SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Ústav informatizácie, automatizácie a matematiky

Tvorba webovej aplikácie s využitím MVP frameworku Nette

DIPLOMOVÁ PRÁCA

Dávid Dubovský

Školiteľ: Ing. Ľuboš Čirka, PhD.

Bratislava 2011

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2010/2011
Evidenčné číslo: FCHPT-5414-40177

S T U
F C H P T

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Mgr. Dávid Dubovský**
ID študenta:	40177
Študijný program:	automatizácia a informatizácia v chémii a potravinárstve
Študijný odbor:	5.2.14 automatizácia
Vedúci práce:	Ing. Ľuboš Čirka, PhD.

Názov práce: **Tvorba webovej aplikácie s využitím MVP frameworku Nette**

Špecifikácia zadania:

Cieľom diplomovej práce je vytvoriť jednoduchý publikačný systém s využitím frameworku Nette, ktorý by mohol nájsť svoje uplatnenie napríklad v procese verejného obstarávania. Nette je framework s otvoreným zdrojovým kódom napísaný v jazyku PHP 5 s plným využitím objektov a uplatňuje MVP architektúru. Diplomant musí pre úspešné zvládnutie úlohy naštudovať problematiku objektovo-orientovaného programovania v jazyku PHP, architektúry MVP a práce s frameworkom Nette. Ďalej musí naprogramovať samotnú aplikáciu a napísať dokumentáciu k aplikácii.

Rozsah práce: 50

Riešenie zadania práce od: 14. 02. 2011
Dátum odovzdania práce: 21. 05. 2011

Slovenská technická univerzita v Bratislave Fakulta chemickej a potravinárskej technológie dekanát - 4 -

Bc. Mgr. Dávid Dubovský
Študent

prof. Ing. Miroslav Fikar, DrSc.
Vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
Garant študijného programu

Čestné prehlásenie

Podpísaný Dávid Dubovský svojím podpisom prehlasujem, že som diplomovú prácu na tému: Tvorba webovej aplikácie s využitím MVP frameworku Nette vypracoval samostatne a s použitím uvedenej literatúry.

Som si vedomý zákonných dôsledkov v prípade, ak hore uvedené údaje nie sú pravdivé.

Na tomto mieste by som súčasne rád poďakoval vedúcemu svojej diplomovej práce Ing. Ľubošovi Čirkovi, PhD. za cenné pripomienky a rady.

V Bratislave 21. 5. 2011 .....

Abstrakt

Diplomová práca sa zaoberá vytvorením internetového modulu, prostredníctvom ktorého je možné vyhlasovať výberové konania na dodanie tovaru a služieb alebo zverejňovať uzavreté zmluvy, zrealizované objednávky a uhradené faktúry.

Aplikácia je vytvorená v prostredí frameworku Nette, ktorý oddeľuje logickú a prezentačnú časť aplikácie a podporuje písanie aplikácie technikou objektovo orientovaného programovania. Aplikácia je postavená na technológiách PHP, JavaScript, jQuery, AJAX, CSS, HTML a MySQL.

Zverejňovanie a správu informácií realizuje autorizovaná osoba. Ku každému zverejnenému záznamu je možné nahrať ľubovoľné množstvo dokumentov. Typ a veľkosť nahrávaných príloh sú aplikáciou limitované. Neaktuálne výberové konania môže autorizovaná osoba presunúť do archívu.

Aplikácia takisto umožňuje správu zoznamu kontaktov a posielanie hromadných správ na tento zoznam. Hromadná korešpondencia je vhodný nástroj na upozorňovanie záujemcov na novo vypísané výberové konania.

Kľúčové slová: framework, internetová aplikácia, Nette, PHP, objektovo–orientované programovanie

Abstract

The thesis deals with the creation of the Internet module. The contracting authority can use this module to tender for the supply of goods and services. Obligee can publish through this module awarded contracts, carried out orders and paid invoices.

The application is created in the framework Nette, which separates the business logic from presentation of the application and supports writing application using object-oriented programming technique. The application is built on technologies PHP, JavaScript, jQuery, AJAX, CSS, HTML and MySQL.

Documents publishing and records management is realised by an authorized person. To each published record can be uploaded any number of documents. Type and size of the uploaded attachments are limited by application. Outdated competitions can be moved to the archive by administrator.

The application also allows administrator to manage contact list and send bulk messages to each item of this list. Mail merge is a useful tool to alert candidates to the newly opened tenders.

Keywords: framework, internet application, Nette, PHP, object-oriented programming

Obsah

Zoznam obrázkov

Zoznam použitých symbolov a skratiek

HTML (HyperText Markup Language) – hypertextový značkovací jazyk

PHP (Hypertext Preprocessor) – hypertextový preprocesor

HTTP (HyperText Transfer Protocol) – hypertextový prenosový protokol

XML (eXtensible Markup Language) – rozšíriteľný značkovací jazyk

AJAX (Asynchronous JavaScript and XML) – asynchrónny JavaScript a XML

URL (Uniform Resource Locator) – globálna adresa dokumentu na Internete

GET / POST– metódy na prenos (získanie) informácií cez HTTP.

PDF (Portable Document Format) – prenosný formát dokumentu

CSS (Cascading Style Sheets) – kaskádové štýly

1. Úvod

Internet je nízko nákladové médium, ktoré umožňuje širokému počtu obyvateľov vyhľadávať informácie rôzneho druhu. Umožňuje takisto jednoduchú publikáciu textov a dokumentov, ktoré môžu užívatelia Internetu prehliadať a sťahovať. Internet sa preto v posledných rokoch stáva živnou pôdou pre verejnú kontrolu hospodárenia štátu, úradov a verejných organizácií daňovými poplatníkmi.

