také jednoduché. Komponent Zend_Mail s využitím objektového prístupu tvorbu a posielanie správ výrazne uľahčuje. Ukážeme si to na príklade, v ktorom vytvoríme a odošleme jednoduchú správu.

```
<?php
// vytvorenie správy
$mail = new Zend_Mail();
// nastavenie odosielateľa
$mail->setFrom('solzenic@symfony.sk', 'Viliam Husár');
// pridanie adresáta
$mail->addTo('jozef@gmail.com', 'Jozef Novák');
// nastavenie predmetu správy
$mail->setSubject('Skúšobná správa');
// nastavenie obsahu správy
$mail->setBodyText('Ahoj, testujem Zend Mail.');
// odoslanie správy
```

```
$mail->send();
```

Zend_Amf_Server. Formátu AMF podrobnejšie popisuje až nasledujúca podkapitola, ale práve jeho podpora v Zend Frameworku bola pri výbere kľúčová. V podstate ide o klasickú webovú službu, podobne ako SOAP, kde komponent Zend_Amf_Server vytvára vstupnú bránu pre AMF požiadavky (ekvivalent SOAP požiadavkám) a mapuje zvolené triedy, ktoré tieto požiadavky spracuvávajú. V kontexte aplikácie sa umiestňuje do ľubovoľného Controlleru, napríklad takto.

```
<?php
require_once(APPLICATION_PATH . '/services/amf/Core.php');
require_once(APPLICATION_PATH . '/services/amf/Directory.php');

class AmfController extends Zend_Controller_Action
{
  public function init()
  {
    // Pohľad (View), je v˜tomto prípade nežiadúci
    $this->getHelper('ViewRenderer')->setNoRender();
  }

  public function indexAction()
  {
    // vytvorenie AMF Servera
    $server = new Zend_Amf_Server();

    // namapovanie dostupných služieb
    $server->setClass('Application_Service_Amf_Core')
      ->setClass('Application_Service_Amf_Directory');

    // spracovanie požiadavky
    echo($server->handle());
  }
}
```

1.5 Objektovo relačné mapovanie - Doctrine

Ale vrátme sa späť k RDBMS. To, že samotné relačné databázy neumožňujú ukladanie objektov už vieme, ale ani tento problém nie je neriešiteľný.

„Objektovo relačné mapovenie je technika použitá v programovacích jazykoch pri riešení nekompatibility dátových typov v relačných databázach. ORM v podstate umožňuje, aby sme mali „virtuálnu objektovú databázu“, ktorá môže byť použitá

z programovacieho jazyka.“ (10, str. 13). V praxi to znamená, že medzi samotným kódom a relačnou databázou sa nachádza ďalšia medzivrstva, ktorá zabezpečuje automaticky mapovanie medzi objektom a záznamom databázy, čím výrazne uľahčuje prácu. Pokročilejšie spracovania ORM dokonca úplne izolujú databázová vrstvu, čo má za následok nezávislosť kódu na databázovom systéme. Čiže ak sa časom zistí, že pôvodný RDBMS systém už nepostačuje, nie je problém ho zmeniť, a to bez zásahu do samotnej aplikácie.

1.5.1 Doctrine

Doctrine je ORM framework pre PHP postavený prevažne na návrhových vzoroch **ActiveRecord**[5], **DataMapper**[6], **UnitOfWork**[7] a **Meta Data Mapping**[8]. *„Použitie vzoru ActiveRecord prináša typické obmedzenia. Najviditeľnejším je nutnosť rozširovať špecifickú základnú triedu (Doctrine_ Record). Vo všeobecnosti je návrh doménového modelu značne limitovaný dizajnom relačného modelu. Hoci aj tu je výnimka. Pri práci s dedičnosťou Doctrine poskytuje niekoľko sofistikovaných mapovacích stratégií, ktoré umožňujú odlíšiť relačný model a poskytnúť tak viac slobody.“* (10, str. 15). Zrejme najlepšie uvidíme ako Doctrine funguje na príklade. Použijeme už známu triedu `Uzivatel`, ktorá ale tentoraz bude využívať Doctrine.

```
<?php

$peter = new Uzivatel();
$peter->setMeno('Peter');
$peter->setPriezvisko('Novák')
$peter->save();

echo $peter->getMeno(); // Peter
echo $peter->getCeleMeno(); // Peter Novák

class Uzivatel extends Uzivatel_Base
{
  public function getCeleMeno()
  {
    return sprintf('%s %s', $this->getMeno(), $this->getPriezvisko);
  }
}
```

[5] http://www.martinfowler.com/eaaCatalog/activeRecord.html
[6] http://www.martinfowler.com/eaaCatalog/dataMapper.html
[7] http://www.martinfowler.com/eaaCatalog/unitOfWork.html
[8] http://www.martinfowler.com/eaaCatalog/metadataMapping.html

```
class Uzivatel_Base extends Doctrine_Record
{
  public function setTableDefinition()
  {
    $this->setTableName('uzivatel');

    $this->hasColumn('meno', 'string', 50, array(
      'type' => 'string',
      'length' => 50,
      'notnull' => true,
    ));

    $this->hasColumn('priezvisko', 'string', 50, array(
      'type' => 'string',
      'length' => 50,
      'notnull' => true,
    ));
  }
}
```

V tomto príklade sú použité až tri triedy, ktoré postupne dedia jedna od druhej. V praxi zaujíma programátora len tá najvyššia (`Uzivatel`). Do tej pridáva potrebnú biznis logiku, špecifickú pre danú aplikáciu. V našom prípade to bola metóda `getCeleMeno()`. Základná trieda Uzivatel_Base obsahuje práve potrebné mapovanie konkrétnej relácie, definovanie polí, vzťahov medzi reláciami, prípadne ďaľšie rozširujúce možnosti (dedičnosť). A všetko ostatné zabezpečuje trieda Doctrine_Record. Všimnime si, že nikde nie sú definované metódy ako `setPriezvisko()`, `getPriezvisko()`, a práve v tom spočíva kúzlo Doctrine. Tu sa vraciame späť k magickej metóde `__call()`, ktorá práve takéto chovanie umožňuje. Nijak nám však nebraní takúto metódu do triedy `Uzivatel` pridať a modifikovať tak proces nastavovania danej vlastnosti.

Možnosti, ktoré Doctrine pri práci s databázou poskytuje, sú rozsiahle a je obtiažne zhrnúť ich do niekoľkých odstavcov. Pokúsil som sa preto vybrať aspoň tie najzákladnejšie, s ktorými sa stretneme aj v riešení tejto práce.

DBAL - Database Abstraction Layer. Databázová abstraktná vrstva predstavuje spomínanú medzivrstvu medzi databázou a ORM. Jej úlohou je poskytnúť rozhranie pre prácu s objektami pri zachovaní nezávislosti použitého databázového systému. Stará sa o mapovanie, či už dátových typov alebo dotazov, podľa zvoleného RDBMS. Okrem toho je nápomocná pri migrovaní živých databáz (zmena atribútov).

DQL - Doctrine Query Language. „*Dotazovací jazyk Doctrine je objektový dotazovací jazyk vytvorený s účelom pomôcť užívateľom pri získavaní komplexných objektov. Mal by byť použitý vždy pri získávaní relačne prepojených objektov.*“ (10, str. 120). Prácu s ním uľahčuje trieda `Doctrine_Query`, ktorá uľahčuje vytváranie týchto dotazov. V nasledujúcom príklade porovnáme SQL dotaz s DQL dotazom vytvoreným pomocou `Doctrine_Query`. V príklade je použitý model pozostávajúci z objektov User (užívateľ) a Phonenumber (telefóne číslo) s tým, že jeden užívateľ má niekoľko telefónnych čísel.

```
<?php
$q = Doctrine_Query::create()
  ->select('u.*, p.*') // výber všetkých vlastností
  ->from('User u') // z objektov User
  ->leftJoin('u.Phonenumbers p'); // a pripojenie objektov Phonenumbers

// vypísanie SQL dotazu
echo $q->getSqlQuery();
?>

SELECT u.id AS u__id,
  u.id as u__id,
  u.first_name AS u__first_name,
  u.last_name AS u__last_name,
  p.id AS p__id,
  p.user_id AS p__user_id,
  p.phonenumber AS p__phonenumber
FROM user u
LEFT JOIN phonenumber p ON u.id = p.user_id
```

YAML - Ain't Markup Language. Jazyk YAML[9] zohráva svoju úlohu pri návrhu nášho modelu. Keď sa vrátime k triede `Uzivatel_Base` a predstavíme si podstatne zložitejší objekt, bolo by pracné pre každy objekt ručne vytvárať takúto základnú triedu. Oveľa jednoduchšie by to bolo napríklad takto:

```
# schema.yml
Uzivatel:
  columns:
    meno:
      type:        string(50)
    priezvisko:
      type:        string(50)
```

[9] `http://yaml.org/`

Takto definovaný model dokáže Doctrine spracovať a vytvoriť základné triedy za nás, čím nám dáva viac času na kvalitné navrhnutie modelu.

Listeners. Každú triedu môžme rozšíriť o špeciálne metódy, ktorých úlohou je nahradiť databázové trigre. Sú to napríklad metódy: `preSave()`, `postSave()`, `preInsert()`, `postInsert()`, `preUpdate()`, `postDelete()` a ako už názvy napovedajú, sú volané pri vybraných operáciách s objektom (vytvorenie, uloženie, zmazanie...). Tie je možné zoskupiť aj do vlastnej triedy (Listener) a tým umožniť ich reťazenie.

Behaviours. Často sa pri návrhu databázového modelu stretávame s opakujúcimi sa podobnosťami. Napríklad, chceme zachovávať informácie, kedy bol záznam vytvorený a upravený. Z toho, čo doteraz vieme, by sme pre túto úlohu mohli použiť, napríklad metódy `preInsert()` a `preUpdate()`, kde by sme tieto hodnoty aktualizovali. Ale existuje aj elegantnejšie riešenie v podobe použitia Behaviours (správania). Behaviour je v podstate samostatná trieda, ktorá má možnosť upraviť samotný model - pridať vlastné atribúty (dátum vytvorenia, dátum upravenia) a nastaviť Listeneri (aktualizácia časových známok pri uložení a vytvorení). Konkrétne takéto „správanie“ už Doctrine obsahuje (Timestampable (10, str. 249)). Behaviour je potrebné nastaviť v základnej triede alebo pomocou YAML v schéme.

```
# schema.yml
Uzivatel:
  actAs:
    Timestampable:
  columns:
    meno:
      type:         string(50)
    priezvisko:
      type:         string(50)
```

Hydrators. Nie vždy však potrebujeme z databázy mať prístupný celý objekt. Často si vystačíme aj s obyčajným poľom, prípadne len s jednou jedinou hodnotou. Aby sme zbytočne v týchto prípadoch nevyťažovali pamäť novými objektami, poskytuje Doctrine možnosť jednotlivé objekty hydratovať a teda upraviť získané záznamy do prijateľnejšej formy bez nutnosti vytvárať objekty.

1.6 Bohaté internetové aplikácie - Flex

„Keď Macromodia v roku 2002 vydala Flash MX, bol označený za nový spôsob budovania Bohatých Internetových Aplikácií (Rich Internet Applications - RIA). Tento termín vymysleli v Macromedii a popisuje novú triedu internetových aplikácií, ktorá poskytuje výhody pripojenia na internet, vrátane prístupu k rôznym webovým službám, ale rieši veľa nepríjemných problémov vychádzajúcich z limitujúcich možností prehliadača. Použitím Flash Playeru na hosťovanie graficky bohatých aplikácií v podobe Flash dokumentu, problémy ako rozdiely medzi prehliadačmi v aplikácii kaskádových štýlov (CSS) a JavaScriptu, boli minulosťou.“ (7, str. XXV) Hlavnú výhodu však predstavuje možnosť asynchronnej komunikácie medzi klientom a serverom. Kďežto pri klasických stránkach používame model požiadavka/odpoveď a pri každej akcii sa načítava celá stránka, pri asynchronnej komunikácii sa aktualizujú len vybrané časti užívateľského prostredia.

V súčasnosti sa do popredia dostáva nová verzia HTML5[10], ktorá prináša mnohé vylepšenia a natívnu podporu multimédií bez nutnosti doplnkov ako Flash Player. Tým dáva do ruky silné argumenty odporcom Flashu, ktorý tejto technológii predpovedajú skorý zánik. Ich tvrdenia však treba brať s rezervou, pretože ako sme už naznačili, Flash nie je len o videu a animáciách. V tomto ohľade s nimi plne súhlasím, tu je už Flash pomaly minulosťou, ale sú tu oblasti ako práve Bohaté Internetové Aplikácie, kde si svoje uplatnenie určite ešte pár rokov zachová. Tiež nesmieme zabudnúť, že HTML5 je stále vo fáze schvaľovania a môže sa meniť.

1.6.1 Flex

Platforma Flex poskytuje rozšírenú sadu nástrojov pre tvorbu bohatých internetových aplikácií a definuje nový koncept tvorby Flashových súborov. Tie už netvoria len poprepájanú sadu jednotlivých obrazcov (frames) doplnených o efekty a animácie. Flash tu zastáva viac-menej len publikačnú vrstvu, pod ktorou sa nachádza plnohodnotná aplikácia s vyspelým užívateľským prostredím a prostriedkami pre komunikáciu so zdrojmi dát.

„Dva základné prostriedky Flex frameworku sú MXML a ActionScript 3.0. MXML predstavuje XML slovník, pomocou ktorého môžu vývojári popísať užívateľské prostredie. MXML obsahuje množinu tagov a elementov, ktoré korešpondujú triedam ActionScriptu. Pri vytváraní SWF súboru sú MXML elementy skompilované do zodpo-

[10] `http://dev.w3.org/html5/spec/Overview.html`

vedajúceho kódu ActionScriptu. Atribúty každého MXML tagu korešpondujú s vlastnosťami a metódami ActionScript triedy.“ (8, str. 6) Tým sa stáva vývoj aplikácie efektívnejším a rýchlejším. Väčšinou sa stretávame s kompromisom, kde užívateľské prostredie je definované pomocou MXML a funkčnosť napísaná v ActionScripte. Na ukážku jednoduchá aplikácia s jedným tlačítkom, ktoré po kliknutí zmení svoj popis a zablokuje sa.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">

  <mx:Button
    id="btn1"
    x="10"
    y="10"
    label="Stlač ma!"
    click="this._buttonClicked()" />