Do popredia záujmu verejnosti sa dostáva takisto problematika verejného obstarávania. Pravidlá zastrešuje zákon č.25/2006 Z. z. o verejnom obstarávaní. Tento zákon podľa §1 ods.1) upravuje zadávanie zákaziek na dodanie tovaru, služieb, uskutočnenia stavebných prác, súťaž návrhov a správu vo verejnom obstarávaní.

Zákon o verejnom obstarávaní §3 ods.1) definuje zákazku ako zmluvu s peňažným plnením uzavretú medzi jedným alebo viacerými verejnými obstarávateľmi alebo obstarávateľmi na jednej strane a jedným alebo viacerými úspešnými uchádzačmi na strane druhej, ktorej predmetom je dodanie tovaru, uskutočnenie stavebných prác alebo poskytnutie služby.

Internet je ideálnym prostredím na to, aby sa výberové konania na dodávku tovaru alebo služieb uskutočnilo transparentne a pred zrakmi čo najväčšieho počtu daňových poplatníkov. Rovnako vypísanie súťaže prostredníctvom Internetu uľahčuje zapojenie viacerých súťažiacich do celého procesu, čo v konečnom dôsledku vedie k zníženiu ceny a zvýšeniu kvality dodaných tovarov a služieb.

Verejným obstarávateľom podľa zákona č.58/2011 Z. z. §6 je obec, vyšší územný celok, právnická osoba z úplnej alebo väčšej časti financovaná verejným obstarávateľom.

Ďalej podľa zákona č.546/2010 Zb., ktorý vstúpil do platnosti prvého januára roku 2011, je povinnosťou zverejňovať zmluvy, ktoré uzavrie povinná osoba a týkajú sa používania verejných prostriedkov.

Cieľom diplomovej práce bolo vytvoriť internetový modul, ktorý je možné vložiť už do existujúcej internetovej prezentácie verejného obstarávateľa alebo povinnej osoby. Prostredníctvom tohto modulu je možné vyhlasovať výberové konania, zverejňovať uzavreté zmluvy, zrealizované objednávky a uhradené faktúry.

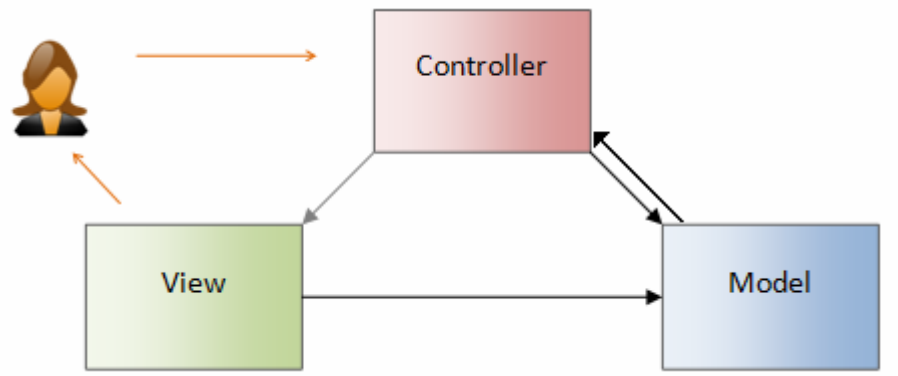
2. Teoretická časť

2.1. Úvod do architektúry MVC a jej prezentačné vzory [1],[2]

MVC (Model-View-Controller) je softvérová architektúra, ktorá umožňuje rozdeliť aplikáciu na tri vrstvy. Tieto vrstvy sú tvorené riadiacou logikou, dátovým modelom a užívateľským prostredím. MVC je dnes rozšírená architektúra tvorby internetových aplikácií.

MVC architektúru prvýkrát popísal Trygve Reenskaug v roku 1979. Táto architektúra zodpovedala možnostiam vtedajších počítačov. View vykresľovalo užívateľské rozhranie a Controller spracovával vstup z klávesnice.

V deväťdesiatych rokoch dvadsiateho storočia prichádza na scénu Internet, kde vstupom je URL a výstupom HTML. Užívateľ vykoná na webovej stránke nejakú akciu. Táto je zachytená Controllerom, ktorý realizuje reakciu na akciu. Zvyčajne zmení Model alebo len priamo ovplyvní View. View tieto zmeny zobrazí užívateľovi späť.



Obr. 1 MVC architektúra

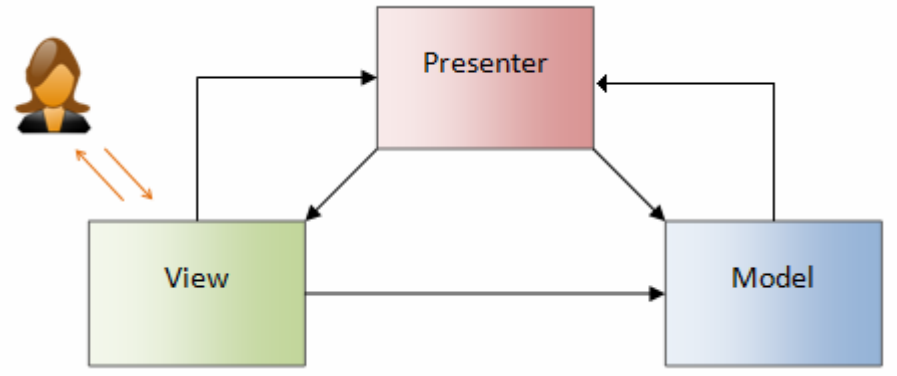
Na obr. 1 vidieť, že aj Controller a View sú navzájom prepojené. Toto prepojenie umožňuje Controleru rozhodnúť o tom, aký View sa má zobraziť.