  <mx:Script>
    <![CDATA[
    public function _buttonClicked():void
    {
      this.btn1.label = 'Ďakujem.';
      this.btn1.enabled = false;
    }
    ]]>
  </mx:Script>
</mx:Application>
```

Ako vidíme tlačítko je definované pomocom MXML tagu `<mx:Button />` a jeho funkcionalita v ActionScripte metódou `_buttonClicked()`. Tento princíp je veľmi podobný XHTML a JavaScriptu.

Medzi známe portály v zahraničí, ale aj u nás, ktoré technológiu Flex využívajú, patria napríklad: SAP BusinessObjects[11], Philips Lighting[12], huste.tv[13].

[11] `http://www.adobe.com/cfusion/showcase/index.cfm?event=casestudydetail&casestudyid=773403&loc=en_us`

[12] `http://www.adobe.com/cfusion/showcase/index.cfm?event=casestudydetail&casestudyid=756406&loc=en_us`

[13] `http://www.huste.tv/`

1.6.2 AMF

„Action Message Format je kompaktný binárný formát, ktorý slúži na serializovanie ActionScript objektov.“ (11) Pomocou tohto formátu je možné medzi dvoma koncovými bodmi prenášať aj zložené objekty. Pod koncovými bodmi si môžme predstaviť napríklad Flex aplikáciu a PHP aplikáciu, ktoré si takto vzájomne vymieňajú dáta.

1.7 LaTeX

Medzi najväčšie problémy programátorov a HTML kóderov patria dynamické PDF dokumenty. Existujú síce kvalitné a prepracované knižnice (mPDF[14]), ktoré dokážu relatívne vierohodne previesť HTML do podoby PDF, ale vyžadujú značnú dávku trpezlivosti na strane kódera. Keďže moje zručnosti v tejto oblasti nepatria k excelentným, rozhodol som sa použiť nástroj, ktorý napriek svojmu veku stále patrí k špičke v oblasti sádzania dokumentov a nezáleži na tom, či ide o obsiahlu monografiu alebo faktúru.

Na rozdiel od klasických nástrojov pre sádzanie dokumentov (Adobe InDesign[15], QuarkXPress[16]), ktoré pracujú na princípe WYSIWYG, TeX a jeho rozšírenie LaTeX pracuje čisto s textom. V praxi to znamená, že ten, čo text píše sa nemusí zaoberať vzhľadom dokumentu, ale svoje úsilie venuje plne obsahu. K publikáciám, kde LaTeX stále patrí k najpoužívanejším, patria akademické práce, technická literatúra a ako zaujímavosť by som spomenul vývesky Dopravného podniku mesta Brna.

Pri zložitejších dokumentoch sa využívajú vlastné šablóny a makrá. Šablóny definujú vzhľad dokumentu, resp. spôsob ako budú sádzané jednotlivé prvky (nadpisy, paragrafy, obrázky) a makrá rozširujú funkcionalitu napríklad o podporu tvorby prezentácií alebo sádzanie matematických vzorcov.

1.8 Ďalšie možnosti spracovania

Nesmieme zabudnúť ani na ďalšie technológie, pomocou ktorých by bolo možné spracovať podobný typ úloh. Či už samostatne alebo využiť hotové profesionálne riešenie.

Vlastnú alternatívu k PHP, MySQL a Flex má aj Microsoft v podobe ASP.NET, MS SQL a SilverLight. Tie v niektorých aspektoch disponujú širšími možnosťami,

[14] `http://mpdf.bpm1.com/`

[15] `http://www.adobe.com/products/indesign.html`

[16] `http://www.quark.com/Products/QuarkXPress/`

ale tie sú patrične ocenené a len náklady spojené s ich prevádzkou (licencie) by znamenali značnú finančnú investícu, ktorú si spoločnosť pri takomto projekte nemohla dovoliť.

Druhú možnosť zastupujú hotové riešenia v podobe CRM systémov. Tie síce obsahujú všetku potrebnú funkcionalitu, no skôr sa hodia do väčších organizácií, kde je možné ich potenciál plne využiť. Pri ich výbere je potrebné počítať s istou všeobecnosťou a zvýšenými nákladmi v prípade úprav na požiadanie. Inými slovami, takéto aplikácie sú príliš zložité na úlohy podobné tejto bakalárskej práci.

2 Analýza súčasnej situácie

2.1 História spoločnosti

Spoločnosť VHML, spol. s r.o. vznikla na začiatku roku 2007 ako plánovaný nástupca fyzickej osoby Viliam Husár - VHML Corporation. Za tri roky svojej pôsobnosti postupne prebrala všetky jej činnosti a na konci roku 2009 bola fyzická osoba zrušená. Preto keď hovoríme o histórii, je potrebné zahrnúť aj obdobie pred rokom 2007.

Za deväť rokov pôsobenia (od roku 2000) si vybudovala stabilné portfólio svojich produktov a služieb, ktorých podiel na obrate sa menil podľa požiadaviek trhu. V úplnom začiatku (2000 - 2002), po získaní potrebnej akreditácie, išlo hlavne o školenia obsluhy motorových vozíkov a technické prehliadky týchto zariadení. Táto činnosť nemala veľké náklady a poskytla dostatok finančných prostriedkov na zabezpečenie ďalšieho rozvoja. Postupne sa pridal servis a predaj manipulačnej techniky, ktoré sa až do roku 2007 stali hlavnými činnosťami. S rastom ekonomiky, ktorá sa v tom období nič netušiac blížila k svojmu vrcholu, rástli aj investície a čoraz väčší počet firiem malo záujem o novú techniku (hlavne značky, ktoré majú u nás autorizovaný predaj a servis). Ako nevyhnutný následok, záujem o servis staršej techniky začal klesať, a bolo potrebné zamerať sa na niečo nové. Rozhodnutie padlo práve na priemyselné pneumatiky, ktoré začala predávať už pod hlavičkou novej spoločnosti VHML, spol. s r.o.

2.2 Portfólio

Ako už z predchádzajúcej časti vyplýva hlavnou oblasťou činnosti je manipulačná technika a služby s ňou spojené. V oblasti servisu a predaja poskytuje tieto služby hlavne pre staršiu techniku a taktiež začala ponúkať čoraz populárnejšie ojazdené vozíky moderných značiek. Tento trh sa javí veľmi perspektívny.

Školenia obsluhy manipulačnej techniky vykonáva vo svojom akreditovanom školiacom stredisku. Pravidelne sa tu uskutočňujú základné a opakovacie školenia s vydaním preukazu vodiča, ktorý má platnosť vo všetkých krajinách Európskej únie.

Priemyselné pneumatiky a kolesá predáva od roku 2007 ako autorizovaný predajca švédskeho koncernu Trelleborg s výhradným zastúpením pre najnovšie modely. Zameriava sa hlavne na kolesá a pneumatiky pre manipulačnú techniku, ale výnimkou nie sú ani poľnohospodárke stroje a stavebné mechanizmy. V tejto oblasti poskytuje okrem dodávky aj samotnú montáž pomocou mobilného lisu priamo

u zákazníka.

Práve priemyselné pneumatiky a kolesá a ich ponuka zákazníkom sú predmetom tejto bakalárskej práce, a preto si o nich povieme trochu viac. Koncern Trelleborg zoskupuje niekoľko rôznych značiek, z ktorých každá má niekoľko modelov kolies a každý model má špecifické parametre (rozmer, tvrdosť, použitá guma, dezén ai.). Ich kombináciou vzniká nespočetné množstvo rôznych kolies. Špecifické parametre si vyžadujú špecifické riešenia a použite hotového softvéru by bolo problematické.

2.3 Súčasná situácia spracovania cenových ponúk

Nakoľko už spoločnosť rok program používa, opíšem v tejto časti proces, akým boli cenové ponuky a cenníky tvorené a posielané zákazníkom pred nasadením programu a ktoré problémy bolo cieľom eliminovať.

Formálna stránka - vzhľad. Pri cenových ponukách boli na začiatku vytvorené vzorové cenové ponuky vo formáte Microsoft Word, a tie boli podľa potreby upravené a nanovo uložené. Stálou modifikáciou tieto ponuky postupne prestali mať podobu pôvodných a po formálnej stránke neboli pozitívnou vizitkou spoločnosti. V prípade cenníkov bola situácia o niečo lepšia, nakoľko tie boli tvorené pomocu Microsoft Excel s využitím Visual Basic for Applications.

Časová náročnosť. Vďaka podpore základnej počítačovej gramotnosti v celej sústave školských inštitúcií, počnúc základnými školami, už práca s textovými a tabuľkovými procesormi (Microsoft Word, Microsoft Excel) nerobí súčasnej mladej generácii problém. To však neplatí o staršej generácii, pre ktorú práca s nimi môže byť zložitá a časovo náročná. Vytvorenie cenovej ponuky trvalo bežne aj pol hodiny a v prípade, že ich obchodník mal vytvoriť väčšie množstvo, strávil pri tom niekoľko hodín, ktoré by sa dali využiť napríklad k oslovovaniu nových zákazníkov.

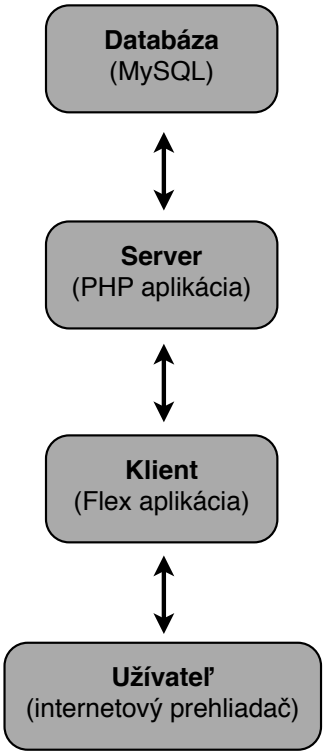
Ukladanie a archivácia. K cenovým ponukám sa bolo treba často vracať. Jediný prvok systematickosti pri ich ukladaní bola zložka s názvom roku, v ktorom boli vytvorené. Pri počte ponúk sto a viac sa ich vyhľadávanie stalo komplikované. Ešte horšia situácia nastala, keď obchodníkov bolo viac a obchodníci aj vedenie spoločnosti potrebovalo mať prehľad o tom, kto akú ponuku dostal.

Ochrana proti zmenám. Dokumenty Microsoft Word a Microsoft Excel nie je žiadny problém upraviť a tým znevýhodniť uchádzača, napríklad vo verejnej súťaži. Samozrejme existujú spôsoby, ako ich ochrániť, ale tie predstavujú komplikácie ako na strane obchodníka, tak na strane zákazníka. Je to nepríjemné, ale stále sa s podobnými praktikami môžme stretnúť.

Stres. Súhrn spomenutých problémov mal aj sekundárne následky v podobe stresu a vypätých situácií. Tie viedli k občasným chybám a zbytočným konfliktom, ktoré nijak neprispievali k pracovnému nasadeniu. Z ľudského hľadiska by som označil práve stres ako hlavný problém, pre ktorý som sa rozhodol vytvoriť túto baklársku prácu.

3 Riešenie

Vytvorenie programu prebiehalo v postupných krokoch. Prvoradé bolo zaznamenať všetky požiadavky a podľa toho navrhnúť databázový model. Aplikácia pozostáva z dvoch častí, server a klient. Server zabezpečuje bránu pre klienta a stará sa prevažne o prácu s databázou. Klient zobrazuje údaje užívateľovi v prijateľnej forme a umožňuje mu prácu s nimi.



Obr. 3.1: Vzťahy medzi jednotlivými časťami aplikácie

Pri vývoji boli použité nasledovné vývojové nástroje:

- **Eclipse Studio**[1] - nástroj pre vývoj PHP aplikácií,
- **Flex Builder 3 Education Edition**[2] - nástroj pre vývoj Flex aplikácií,
- **Charles Web Debugging Proxy**[3] - proxy server pre vývojárov,
- **MySQL Workbench**[4] - nástroj pre správu a návrh MySQL databáz.

[1] `http://eclipse.org/`
[2] `http://yaml.org/`
[3] `http://www.charlesproxy.com/`
[4] `http://wb.mysql.com/`

A zabudnúť nesmieme ani na názov aplikácie. Za ten som vybral latinské slovo **effero**, ktoré v prekladé znamená dvíhať a symbolizuje zameranie spoločnosti.

3.1 Požiadavky riešenia - zadanie

Predtým ako pristúpim k samotnému riešeniu, stručne zhrniem požiadavky, ktoré bolo potrebné pri jej vývoji zohľadniť. Tie reflektujú problémy z predchádzajúcej kapitoly.

Správa zákazníkov. Kompletná správa kontaktov s možnosťou rôznych doplnkových údajov. Možnosť preddefinovania zliav a adries elektronickej pošty.

Katalóg produktov a služieb. : Katalóg produktov a produktových rád. Katalóg by mal pozostávať z kategórií v stromovej štruktúre, kategória obsahuje produktové rady a tie obsahujú produkty. Súčasne je potrebná možnosť katalógu služieb ako školenia, technické prehliadky a iné.

Tvorba cenových ponúk a cenníkov. Každá cenová ponuka môže obsahovať položky z katalógu produktov a služieb s možnosťou vlastných položiek. Každá položka môže mať upravenú zľavu. Cenovú ponuku a cenník bude možné zobraziť, vytlačiť alebo poslať priamo elektronickou poštou (správy musia byť dohľadateľné). Ponuky a cenníky by mali byť filtrovateľné podľa všetkých dôležitých kritérií.

Užívatelia. Sú potrebné dve úrovne užívateľského prístupu. Vedenie spoločnosti má mať prístup ku všetkým záznamom a možnosť pridávať nových užívateľov. Zamestnanci majú prístup len k ich ponukám a cenníkom a možnosť pridávať užívateľov nemajú.

3.2 Databázový model

Správne navrhnutý databázový model predstavuje základný kameň úspechu každej aplikácie. Preto jeho návrh v žiadnom prípade netreba unáhliť. Už od začiatku sa treba zaujímať o tie najmenšie detaily, ktoré z pohľadu užívateľa nemajú veľký význam, ale z pohľadu databáz by mohli predstavovať výrazné odlišnosti. Treba si uvedomiť, že oveľa jednoduchšie je zapracovať zmeny pri návrhu modelu ako v dobe, keď už aplikácia bude pomerne rozsiahla. Okrem správneho zachytenia vzťahov a jednotlivých entít je potrebné dodržať aj formálnu stránku návrhu. Sem patrí pomenovanie

relácií a atribútov, ktoré by si mal vývojár stanoviť na začiatku a hlavne ich dodržiavať. Pri návrhu tohto modelu boli pravidlá nasledovné:

- názvy relácií a atribútov sú v anglickom jazyku a jednotnom čísle (`user`, `product`),
- používanie zápisu camelCase (`productFamily`, `priceList`),
- názvy prepojovacích tabuliek reflektujú vzťah medzi jednotlivými entitami, `product_has_property`, `company_has_meta`),
- cudzie kľúče sa skladajú z názvu entity a skratky Id (`productFamilyId`, `companyId`).

3.2.1 Technická špecifikácia

Pred návrhom databázového modelu je potrebné vhodne zvoliť technické parametre databázy na základe jej charakteru (predpokladaná veľkosť, vyťaženosť - počet dotazov), ktoré časti biznis logiky chceme nechať na RDBMS a ktoré ošetríme v rámci ORM.

Základné parametre, ktoré sa pri MySQL databázach môžu líšiť, sú kódovanie a typ tabuľky (1, str. 175). Kódovanie UTF-8 predstavuje štandard a jeho použitie v rámci všetkých častí aplikácie eliminuje problémy s neštandardnými znakmi. Typ tabuliek InnoDB (1, str. 176), na rozdiel od MyISAM (1, str. 175), umožnuje transakcie a má podporu pre prácu s referenčnou integritou, a preto bol použitý.

3.2.2 Databázový model

Pre lepšiu prehľadnosť rozdelím jednotlivé entity do skupín podľa funkčnosti. Podrobný databázový model je zobrazený v prílohe.

názov entity	**popis**
Užívatelia	
`user`	užívatelia aplikácie
Adresár	
`company`	adresár spoločností
`meta`	číselník možných doplňujúcich údajov o spoločnosti
`company_has_meta`	doplňujúce údaje o spoločnosti
Katalóg produktov a služieb	
`product`	konkrétne produkty
`service`	ponúkané služby
`category`	stromovo usporiadané kategórie produktov
`producer`	číselník dodávateľov
`productFamily`	produktové rady
`property`	číselník vlastností produktov
`measureUnit`	číselník merných jednotiek pre produkty a služby
`productPrice`	ceny produktov
`servicePrice`	ceny služieb
`productFamily_has_property`	konkrétne vlastnosti produktových rád
`product_has_property`	konkrétne vlastnosti produktov
`product_has_service`	služby doporučené pre konkrétne produkty
Cenové ponuky	
`quotation`	cenové ponuky
`quotation_has_product`	produkty na cenovej ponuke
`quotation_has_service`	služby na cenovej ponuke
`customItem`	vlastné položky cenovej ponuky
`quotation_has_image`	obrázky cenovej ponuky
`quotation_has_email`	odoslané správy s cenovou ponukou
Cenníky	
`priceList`	cenníky pre konkrétneho zákazníka
`priceListTemplate`	vzory cenníkov
`priceList_has_email`	odoslané správy s cenníkmi
Média	
`image`	obrázky produktov

Tabuľka 3.1: Popis entít databázového modelu

3.3 Aplikácia - server

Serverová časť aplikácie predstavuje klasickú PHP aplikáciu založenú na Zend Framework. Nedostatok v podobe slabej podpory Modelu bol kompenzovaný integráciou frameworku Doctrine. Zdrojový kód aplikácie sa nachádza na CD nosiči v zložke `Effero/Server`.

3.3.1 Štruktúra

Adresárová štruktúra zodpovedá doporučenej s tým, že bola rozšírená kôli potrebám Doctrine. Zložka s modelmi (`application/models`) okrem tried modelov obsahuje zložky `Base` a `Listener`. V prvej sa nachádzajú základné triedy modelov, ktoré obsahujú potrebné mapovanie na RDBMS a v druhej Listenere k tým modelom, u ktorých boli využité. K ich popisu sa dostaneme neskôr.

3.3.2 Konfigurácia

Konfigurácia serverovej časti bola rozdelená kvôli lepšej údržbe do dvoch súborov.

Prvý z nich obsahuje základné nastavenia aplikácie a prostredia (vývojové, produkčné). Sem patrí napríklad nastavenie zobrazovania chýb alebo spúštacia trieda a nachádza sa v `application/configs/application.ini`

Druhý obsahuje údaje o vlastníkovi aplikácie, prihlasovacie údaje k databáze a využívaným službám (elektronická pošta, LiveDocx). Tieto nastavenia sa nachádzajú v súbore `application/configs/effero.xml`. Takéto oddelenie umožní využiť aplikáciu aj iným spoločnostiam v prípade záujmu.

3.3.3 Práca s modelmi, biznis logika

Modely v sebe ukrývajú celú biznis logiku aplikácie. Jej význam a použitie popíšem na niekoľkých príkladoch. V prvom príklade budeme pracovať s produktom (`Application_Model_Product`). Každý produkt má svoju cenu, ktorá sa ale z času na čas mení, a teda jej platnosť je časovo ohraničená. Pri práci s produktom nás však zaujíma cena aktuálna, a preto je praktické ju niekde uchovávať a aktualizovať. A o to sa stará metóda `updateCurrentPrice()`, ktorá prejde všetky ceny produktu a nastaví aktuálnu.

```
<?php
class Application_Model_Product extends Application_Model_Base_Product
```

```
  public function updateCurrentPrice()
  {
    $currentPrice = null;
    $currentPriceValidFrom = null;

    foreach ($this->ProductPrices as $productPrice)
    {
      if ($currentPrice == null
          && $productPrice->validFrom <= date('Y-m-d'))
      {
        $currentPrice = $productPrice->value;
        $currentPriceValidFrom = $productPrice->validFrom;
      }
      elseif ($currentPriceValidFrom < $productPrice->validFrom
              && $productPrice->validFrom <= date('Y-m-d'))
      {
        $currentPrice = $productPrice->value;
        $currentPriceValidFrom = $productPrice->validFrom;
      }
    }

    $this->currentPrice = $currentPrice;
  }
```

Pri produkte ešte ostaneme. Každý patrí do produktovej rady. Produktová rada je charakterizovaná svojimi vlastnosťami, tzn. že produkty naprieč produktovými radami majú rozličné vlastnosti (rozmer, tvrdosť, dezén a i.). Názov samotného produktu je potom poskladaný na základe týchto vlastností a ich konkrétnych hodnôt pri produktoch. Každá vlastnosť (`Application_Model_Property`) preto obsahuje svoj vzor, akým sa do názvu produktu vloží a prioritu. Spoločne s už konkrétnymi hodnotami (`Application_Model_ProductProperty`) týchto vlastností na produkte je možné poskladať názov. A o to sa takisto stará model.

```
<?php
class Application_Model_Product extends Application_Model_Base_Product
{
public function generateShortName()
{
    if (true === $this->ProductFamily->generateNames)
    {
      $shortNameParts = array();

      if (true === $this->ProductFamily->includeFamilyName)
      {
        $shortNameParts[] = array($this->ProductFamily->name, 0);
```

```
    }

    foreach ($this->ProductProperties as $productProperty)
    {
      $shortNameParts[] =
        $this->_parseProductProperty($productProperty);
    }

    usort($shortNameParts, function($arrayA, $arrayB)
    {
      if ($arrayA[1] == $arrayB[1])
      {
        return 0;
      }

      return ($arrayA[1] < $arrayB[1]) ? -1 : 1;
    });

    $shortNameArray = array();

    foreach ($shortNameParts as $shortNamePart)
    {
      if ($shortNamePart[0] == '')
      {
        continue;
      }

      $shortNameArray[] = $shortNamePart[0];
    }

    $this->shortName = implode(' ', $shortNameArray);
  }
}
```

Medzi ďalšie príklady, kedy sú použité podobné metódy, patria:

- počítanie celkovej ceny cenovej ponuky,
- práca so stromovou štruktúrou v kategóriách,
- aktuálna cena služby,
- generovanie PDF cenových ponúk a cenníkov.

Trochu odlišný prístup je použitý pri Listeneroch. Tieto triedy nahradzujú trigre a zabezpečujú automatické vykonanie akcie pri zmene záznamu (vytvorenie, úprava, zmazanie). Prvý príklad sa týka cenníku. Ten, okrem toho, že sa môže poslať zákazníkovi priamo elektronickou poštou, obsahuje aj odkaz, na ktorom je prístupný

verejne. Tento odkaz treba ochrániť, aby sa k nemu nedalo dostať pomocou jednoduchého identifikátora. Preto má každy cenník (aj cenová ponuka) pristupový kód (`accessCode`), ktorý sa generuje pri vytvorení záznamu.

```
<?php
class Application_Model_Listener_PriceList extends
      Doctrine_Record_Listener
{
  public function preInsert(Doctrine_Event $event)
  {
    $accesCode = md5($event->getInvoker()->companyId . time());
    $event->getInvoker()->accessCode = $accesCode;
  }
}
```

Ďalšie listenery sú využité pri:

- ukladaní cenovej ponuky - prepočítanie celkovej ceny,
- zmazaní obrázku - odstránenie súborov,
- ukladaní produktu - aktualizácia názvu a ceny,
- ukladaní produktovej rady - aktualizovanie vlastností produktov.

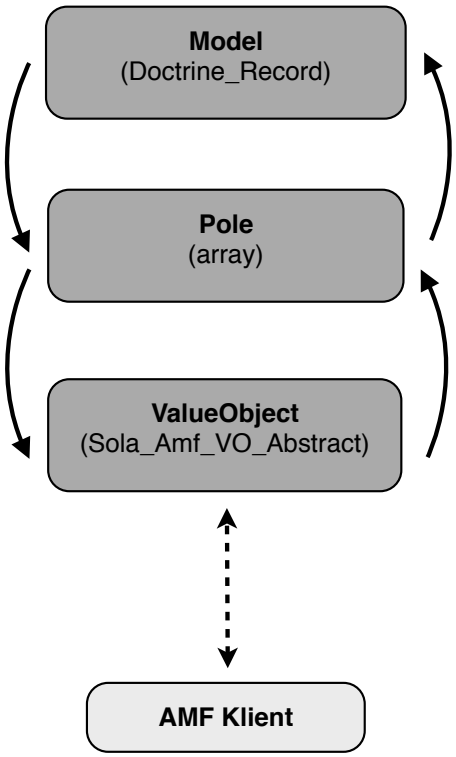
3.3.4 AMF Server

AMF server tvorí kľúčovú časť aplikácie v spolupráci s klientom. Príjma požiadavky, spracúva ich a odpovedá na ne. Rozdelený je do siedmich tried (služieb) a tie nájdeme v zložke `application/services/amf/`. Každá služba pracuje s niekoľkými modelmi, prípadne obsahuje metódy pre správu komunikácie (autentifikácia). Modely sú rozčlenené do služieb podľa tabuľky 3.2.

trieda (služba)	operácie
`Application_Service_Amf_Auth`	
`User`	autentifikácia
`Application_Service_Amf_Catalog`	
`Producer`	CRUD
`Service`	CRUD
`ProductFamily`	CRUD
`Category`	CRUD
`Product`	CRUD
`Application_Service_Amf_Core`	
`User`	CRUD
`Meta`	CRUD
`Property`	CRUD
`MeasureUnit`	CRUD
`Application_Service_Amf_Directory`	
`Company`	CRUD, vyhľadávanie v obchodnom registri
`Meta`	CRUD
`Application_Service_Amf_Media`	
`Image`	CRUD
`Application_Service_Amf_PriceList`	
`PriceList`	CRUD, posielanie elektronickou poštou
`Application_Service_Amf_Quotation`	
`Quotation`	CRUD, posielanie elektronickou poštou

Tabuľka 3.2: Servisné triedy pre AMF

Z pohľadu modelov umožňuje všetky základné operácie (čítanie, ukladanie, mazanie). Medzi AMF serverom a klientom sú objekty serializované do jednoduchých objektov nazvaných ValueObject. Každý model má svoj ValueObject, ktorý rozširuje abstraktnú triedu `Sola_Amf_VO_Abstract`. Tá obsahuje metódy, pomocou ktorých je možné previesť objekt z/do poľa a to synchronizovať s objektom modelu. Tento proces zobrazuje nasledujúci diagram.



Obr. 3.2: Serializácia modelu pre AMF

Služby obsahujú metódy, pomocou ktorých je možné posielať elektronické správy, získavať PDF súbory, náhrávať obrázky a iné.

3.3.5 Autentifikácia a autorizácia

Nakoľko je aplikácia verejne prístupná, je potrebné ju ošetriť proti nežiadúcim prístupom. Autentifikáciu prevádza trieda `Application_Service_Amf_Auth`, ktorá po úspešnom overení vráti identitu užívateľa a tá je odoslaná klientovi v podove ValueObject. Táto identita je po prihlásení dostupná aj v celej serverovej časti.

O autorizácii na jednotlivé činnosti rozhoduje rola užívateľa (`role`). Užívateľ role `admin` môže vykonávať všetky činnosti. Nižšiu úroveň predstavuje užívateľ role `user`, ktorý má prístup len k cenníkom a cenovým ponukám, ktoré sám vytvoril. Taktiež nemá možnosť pridávať a upravovať iných užívateľov.

3.3.6 Spracovanie chýb

Ošetrenie chýb má niekoľko významov. Jednak pomáha už pri vývoji aplikácie, kde vývojar presne vie, čo sa stalo a môže navrhnúť potrebné opatrenia. Ale hlavný vý-

znam má pre koncového užívateľa. Preňho je veľmi dôležité zrozumiteľné vyjadrenie chyby s prípadným návrhom ako pokračovať.

Samotné chybové hlášky sú spracované až v klientskej časti aplikácie. Serverová časť sa snaží presne zistiť, aká chyba nastala a v zrozumiteľnej forme ju podať klientskej časti. Tu má „zrozumiteľné“ mierne odlišný význam. Všetky výnimky (chyby) musia byť transformované do výnimiek pre AMF server. To ošetrujú opäť spomínané servisné triedy. Pri každej činnosti, kde predpokladáme výnimku, ju musíme zachytiť a zmeniť na AMF výnimku. Vhodným príkladom je ukladanie produktu. K najčastejším chybám patria, tzv. validačné (prekročenie max. dĺžky, nezadanie povinných polí) a záznamové (neplatné cudzie kľúče). Ale vylúčiť nemôžme ani nepredpokladateľné chyby. Potom celý proces ukladania aj s popisom vyzerá nasledovne.