Pri tvorbe internetovej aplikácie je možnosť sa rozhodnúť na ktorej strane bude rozhodujúca časť prezentačnej logiky. Pri plne AJAXových aplikáciách to môže byť prostredníctvom JavaScriptu na strane klienta. Pri klasických internetových aplikáciách je naopak prezentačná logika generovaná na strane serveru. Internetový prehliadač

predstavuje zobrazovacie zariadenie. Najčastejším prípadom je použitie serverového MVC, ktoré využíva aj prvky AJAXu.

Model obsahuje dáta a aplikačnú logiku. View je serverový kód, ktorý generuje výstup do prehliadača v podobe HTML alebo XML. Controller zachytáva HTTP požiadavky a tie v prípade potreby posiela Modelu, ktorý sa postará o získanie potrebných dát a ich prepojenie s View. Controller spracováva užívateľský vstup.

Menej známa je architektúra MVP (Model-View-Presenter), ktorá však preberá filozofiu MVC (obr. 2). Architektúra sa využíva najmä u widgetových systémov, ale nájde uplatnenie aj v prostredí internetových technológií.



Obr. 2 MVP architektúra

V porovnaním s MVC sa zmenili nasledujúce veci:

- Užívateľský vstup aj výstup je plne kontrolovaný cez View (prostredníctvom ovládacích prvkov užívateľského rozhrania ako tlačidlo, textové pole, internetový odkaz).
- Dôvodom oddelenia View a Presenteru už nie je nutnosť ošetrenia vstupu, ale potreba dodržať architektonický štýl (prezentačná vrstva s MVP sa udržuje lepšie než monolitické View). O ošetrenie vstupov sa stará model.
- View má typicky priamu väzbu na Presenter.
- Presenter v množstve prípadov priamo pracuje s View, takže i táto väzba je silnejšia než v prípade vzoru MVC.
- V prípade MVP architektúry Model obsahuje dáta a aplikačnú logiku. View okrem generovania výstupu aj ošetruje užívateľský vstup tak, že jednotlivým akciám užívateľa priraďuje akcie, ktoré ošetruje Presenter. Samotný Presenter obsahuje aplikačnú a prezentačnú logiku a manipuluje s Modelom.

2.2. MVP Framework Nette [2],[3]

Nette je framework, ktorý podporuje tvorbu aplikácií založených na architektúre MVP. Nette je framework s otvoreným zdrojovým kódom napísaný v jazyku PHP 5 s plným využitím objektov. Nette je vyvíjaný českým programátorom Dávidom Grudlom.

Pri tvorbe aplikácie v prostredí frameworku sa musí programátor prispôsobiť určitým pravidlám, ktoré mu majú uľahčiť tvorbu aplikácie. Frameworky zvyčajne za programátora robia veci, ktoré sa mnohokrát opakujú a stávajú sa nudnou rutinou. Napríklad tvorba formulárov, ošetrenie proti SQL injection a XSS útokom a mnohé iné. Na strane druhej si využitie frameworku vypýta daň v podobe spomínaných pravidiel. Ďalej, framework môže mať určité limity pri vytváraní niektorých komponent aplikácie. Napríklad ak programátor potrebuje vytvoriť netradičný formulár, framework nemusí ponúkať zabudované funkcie na takéto prípady a vtedy je potrebné kombinovať zabudované funkcie s vlastným kódom. Prípadne napísať celé riešenie samostatne.

2.2.1. Životný cyklus aplikácie

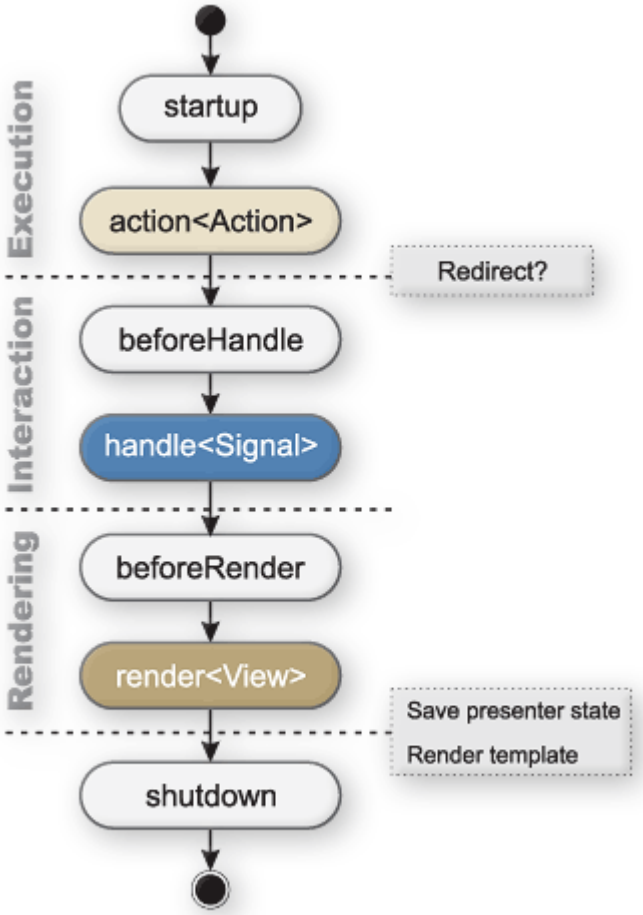
Životný cyklus aplikácie je možné rozdeliť do nasledujúcich bodov:

1. **Router** z URL vytvorí objekt *PresenterRequest* (objekt obsahuje názov presenteru)
2. **PresenterLoader** z názvu presenteru odvodí triedu a prípadne názov súboru
3. Presenter volá **metódy podľa aktuálneho action a view** (prípadne aj subrequestu)
4. Presenter načíta **šablóny**
5. Renderovanie: v tomto a predchádzajúcom bode sa obvykle vytvárajú **odkazy** na iné presentery a ich akcie a pohľady (action a view), do toho sa zapojuje opäť *PresenterLoader* a *Router*.

Takým pomyseľným jadrom aplikácie je presenter. Presenter reaguje na udalosti pochádzajúce od užívateľov a zaisťuje potrebné zmeny v modelu alebo pohľade. Aj presenter sám o sebe má určitý životný cyklus, ktorý sa delí na štyri fázy:

1. výkonná (execution)

2. zmena vnútorných stavov (interaction)
3. vykresľovanie (rendering)
4. ukončenie činnosti (shutdown)



Obr. 3: Životný cyklus presenteru

Na obr. 3 sú bielou farbou označené metódy pre všetky akcie a pohľady. Modrou farbou je označená metóda spracovávajúca konkrétny signál a hnedou metóda pre konkrétny pohľad.

Výkonná fáza obsahuje dve metódy. Metóda *startup* je volaná pri každom začiatku životného cyklu presenteru. Metóda nemusí byť explicitne napísaná, v prípade ak chýba, framework sa o jej načítanie postará. Metóda *action{Action}* by mala obsahovať vykonanie operácií, po ktorých by malo nastať presmerovanie.

Fáza meniaca vnútorné stavy obsahuje metódu *handle{Signal},* ktorá spracováva signály (subrequesty).

Fáza vykresľovania obsahuje metódy *beforeRender* a *render{View}*. Metóda *beforeRender* môže obsahovať nastavenie filtrov pre vykreslenie. Metóda *render{View}*

má na starosti vykreslenie pohľadu a tvorbu odkazov v šablónach a priradenie premenných do šablón.

Potom prichádza fáza, kedy sa uložia všetky vnútorné stavy aplikácie a perzistentné premenné a šablóna sa vykreslí na výstup.

Nasleduje fáza ukončenia činnosti vyvolaná pri ukončení životného cyklu presenteru.

2.2.2. Rútovanie

Pod pojmom rútovanie sa rozumie prepojenie vonkajších požiadaviek a dát. Požiadavkou z vonku pre webovú aplikáciu je URL adresa. Rútovanie umožňuje vytvárať pekné URL adresy (user-friendly URL). Takýto tvar adries je dôležitý z pohľadu optimalizácie aplikácie pre SEO. V Nette framework je táto možnosť voliteľná. Vytvorenie rútovacích pravidiel je možné až po napísaní aplikácie. V prípade ak sa špeciálne nenastaví žiadne rútovacie pravidlo, framework použije vlastné zabudované.

Úlohou rútru je nielen URL adresu spracovať a vytvoriť z nej príkaz pre aplikáciu, ale aj naopak, z interného príkazu vytvoriť URL. Tvorbu user-friendly URL adries má na starosti trieda *Route*.

Prvým parametrom rútru je maska cesty a druhým parametrom je pole začiatočných hodnôt.

Rúter s maskou index.php určí, že pri požiadavke na stránku index.php sa otvorí presenter s názvom *Article* a akcia *show*. Príznak jednosmerky `Route::ONE_WAY` znamená, že rúter môže požiadavky iba prijímať, ale už samotné URL nevytvára.

```
$router[] = new Route('index.php', array('presenter' => 'Article',
                        'action' => 'show',), Route::ONE_WAY);
```

O generovanie URL pre presenter Article a akcie show sa použije vhodný nasledujúci rúter:

```
$router[] = new Route('<presenter>/<action>/<id>',
                        array('presenter' => 'Article',
                              'action' => 'show','id' => NULL,
                             )
                       );
```

a výsledná URL bude mať teda nasledujúci tvar:

```
example.com/article/show/5
```

Rútrovanie umožňuje aj preložiť názvy presenterov, ktoré sú v URL. Je potrebné zadefinovať prekladovú tabuľku.

```
Route::setStyleProperty('presenter', Route::FILTER_TABLE,
                          array('clanok' => 'Article',
                                'kategoria' => 'Category',
                               )
                         );
```

Táto teraz zabezpečí preloženie URL z tvaru:

```
example.com/category/article/
```

na:

```
example.com/kategoria/clanok/
```

Rútrovanie ponúka aj využitie voliteľných sekvencií. Tie sa uzatvárajú do hranatých zátvoriek.

```
$route = new Route('[<lang [a-z]{2}>/]<name>', array());
```

Takýto vzor akceptuje URL cesty v tvare:

```
example.com/sk/download  => lang=sk, name=download
example.com/download     => lang=NULL, name=download
```

Zátvorky je možné aj ľubovolne vnoriť.

```
$route = new Route('[<lang [a-z]{2}>]/]<name>[/page-<page>]',
                     array('page' => 0,)
                   );
```

Takýto vzor akceptuje URL cesty v tvare:

```
example.com/clanok/
example.com/clanok/strana-2
example.com/sk/clanok/strana-2
```

Rútrovanie umožňuje aj mnoho iných vecí.