```
<?php
class Application_Service_Amf_Catalog
{
  ...
  public function saveProduct(ProductVO $productVO)
  {
    const PRODUCT_VO_FAILED          = 0x0101;
    const PRODUCT_NOTFOUND           = 0x0102;
    const PRODUCT_SAVE_FAILED        = 0x0103;
    const PRODUCT_VALIDATION_FAILED  = 0x0104;
    const PRODUCT_DELETE_FAILED      = 0x0105;
    ...
    try
    {
      // uloženie produktu
      $product->synchronizeWithArray($productVO->toArray(), true);
      $product->save();
    }
    // zachytenie validačných výnimiek
    catch(Doctrine_Validator_Exception $e)
    {
      throw new Zend_Amf_Exception($e->getMessage(),
        self::PRODUCT_VALIDATION_FAILED, $e);
    }
    // zachytenie záznamových výnimiek
    catch(Doctrine_Record_Exception $e)
    {
      throw new Zend_Amf_Exception($e->getMessage(),
        self::PRODUCT_SAVE_FAILED, $e);
    }
    // zachytenie nepredpokladaných výnimiek
    catch(Exception $e)
```

```
    {
      throw new Zend_Amf_Exception($e->getMessage(),
        self::PRODUCT_SAVE_FAILED, $e);
    }