2.2.3. Generovanie odkazov

Každú HTTP požiadavku v Nette\Aplication je možné opísať trojicou *modul:presenter:action* doplnenú o prípadné pole argumentov. V Nette frameworku sa odkazy na jednotlivé presentery a akcie generujú prostredníctvom metódy *link()*.

```
$this->link(destination [,arg [,arg ...]]);
```

kde destination je:

- 'anotherAction' (odkaz na aktuálny presenter a anotherAction)

- 'AnotherPresenter:anotherAction' (odkaz na AnotherPresenter a anotherAction)
- 'AnotherPresenter:' (odkaz na AnotherPresenter a východziu actionDefault)
- 'AnotherModule:Presenter:action' (odkaz do submodulu)
- ':TotalyAnotherModule:Presenter:action' (odkaz do iného modulu)
- '//AnotherPresenter:anotherAction' (absolútny odkaz s http://domena.tld/... na začiatku na AnotherPresenter a anotherAction)
- 'this' (odkaz na aktuálny presenter a aktuálnu action)

Na iné presentery, akcie alebo stavy sa odkazuje v šablónach. Napríklad v šablóne sa zmena stavu zapíše prostredníctvom signálu nasledujúco:

```
$presenter->link(‚insert!‘, 2)
```

V presenteri je tomuto zápisu zodpovedná akcia handleInsert():

```
public function handleInsert($coin)
{
   $this->money += max(0, (int) $coin);
   $this->redirect('this');
}
```

ktorá zmení stav aplikácie a v URL sa objaví stav aplikácie:

```
index.php?money=2
```

ktorý zodpovedá akcii:

```
index.php?money=2&do=insert
```

O presmerovanie na stavové URL sa stará programátor v danom presenteri:

```
$this->redirect('this');
```

Inak by sa mohlo stať, že ak je URL príkazové, po opätovnom načítaní stránky je daná akcia vykonaná opäť. Napríklad viacnásobné odoslanie formulára.

Signál teda predstavuje požiadavku užívateľa pod prahom pohľadu. To znamená, že ide o akciu, po ktorej nedôjde ku zmene pohľadu. Zápis:

```
$component->link()
```

alebo ak sa pridá vykričník:

```
$presenter->link()!
```

Pohľad môže meniť iba presenter. Akcia, ktorá zmení pohľad sa zapíše:

```
$presenter->link()
```

alebo ak je potrebné zadefinovať priamo názov akcie v presenteru, ktorá zmení pohľad, je možné písať:

```
$this->presenter->link('inyPohlad')
```

2.2.4. Adresárová štruktúra

Aplikácia dodržujúca MVP architektúru by mala mať určitú adresárovú štruktúru. Hlavnou časťou adresárovej štruktúry sú tri priečinky.

```
app/
   +
   +
libs/
   +
...+
www/
   +
...+
```

V priečinku *app* sa nachádza aplikačná logika aplikácie. Každá časť aplikačnej logiky má vytvorený svoj vlastný priečinok a týmto sa dodržiava oddelenie aplikácie na tri vrstvy: Model, View a Presenter.

```
app/
  +presenters/
  +models/
  +templates/
  +temp/
  +bootstrap.php
  +config.ini
  +...
```

V priečinku *presenters* sa nachádzajú presentery aplikácie. V priečinku *models* sa nachádzajú modely aplikácie. V priečinku *templates* budú šablóny aplikácie. Tieto sú tvorené skriptovacími jazykmi HTML a CSS. Týmto sa oddeľuje logická a prezentačná vrstva aplikácie. Do priečinku *temp* si aplikácia prostredníctvom frameworku ukladá dočasné dáta. Tento priečinok musí mať nastavené práva na zápis. Súbor *config.ini* obsahuje konfiguráciu aplikácie. V tomto súbore sa nachádza napríklad nastavenie pripojenia na databázu.

```
[development < common]
; database options in development mode (localhost prevadzka)
database.driver = mysql
database.database = nette
database.charset = utf8
database.lazy = TRUE
database.host = localhost
```

```
database.profiler = TRUE
database.username = username
database.password = password
```

Tento súbor musí byť chránený proti otvoreniu z vonku z prostredia Internetu. V opačnom prípade by si mohol ktokoľvek prečítať konfiguračné údaje, akými sú prístupové heslá do databázy a iné. Súbor je možné ochrániť vytvorením súboru *.htaccess*, ktorý sa nachádza vtom istom priečinku ako konfiguračný súbor. Do súboru sa zapíše:

```
Order Allow,Deny
Deny from all
```

Súbor *.htaccess* umožňuje upraviť a nastaviť niektoré direktívy serveru a funguje iba na Apache serveroch.

Súbor *bootstrap.php* štartovací súbor celej aplikácie. Načíta všetky knižnice frameworku Nette, nastavuje prostredie a v tomto súbore sa definujú aj rútovacie pravidlá.

V priečinku *www* sa nachádzajú súbory s javascriptovými funkciami, skriptovacie CSS súbory a úvodný súbor aplikácie, najčastejšie nazývaný ako *index.php*.

```
www/
  +css/
  +js/
  +index.php
  +...
```

Priečinok *www* musí byť verejne dostupný z Internetu. O toto nastavenie sa postará najčastejšie administrátor serveru. Z Internetu je teda možné mať prístup iba k jednému súboru aplikácie, ktorým je súbor *index.php.* Všetky ostatné dôležité súbory aplikácie sú v priečinku *app* a sú z Internetu nedostupné. Týmto sa výrazne zvyšuje bezpečnosť celej aplikácie.

Samotný súbor *index.php* obsahuje iba definíciu konštánt, ktoré aplikácia potrebuje k správnej funkčnosti. Ide o zadefinovanie absolútnych ciest k jednotlivým priečinkom aplikácie.

```
// absolute filesystem path to the web root
define('WWW_DIR', __DIR__);

// absolute filesystem path to the application root
define('APP_DIR', WWW_DIR . '/../app');
```

```
// absolute filesystem path to the libraries
define('LIBS_DIR', WWW_DIR . '/../libs');
// absolute filesystem path to the temporary files
define('TEMP_DIR', WWW_DIR . '/../temp');
// absolute URL path to the web root
define('WWW_ROOT', 'http://localhost/nette/www/');
// load bootstrap file
require APP_DIR . '/bootstrap.php';
```

V priečinku *libs* sa nachádzajú súbory samotného frameworku Nette, prípadne sa sem dopĺňajú rôzne iné knižnice, ktoré aplikácia využíva.