    return (int) $product->id;
  }
```

V prípade chyby tak server odošle naspäť AMF výnimku, u ktorej je významný hlavne kód chyby. Tie sú definované na začiatku servisných tried v podobe konštant (napr.: `self::PRODUCT_SAVE_FAILED`). O osude takejto výnimky na strane klienta si povieme v subkapitole venovanej klientovi.

3.3.7 Využitie LiveDocx

Webová služba LiveDocx[5] je využívaná hlavne na hromadné generovanie šablónovitých dokumentov typu MS Word. Po vytvorení a nahraní šablóny na server sa službe posielajú už len čisté dáta, ktoré vloží do šablóny a vráti hotový dokument. Tento princíp si našiel uplatnenie pri cenníkoch. Postup je nasledovný. Aktuálne ceny nie sú čerpané priamo z cien produktov, ale z CSV súboru. Tieto ceny môžu byt dodatočne upravené (zľava). Na záver sú ceny a ostatné údaje o cenníku poslané službe LiveDocx, ktorá nám vráti hotový dokument vo formáte PDF. S tým môžme ďalej pracovať, napríklad ho rovno odoslať elektronickou poštou. Celý postup sa nachádza v triede `Application_Model_PriceList`.

3.3.8 Využitie LaTeX

Pri tvorení cenových ponúk by využitie LiveDocx prinieslo viac problémov ako úžitku, nakoľko má určité problémy s vizuálnymi prvkami pri konverzii z DOC do PDF. Možno trochu netradičné riešenie, ale z vizuálnej stránky jedno z najlepších, je tvorba cenových ponúk pomocou LaTeXu. Ten nie je podporovaný ani v Zend Frameworku ani v PHP, preto bolo potrebné vytvoriť vlastný kompilátor dokumentov.

Základ tvorí trieda `Sola_Latex_Document`, ktorá zo zdrojového súboru vytvorí dokument. Medzi jeho výhody patrí možnosť podpory viacerých distribúcií LaTeXu a aj možnosť voľby použitého prekladacieho programu (pdflatex, latex, pslatex...). Okrem toho podporuje ukladanie už preložených dokumentov do vyrovnávacej pamäte, aby nemuseli byť vždy pri zobrazení nanovo generované.

[5] `http://www.livedocx.com`

Aby sme vždy nemuseli tvoriť celý dokument manuálne, rozšírením abstraktnej triedy `Sola_Latex_Template_Abstract`, môžme podobne ako pri LiveDocx vytvárať šablóny. Šablónu cenovej ponuky reprezentuje trieda `Application_Latex_Quotation`. Tá vytvorí z cenovej ponuky zdrojový kód, ktorý následne kompilátor preloží do PDF dokumentu.

Takto je celý proces tvorby dokumentov, cenníkov aj cenových ponúk plne automatický. Dokumenty sú formálne jednotné a chránené pred neželanými zásahmi zo strany užívateľa a zákazníka.

3.3.9 Prvotné nahratie dát

Na to, aby sa program mohol začať plne využívať, musí databáza obsahovať potrebné prvotné dáta. Sem patrí hlavne katalóg produktov. Aj keď je možné nahrať produkty manuálne, pri ich počte tisíc a viac nie je tento spôsob príliš efektívny. Doctrine rieši aj tento problém a záznamy je možné nahrávať pomocou súbor YAML, kde sú jednoduchým spôsobom zaznamenané. Takto boli nahraté kategórie a produktové rady. Pre produkty som vytvoril špeciálnu abstraktnú triedu `Application_Doctrine_Fixtures_Trelleborg_Abstract`, ktorá obsahuje metódy špeciálne pre jednotlivé produktové rady. Miesto toho, aby sme zadávali každy produkt zvlášť, stačí zadať všetky možné vlastnosti, spôsob akým ovplyvňujú cenu a základnú cenu a o zvyšok sa postára táto trieda. Všetky prvotné dáta sa nachádzajú v zložke `application/doctrine/fixtures/`.

3.4 Aplikácia - klient

Klient predstavuje tú časť aplikácie, s ktorou prichádza do konktaktu koncový užívateľ. Aj preto sa jej návrh odlišuje od serverovej časti. Hlavný dôraz sa kladie hlavne na užívateľské prostredie. To musí byť jednoduché, prehľadné a hlavne intuitívne.

3.4.1 Štruktúra

Kód aplikácie a jednotlivé triedy sú štrukturované do balíčkov (packages), ktoré zodpovedajú adresárovej štruktúre. Nachádzajú sa v zložke `src` a najdôležitejší baliček je `effero`, ktorého obsah si bližšie popíšeme.

Components obsahuje vlastné vizualné komponenty aplikácie. Tam patria hlavne dialógové okná jednotlivých modelov.

Events obsahuje triedy udalostí. Ich význam je bližšie popísaný v subkapitole Práca so záznamami.

Modules sú jednotlivé subaplikácie.

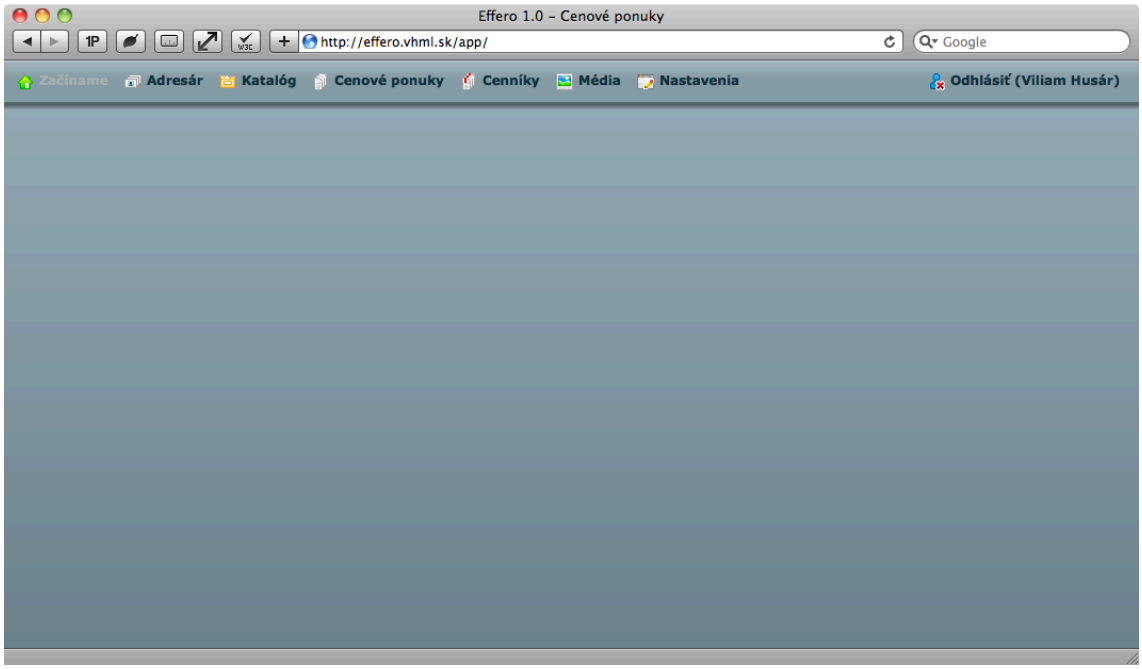
Resources obsahuje triedy pre prácu s chybami a upozorneniami. Tiež sa tu nachádzajú všetky obrázky použité v aplikácii.

Utils obsahuje rôzne pomocné triedy využívané naprieč celou aplikáciou.

VO obsahuje triedy ValueObject pre všetky modely. Práve ich objekty sú posielané vo formáte AMF serverovej časti.

3.4.2 Jadro aplikácie

Aplikácia je modulárna a pozostáva z jadra a niekoľkých modulov. Jadro zabezpečuje komunikáciu medzi klientom a serverom. Definuje objekty triedy `RemoteObject`, ktoré presne mapujú jednotlivé metódy služieb poskytovaných na strane serverovej časti aplikácie. Okrem toho sa stará o autentifikáciu užívateľov a obsahuje navigačnú lištu pre prepínanie medzi jednotlivými modulmi.



Obr. 3.3: Aplikácia po prihlásení

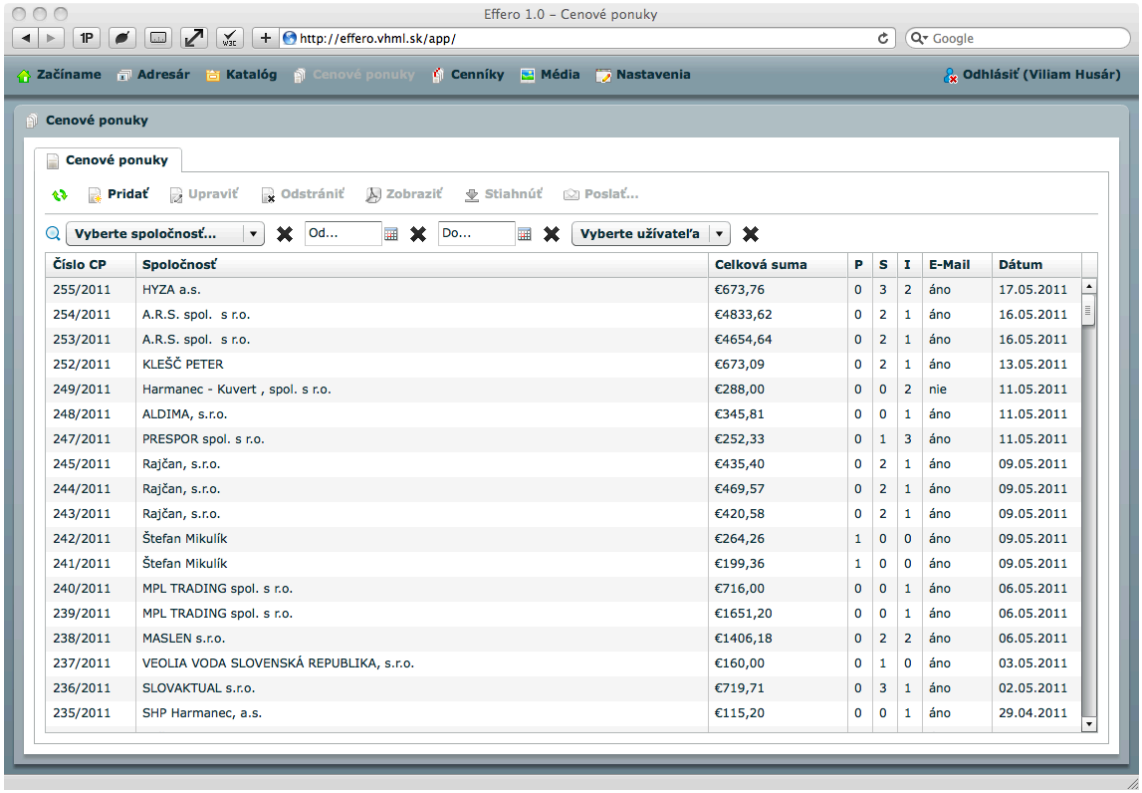
3.4.3 Moduly

Podobne ako serverová časť aj klient je rozdelený do niekoľkých častí. Tie sa nazývajú moduly a každý je malou samostatnou aplikáciou. Pri spustení klienta sa načíta len samotné jadro a až po výbere konkrétneho modulu sa tento aktivuje. Tento prístup šetrí pamäť a aplikácia sa načítava rýchlejšie. V prípade, že užívateľ potrebuje pracovať napríklad len s cenovými ponukami, nemusí čakať na načítanie ostatných modulov.

Moduly medzi sebou priamo nekomunikujú. Ich úplna izolácia by mohla spôsobiť, že ak upravíme spoločnosť v module Adresár, zoznam spoločností v Cenových ponukách by sa o tom vôbec nedozvedel a stále by obsahoval neaktuálne dáta. Tu je využitý návrhový vzor Observer (pozorovateľ). V praxi to znamená, že pri každej zmene záznamu jadro odošle špeciálny objekt typu Event (udalosť), na ktorý jednotlivé moduly podľa potreby reagujú. V kontexte spomenutého príkladu to znamená, že pri zmene záznamu niektorej spoločnosti je o tejto udalosti informovaný vizuálny prvok, ktorý obsahuje zoznam spoločností v module Cenových ponúk a ten si zo servera vyžiada aktuálne dáta.

3.4.4 Užívateľské prostredie a práca so záznamami

Užívateľské prostredie pozostáva z niekoľkých prvkov. Základom každého modulu je okno, v ktorom sú zobrazené relevantné záznamy. Okrem nich obsahuje nástrojovú lištu. Na tej sa nachádajú tlačítka pre prácu so záznamami a vstupy potrebné pre vyhľadávanie alebo filtrovanie záznamov.

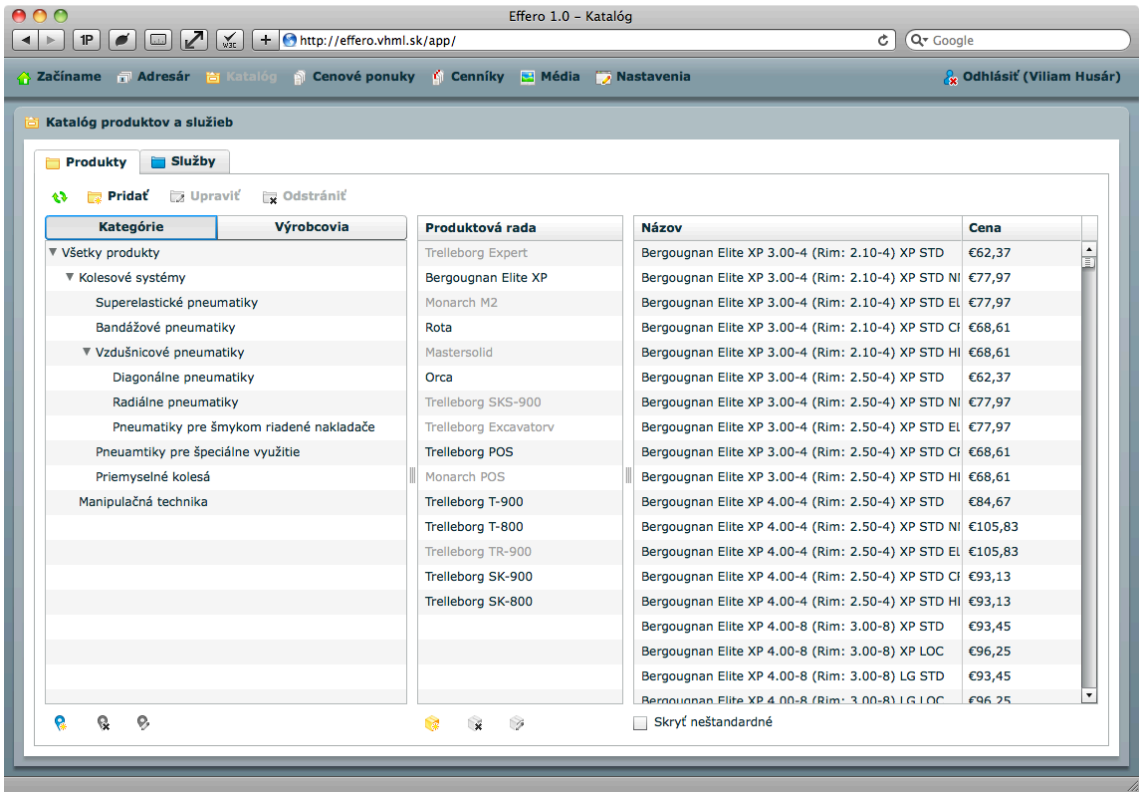


Číslo CP	Spoločnosť	Celková suma	P	S	I	E-Mail	Dátum
255/2011	HYZA a.s.	€673,76	0	3	2	áno	17.05.2011
254/2011	A.R.S. spol. s r.o.	€4833,62	0	2	1	áno	16.05.2011