Štruktúra samotného priečinku *app* sa odvíja od toho, či na aplikácii pracuje sólista alebo tím programátorov. V prípade, ak na aplikácii pracuje viacero programátorov súčasne, používa sa ukladanie oddeleným spôsobom.

```
presenters/
    FrontModule/
        HomepagePresenter.php
    AdminModule/
        AuthPresenter.php

templates/
    FrontModule/
        @layout.phtml
        Homepage.default.phtml
    AdminModule/
        @layout.phtml
```

V prípade, ak na aplikácii pracuje jeden programátor, používa sa ukladanie na jedno miesto.

```
presenters/
    FrontModule/
        HomepagePresenter.php
        templates/
            @layout.phtml
            Homepage.default.phtml
    AdminModule/
        AuthPresenter.php
        templates/
            ...
```

2.2.5. Formuláre

Tvorba formulárov je jedným z dôvodov, prečo používať framework. Vytvorenie jedného formulára si vyžaduje viacero krokov. Medzi ne patrí:

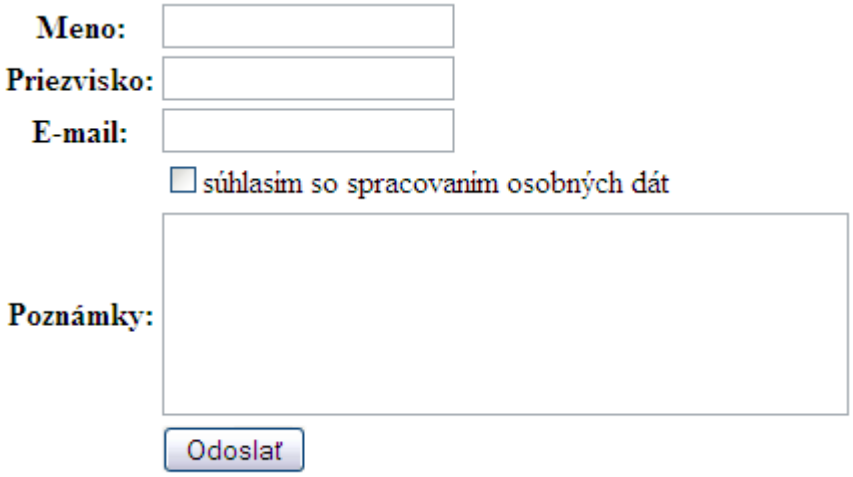
- napísať HTML kód
- napísať validáciu na strane klienta (JavaScript)
- napísať validáciu na strane serveru (PHP)
- otestovať vo viacerých prehliadačoch
- ošetriť magic quotes

Nette framework ponúka vstavané metódy, ktoré vygenerujú formulár aj so všetkými potrebnými bezpečnostnými prvkami. Nette obsahuje dva spôsoby vytvárajúce formuláre.

Prvý spôsob sú triedy *Nette\Forms*, ktoré umožňujú vytvoriť formulár aj mimo Nette aplikácie. Umožňuje tak ľahkú implementáciu do aplikácie, ktorá nepodporuje MVP architektúru, alebo nevyužíva framework. Nasledujúci kód vytvorí formulár:

```
//vlozi minimalizovanu verziu frameworku
require './Nette/nette.min.php';
//inicializuje objekt triedy
$form = new Form;
// name je názov prvku, Meno je popis
$form->addText('name', 'Meno:');
$form->addText('surname', 'Priezvisko:');
$form->addText('email', 'E-mail:');
$form->addCheckbox('suhlas',  'súhlasím  so  spracovaním  osobných
                    dát');
// 40 – stlpcov, 5 - riadkov
$form->addTextArea('text', 'Poznámky:',40,5);
$form->addSubmit('send', 'Odoslať');
echo $form;
```

ktorý po otvorení v internetovom prehliadači vyzerá nasledujúco (obr. 4):

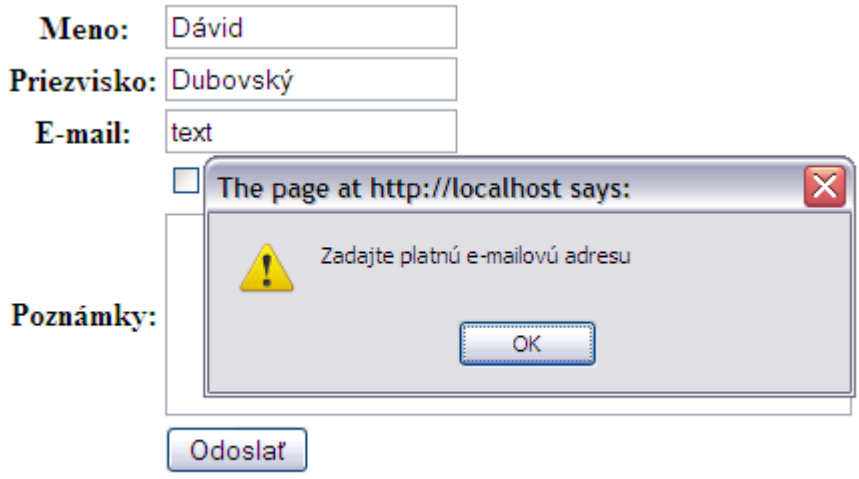


Obr. 4 Formulár triedy Nette/Forms

Takýto formulár však neoveruje správnosť dát. Trieda *Nette\Forms* má zabudované metódy na overenie na strane klienta, ako aj na strane serveru.