253/2011	A.R.S. spol. s r.o.	€4654,64	0	2	1	áno	16.05.2011
252/2011	KLEŠČ PETER	€673,09	0	2	1	áno	13.05.2011
249/2011	Harmanec - Kuvert , spol. s r.o.	€288,00	0	0	2	nie	11.05.2011
248/2011	ALDIMA, s.r.o.	€345,81	0	0	1	áno	11.05.2011
247/2011	PRESPOR spol. s r.o.	€252,33	0	1	3	áno	11.05.2011
245/2011	Rajčan, s.r.o.	€435,40	0	2	1	áno	09.05.2011
244/2011	Rajčan, s.r.o.	€469,57	0	2	1	áno	09.05.2011
243/2011	Rajčan, s.r.o.	€420,58	0	2	1	áno	09.05.2011
242/2011	Štefan Mikulík	€264,26	1	0	0	áno	09.05.2011
241/2011	Štefan Mikulík	€199,36	1	0	0	áno	09.05.2011
240/2011	MPL TRADING spol. s r.o.	€716,00	0	0	1	áno	06.05.2011
239/2011	MPL TRADING spol. s r.o.	€1651,20	0	0	1	áno	06.05.2011
238/2011	MASLEN s.r.o.	€1406,18	0	2	2	áno	06.05.2011
237/2011	VEOLIA VODA SLOVENSKÁ REPUBLIKA, s.r.o.	€160,00	0	1	0	áno	03.05.2011
236/2011	SLOVAKTUAL s.r.o.	€719,71	0	3	1	áno	02.05.2011
235/2011	SHP Harmanec, a.s.	€115,20	0	0	1	áno	29.04.2011

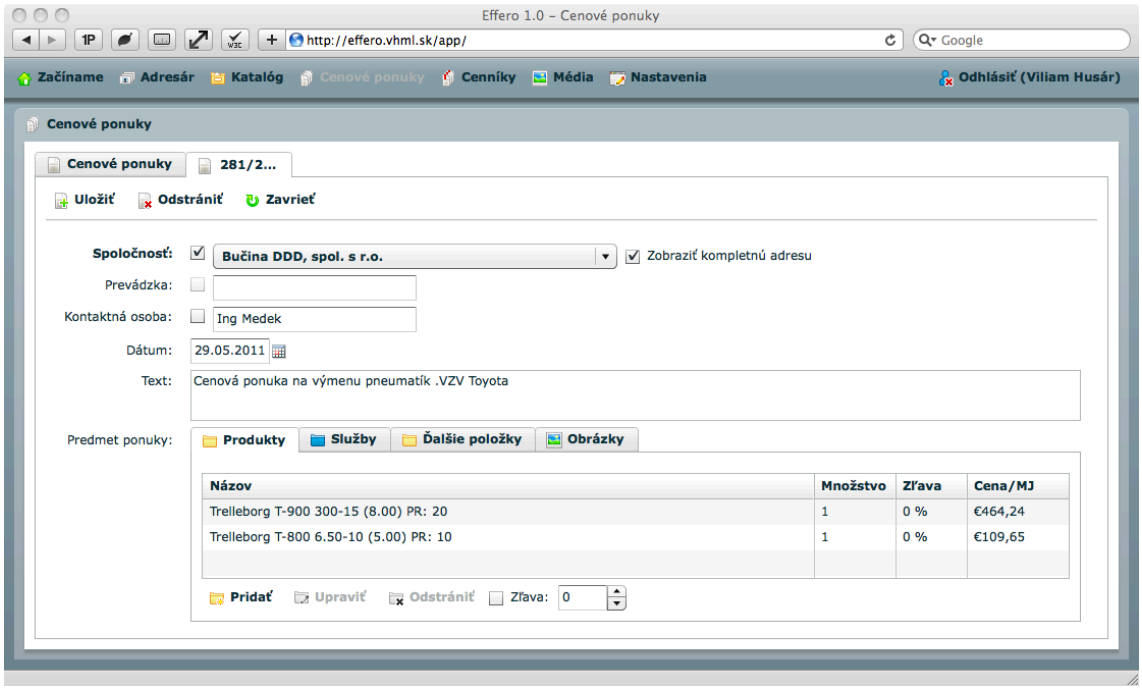
Obr. 3.4: Užívateľské prostredie v module Cenové ponuky

Mierne odlišný koncept využívá modul katalóg (obr. 3.5). V ňom je možná navigácia naprieč kategóriami, výrobcami a produktovými radami. Zoznam produktov sa patrične aktualizuje podľa aktuálneho výberu.

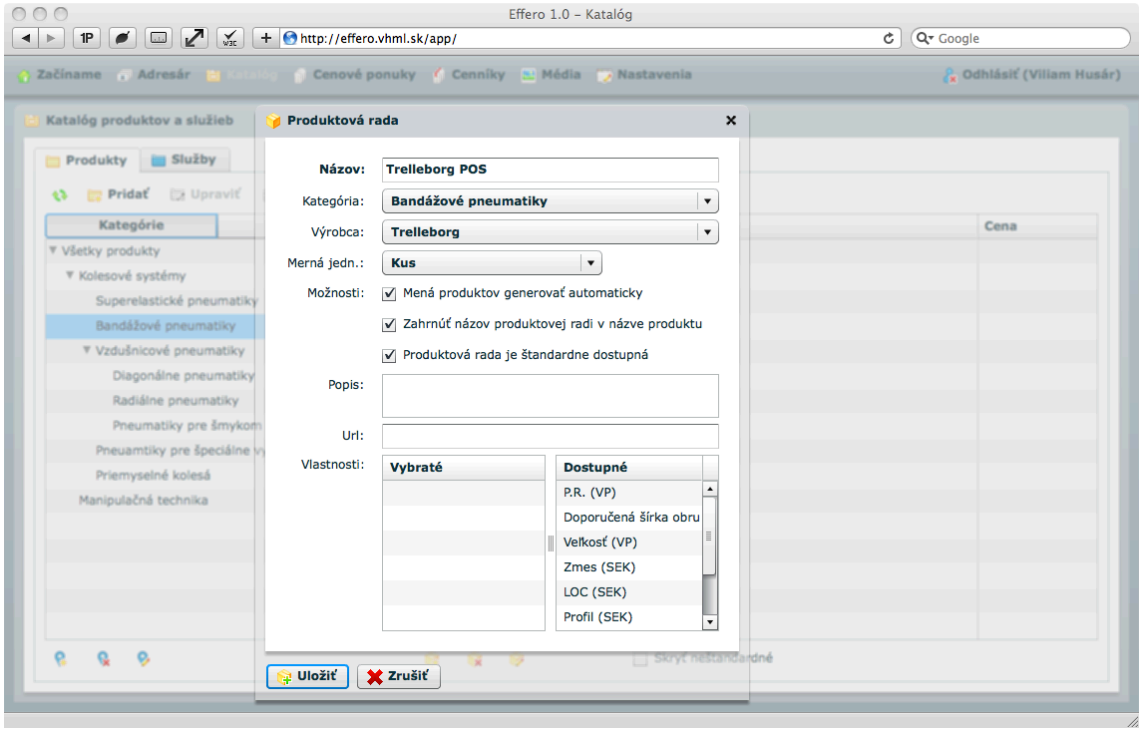
Editácia záznamov využíva dva vizuálne prvky. Jednoduchšie záznamy (kategória produktov, výrobca, produktová rada...) sa editujú v modálnom okne (obr. 3.7). Tie zložitejšie (cenová ponuka, spoločnosť) sa zobrazujú na záložke (obr. 3.6), čo umožnuje súčasnú prácu s viacerými záznamami.



Obr. 3.5: Užívateľské prostredie v module Katalóg



Obr. 3.6: Editácia cenovej ponuky



Obr. 3.7: Editácia produktovej rady

3.4.5 Spracovanie chýb

V podkapitole venovej serverovej časti sme spomenuli, že klient v prípade chyby dostane len kód chyby v podobe hexadecimálneho čísla. O jeho preklad do zrozumiteľnej formy sa stará pomocná trieda `Errors` v balíčku `effero.resources`. Tá obsahuje metódy, pre každý modul jednu, ktoré preložia kód chyby do jeho textovej podoby. Ak sa chyba vyskytne, užívateľ je informovaný formou vyskakovacieho okna.

3.4.6 Interaktívna ukážka aplikácie

Zaznamenať a plne predviesť možnosti aplikácie v podobe textu a obrázkov nie je celkom možné. Preto je súčasťou tejto práce aj niekoľko videonahrávok, ktoré zaznamenávajú najčastejšie činnosti vykonávané užívateľom a dotvárajú tak komplexnejší obraz o aplikácii a jej fungovaní. Nachádzajú sa na kompaktnom disku v adresári `Videoukážky`.

3.5 Nasadenie aplikácie

Po úspešnom dokončení aplikácie prichádza na rad nasadenie do reálnej prevádzky. Možností, kde umiestniť aplikáciu je k dispozíici niekoľko. Najjednouchšie sa javí využite niektorej z dostupných hostingových služieb. Druhá, zložitejšia a nákladnejšia možnosť, je použitie vlastného servera, ktorý by okrem prevádzkovania tejto aplikácie priniesol aj množstvo iných výhod v rámci spoločnosti. Vybraná bola druhá možnosť z nasledujúcich dôvodov:

- väčšina prístupov z lokálnej siete - vyššia rýchlosť,
- možnosť vlastnej konfigurácie serveru, bez nutnosti kontaktovať podporu,
- využitie tlačového serveru,
- uskladnenie dôležitých firemných dát (účtovníctvo, personalistika...) a ich automatické pravidelné zálohovanie na externé médium.

3.5.1 Hardvér

Z hardvérového hľadiska nie je potrebný žiadny ohromujúci výpočtový výkon. Skôr je dôležitá nízka spotreba, malé rozmery, tichý chod (počítač bude umiestnený priamo v kancelárii) a spoľahlivosť. Všetky tieto požiadavky spĺnal počítač Apple Mac mini v nasledovnej konfigurácii:

- rozmery (v - š - d): **5,08 cm - 16,51 cm - 16,51 cm**,
- spotreba (nečinnosť - maximálna): **14 W - 110 W**,
- procesor: **Intel Core2Duo 2,26 GHz**,
- operačná pamäť: **4 GB**,
- pevný disk: **160 GB, 1 TB (externý disk)**.

Obr. 3.8: Počítač Apple Mac mini

3.5.2 Softvér

Počítače Mac využívajú operačný systém Mac OS X Snow Leopard, ktorý je založený na vlastnej implementácii unixového jadra Darwin. To z neho robí ideálnu serverovú platformu. Pre potrebnú funkcionalitu serveru boli skompilované a nainštalované tieto programy:

- webový server: **Apache HTTP Server 2.2.16**[6],
- webový server: **Apache HTTP Server 2.2.16**[7],
- databázový server: **MySQL Community Server 5.1**[8],
- LaTeX distribúcia: **MacTeX-2009 Distribution**[9],

3.6 Ekonomické zhodnotenie a prínosy riešenia

Náklady a prínosy riešenia z ekonomického pohľadu je veľmi zložité presne kvantifikovať. A to hlavne z dôvodu, že významnú časť prínosov nie je možné presne vyčísliť, čo však neplatí o nákladoch. Tento nepomer môže viesť k dojmu, že podobný typ aplikácie je pre spoločnosť nevýhodný. Osobne si však myslím, že opak je pravdou a svoje tvrdenie sa pokúsim zdôvodniť.

Priame náklady zahŕňajú náklady spojené s vývojom aplikácie. Inými slovami cenu, za ktorú by spoločnosť aplikáciu kúpila. Pretože s väčšinou technológii (Zend Framework, Doctrine, Flex, LaTeX) som pracoval prvýkrát, reálne strávený čas je v tomto prípade irelevantný. No po skúsenostiach s ostatným projektami odhadujem jej náročnosť na 200 programovacích hodín, čo pri sadzbe 20,- € na hodinu dáva celkovú sumu **4 000,- €**.

Nepriame náklady predstavujú náklady spojené s prevádzkou aplikácie. Sem patrí cena za server a jeho prevádzku. Nakoľko je využívaný z väčšej časti na iné úlohy v rámci spoločnosti, bolo by nesprávne zaradiť ho medzi priame náklady. Ak určíme podiel, ktorým aplikácia využívá server na 15 %, predstavuje tento náklad **90 €**.

[6] `http://httpd.apache.org/`
[7] `http://httpd.apache.org/`
[8] `http://mysql.com/products/community/`
[9] `http://www.tug.org/mactex/`

Priame prínosy sú tie, ktoré je možné aspoň odhadom vyčísliť. Hlavné kritérium, ktoré som si na ich meranie zvolil, je čas, ktorý aplikácia v priebehu roku používania ušetrila. Ten som zmeral pomocou jednoduchého experimentu, kde som užívateľa požiadal o vytvorenie cenovej ponuky a cenníku pôvodným spôsobom a následne pomocou novej aplikácie. Meraná bola doba od prečítania správy s požiadavkou po doručenie cenovej ponuky a cenníku. Pri cenovej ponuke bol rozdiel časov až 12 minút (pôvodný spôsosb: 16 minút, aplikácia: 4 minúty). Pri cenníkoch bol rozdiel o niečo menší, 5 minút (pôvodný spôsosb: 8 minút, aplikácia: 3 minúty). Pri ročnom objeme 300 cenových ponúk a 60 cenníkov dostávame celkový ušetrený čas **65 hodín**. To pri hodinovej mzde obchodníka 10,- € za hodinu predstavuje **650,- €** ročne a približne šesťročnú návratnosť.