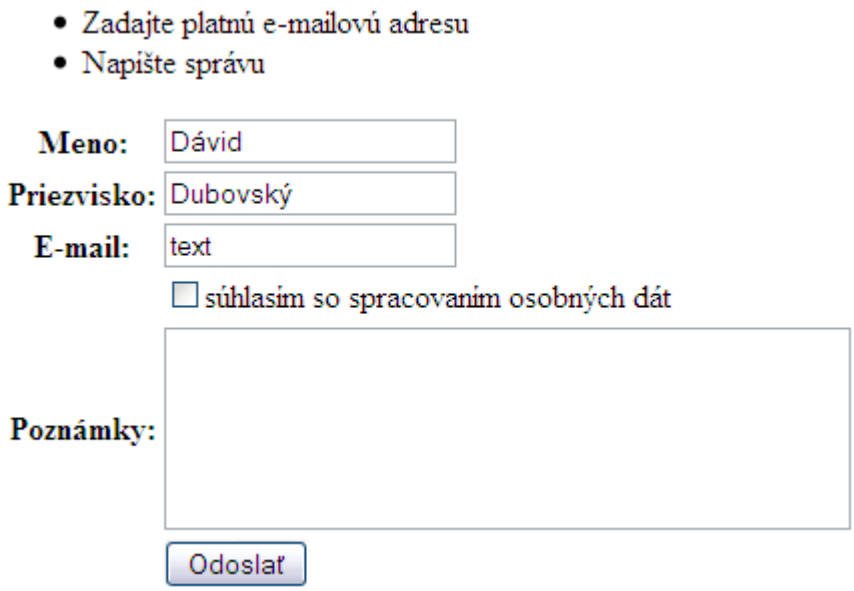
```
$form->addText('name', 'Meno:')
     ->addRule(Form::FILLED, 'Zadajte meno');
$form->addText('surname', 'Priezvisko:')
     ->addRule(Form::FILLED, 'Zadajte priezvisko');
$form->addText('email', 'E-mail:')
     ->addRule(Form::EMAIL, 'Zadajte platnú e-mailovú adresu');
$form->addTextArea('text', 'Poznámky:',40,5)
     ->addRule(Form::FILLED, 'Napíšte správu');
```

Nasledujúci obr. 5 zobrazuje upozornenie užívateľa na to, že sa pokúsil odoslať nesprávne údaje.



Obr. 5 Formulár s overovaním dát na strane klienta

V prípade ak užívateľ má vypnutý JavaScript, prebehne validácia na strane servera a zobrazeniu informácie o nesprávne vyplnených položkách (obr. 6).



Obr. 6 Formulár s overovaním dát na strane servera

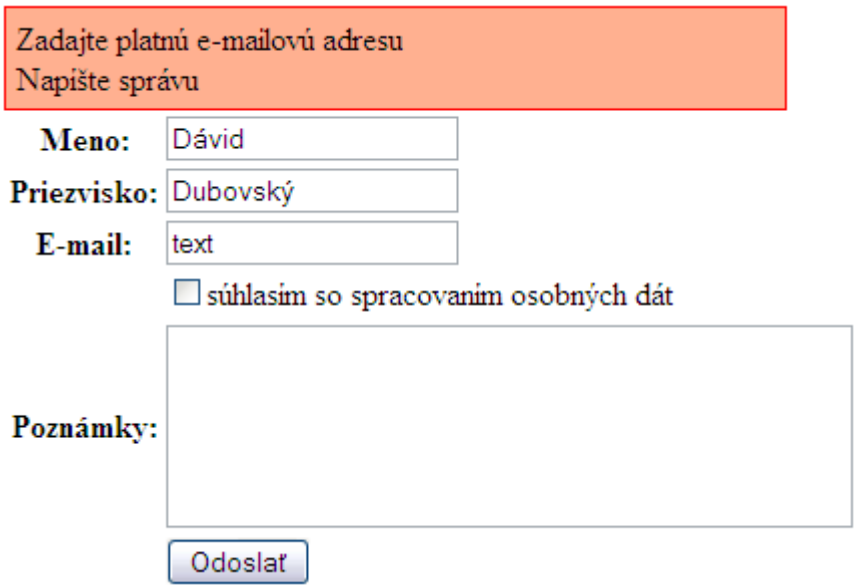
Nette vygeneruje nečíslovaný zoznam chýb, ktorý má vlastnú CSS triedu.

```
<ul class="error">
   <li>Zadajte platnú e-mailovú adresu</li>
   <li>Napište nám prosím vzkaz</li>
</ul>
```

Tento zoznam je však nevýrazný. Pomocou CSS triedy je možné chyby zvýrazniť.

```
ul.error {
   border: 1px solid red;
   width:30%;
   list-style-type: none;
   padding: 5px; margin: 0px;
   background-color:#FFB090;
}
```

Chybová správa dostáva pekný tvar (obr. 7).



Obr. 7 Formulár s upravenou chybovou správou

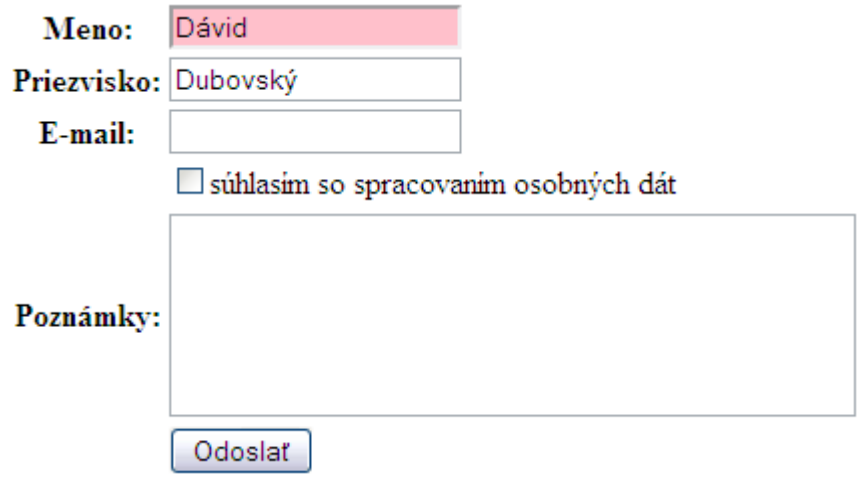
Takisto je možné nastaviť metódu HTTP na GET a zmeniť cieľ, kam sa má formulár odoslať. Metóda HTTP POST a spracovanie na tej istej stránke sú prednastavené hodnoty.

```
$form->setAction('./spracuj.php');
$form->setMethod('get');
```