Nepriame prínosy tvoria podľa mňa pre spoločnosť významnejšiu časť prínosov. Považujem za ne:

- možnosť venovať ušetrený čas na oslovenie nových zákazníkov,
- zvýšenie šance na získania objednávok, z dôvodu promptnej reakcie na požiadavky zákazníka,
- zvyšenie kvality medzifiremnej komunikácie,
- eliminovanie zbytočných konfliktov spôsobených nepozornosťou,

Práve súhrn spomenutých prínosov výrazne zvyšuje šance spoločnosti na získavenie nových zákazníkov a následné zvyšovanie zisku.

Záver

Nakoľko samotná aplikácia vznikla s ročným predstihom pred napísaním práce, poskytlo mi to určitý časový nadhľad. Pri vracaní sa k jej kódu som si uvedomil, že niektoré veci by som spravil inak a možno lepšie, ale to patrí k neustálemu vývoju a zlepšovaniu sa.

Verím, že aplikácia splnila svoj cieľ a pomohla spoločnosti zlepšiť niektoré vnútropodnikové procesy. Aplikácia za rok používania vyprodukovala takmer tristo cenových ponúk a šesťdesiat cenníkov. Rýchlo si získala obľubu u používateľov a zaradila sa medzi hlavné nástroje pre komunikáciu so zákazníkmi.

Taktiež dúfam, že čitateľovi táto práca rozšírila znalosti z oblasti internetových aplikácií a možno zmenila pohľad na niektoré technológie, ktoré sa za posledné roky posunuli výrazne dopredu. Ich vzájomným prepojením som chcel poukázať na možnosti, ktoré spoločne poskytujú.

Literatúra

Monografické a seriálové publikácie

[1] KOFLER, M. *Mistrovství v MySQL 5.* Brno: Comupter Press, 2007. 805 s. ISBN 978-80-251-1502-2.

[2] TEOREY, TOBY J. *Database design: know it all / Toby Teorey et al.* Burlington: Elsevier Inc., 2009. 349 s. ISBN 978-0-12-374630-6.

[3] LAVIN, P. *PHP objektově orientované: koncepty, techniky a kód.* Praha: Grada, 2009. 224 s. ISBN 978-80-247-2137-8.

[4] GUTMANS, A., BAKKEN, SS., a RETHANS, D. *Mistrovství v PHP 5.* Brno: Computer Press, 2008. 655 s. ISBN 978-80-251-1519-0.

[5] FOWLER M., RICE D., FOEMMEL M., HIEATT E., MEE R. *Patterns of Enterprise Application Architecture.* Boston: Addison Wesley, 2002. 560 s. ISBN 0-321-12742-0.

[6] ALLEN, R., BROWN S., LO N. *Zend Framework in Action* Greenwick: Manning Publications Co., 2009. 401 s. ISBN 1-933-98832-0.

[7] JACOBS, S - WEGGHELEIRE, K. *Flex 3 Bible.* Berkeley: Apress, 2008. 545 s. ISBN 978-1-4302-0443-5.

[8] GASSNER, D. *Foundation Flex for Developers: Data-Driven Applications with PHP, ASP.NET, ColdFusion, and LCDS.* Indianapolis: Wiley, 2008. 978 s. ISBN 978-0-470-28764-4.

Elektronické publikácie

[9] *Programmer's Reference Guide: Recommended Project Directory Structure* [online]. 2009 [cit. 2011-05-12].
Dostupný z:
<`http://framework.zend.com/manual/en/project-structure.project.html`>.

[10] *Doctrine ORM for PHP: Guide to Doctrine for PHP* [online]. 2009 [cit. 2011-05-12].
Dostupný z: <`http://symfony.sk/manuals/doctrinemanual12.pdf`>.

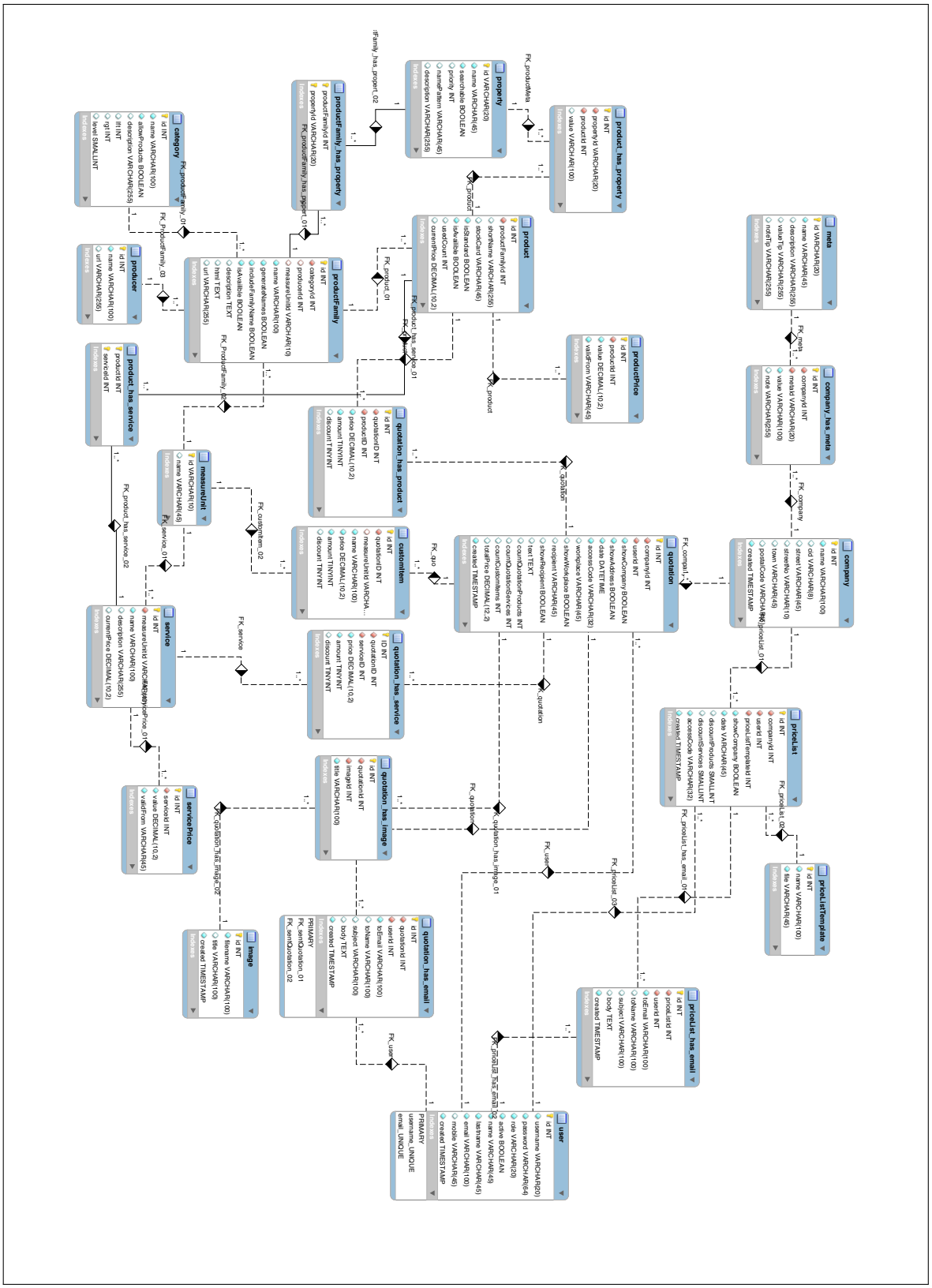
[11] *Adobe Systems Inc.: AMF 3 Specification* [online]. 2008 [cit. 2011-05-12]. Dostupný z: <http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf>.

Zoznam použitých skratiek

RDBMS	Relational database management system
PHP	Code Divisisson Multiple Access
SQL	Structured Query Language
WYSIWYG	What You See Is What You Get
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OOP	Object Oriented Programming
MVC	Model-View-Controller
CRUD	Create Read Update Delete
ORM	Object Relational Mapping
AMF	Action Message Format
PDF	Portable Document Format
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
DBAL	Database Abstraction Layer
DQL	Doctrine Query Language
YAML	Yet Another Markup Language
CSS	Cascading Style Sheet
XHTML	Extensible Hypertext Markup Language
HTML	Hypertext Markup Language
XML	Extensible Markup Language
MXML	Macromedia Flex Markup Language
SWF	Shockwave Flash

CRM	Customer Relationship Management
CMS	Content Management Systems
CSV	Comma Separated Variable
HTTP	Hypertext Transfer Protocol
RIA	Rich Internet Application

A Relačný model databázy effero



Obr. A.1: Relačný model databázy effero

B Obsah CD

zložka	**popis**
`/Effero`	obsahuje samotnú aplikáciu a všetky súčasti MVC
`/Client`	zdrojový kód klientskej časti aplikácie
`/Databázová schéma`	databázová schéma
`/Server`	zfrojový kód serverovej časti aplikácie
`/Videoukážky`	videoukážky práce s aplikáciou
`/Práca - LaTeX`	zdrojový kód práce

Tabuľka B.1: Obsah CD

C Dokumenty

SPOL. S R.O. vhml

Cenová ponuka 280/2010

pre: Uhoľné sklady a.s.
Tepláreńska 4
97101 Prievidza

Dátum	*Vybavuje*	*E-Mail*	*Telefón*
02.09.2010	Viliam Husár	vhusar@vhml.sk	0905 931 169

Produkt / služba	Množstvo	MJ	Zľava (%)	Cena / MJ (€)
Bergougnan Elite XP 3.00-4 (Rim: 2.50-4) XP STD web	2	ks	10	60,07
Trelleborg SK-900 12-16.5 (9.75) PR: 10	3	ks	0	324,12
Demontáž - montáž (rozmery: 10, 12, 13)	2	ks	0	5,30
Školenie obsluhy MV - základné	1	ks	0	99,00
Spolu (bez DPH)				**1 202,09**

Táto cenová ponuka je dostupná na adrese: http://effero.vhml.sk/quotation/?code=6f8fd78d6d195371dfad8121e8f09250

VHML, spol. s r.o
Partizánska cesta 89
974 05 Banská Bystrica

telefón - fax: +421 (48) 416 44 35 - 6
e-mail: vhml@vhml.sk
www: http://vhml.sk

ičo: 36 735 060
ič dph: SK 2022317748
zapísaná v OR OS Banská Bystrica, Sro, 12626/S

Obr. C.1: Vzor cenovej ponuky

SPOL. S R.O. vhml

CENNÍK VZDUŠNICOVÝCH PNEUMATÍK 2010

Uhoľné sklady a.s.

	veľkosť inch	PR	rim	T-800	T-900
8	5.00-8	10	3,00	€ 58.2	€ 62.2
	18x7-8	16	4,33	€ 72.3	€ 86.3
9	6.00-9	12	4,00	€ 74.3	€ 92.3
	21x8-9	14	6,00	€ 108.4	-
10	6.50-10	10	5,00	€ 86.3	-
		12		-	€ 114.4
	23x9-10	20	6,50	€ 168.7	€ 198.8
12	7.00-12	14	5,00	€ 124.5	€ 152.6
		16		-	€ 152.6
	23x10-12	20	8,00	-	€ 210.8
	27x10-12	14	8,00	€ 244.9	-
		16		-	€ 230.9
15	7.00-15	14	5,50	€ 164.9	€ 192.7
	7.50-15	14	6,00	€ 196.7	-
		16		-	€ 242.9
	28x9-15	14	7,00	€ 184.7	€ 208.8
	8.25-15	14	6,50	€ 230.9	€ 287.1
	250-15	20	7,50	€ 277.1	€ 281.1
	300-15	20	8,00	€ 329.3	€ 365.4
16	7.50-16		6,00	-	€ 246.9
20	9.00-20	14	7,00	-	€ 481.9
	10.00-20	16	7,50	-	€ 570.3
	11.00-20	18	8,00	-	€ 648.6
	12.00-20	20	8,50	-	€ 759.0
		24		-	€ 783.0
24	12.00-24	24	8,50	-	€ 979.6
	14.00-24	28	10,00		€ 1294.8

veľkosť inch	PR	rim	SK-800	SK-900
10-16.5	8	8,25	€ 162.7	€ 188.8
10-16.5	10	8,25	€ 176.7	-
12-16.5	10	9,75	€ 234.9	€ 255.1
14-17.5	14	10,50	-	€ 511.8
15-19.5	14	11,75	-	€ 586.3
23X8 1/2-12	6	7,00	€ 68.4	-
27X8 1/2-15	8	8,50	€ 86.5	-

VHML, spol. s r.o.
Partizánska cesta 89
974 05 Banská Bystrica

telefón / fax: + 421 (48) 416 44 35 / 6
e-mail: vhml@vhml.sk

Obr. C.2: Vzor cenníku