Pomocou metódy *getControlPrototype()* je možné danému formulárovému prvku nastaviť rôzne CSS atribúty:

```
->getControlPrototype()->style = "background: pink";
```

a tým upravovať prvky formulára (obr. 8):



Obr. 8 Úprava prvkov formulára

Rovnakou metódou je tiež možné nastaviť formulárovému prvku CSS triedu, alebo naň zavesiť JavaScriptovú udalosť:

```
->getControlPrototype()
//zavesi na prvok vlastne JS funkcie
->onfocus('onFocusForm(this)')
->onblur('onBlurForm(this)')
//priradi CSS triedu
->class('input');
```

Na kontrolu odoslania formulára sa používa metóda *isSubmitted()* a na kontrolu správnosti vyplnenia metóda *isValid()*.

```
// ak bol formular odoslany
if ($form->isSubmitted()) {
        // ak su vsetky polozky vyplnene spravne
        if ($form->isValid()) {
            echo '<h2>Formulár bol odoslaný</h2>';
         // nacita zadane udaje do premennej
            $values = $form->getValues();
            // vypise zadane udaje do prehliadaca
          Debug::dump($values);
        }
}
```

Druhý spôsob sú triedy *Nette\Application\AppForm*, ktoré sa používajú pri vytváraní formulárov v presenteroch. Na vytváranie sa používajú špeciálne komponenty, ktoré sú nazývané továrničky. Najprv je potrebné zo šablóny prostredníctvom filtru poslať informáciu do presenteru, aby sa daný komponent vykreslil.

```
{control createContactForm}
```

Ku značke *control* existuje aj alias *widget*. Značka *control* sa do PHP preloží ako:

```
$control->getWidget("createContactForm ")->render();
```

V presenteri sa potom komponent vytvorí pomocou verejnej metódy. Nasledujúca metóda vytvorí formulár s využitím metód triedy *Nette\Application\AppForm*:

```
public function createComponentCreateContactForm($name)
{
   $contactForm = new AppForm($this, $name);
   $contactForm->addText('name', 'Meno:')
               ->addRule(Form::FILLED, 'Zadajte meno');
   $contactForm->addText('surname', 'Priezvisko:')
               ->addRule(Form::FILLED, 'Zadajte priezvisko');
```

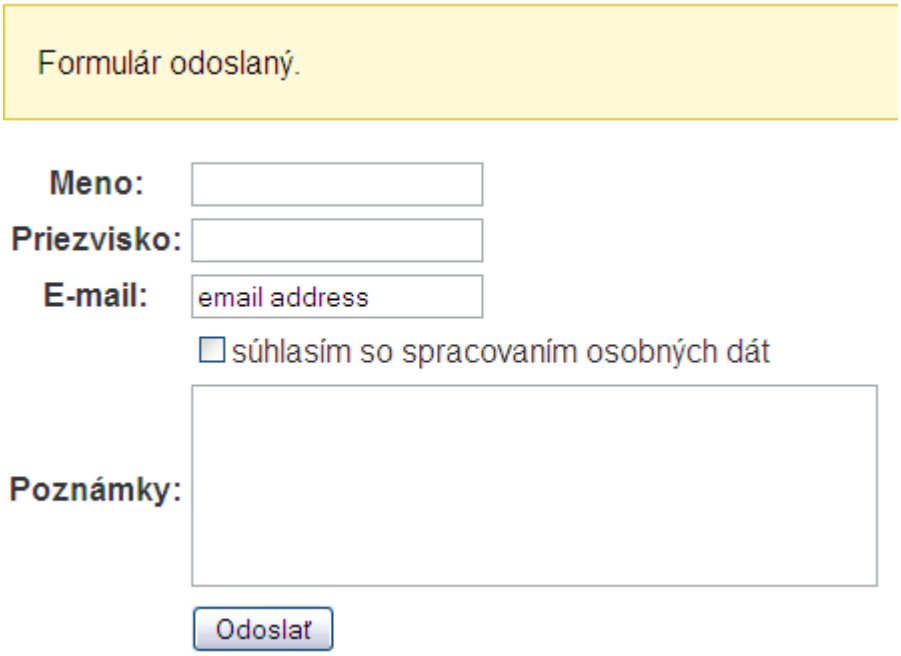
```
        $contactForm->addText('email', 'E-mail:')
                    //musi byt vyplnene
                    ->addRule($contactForm::FILLED, 'Zadajte e-mail')
                    //musi mat tvar email adresy
                    ->addRule(AppForm::EMAIL, 'Platný e-mail.')
                    ->getControlPrototype()
                    //zavesi na prvok vlastne JS funkcie
                    ->onfocus('onFocusForm(this)')
                    ->onblur('onBlurForm(this)')
                    //priradi CSS triedu
                    ->class('input');
        $contactForm['email']->setDefaultValue('email address');
        $contactForm->addCheckbox('suhlas', 'súhlasím so spracovaním
                                            osobných dát');
        $contactForm->addTextArea('text', 'Poznámky:',40,5)
                    ->addRule(Form::FILLED, 'Napíšte nám prosím odkaz');
        $contactForm->addSubmit('send', 'Odoslať');
        $contactForm->onSubmit[] = array($this,
                                         'formContactFormSubmitted'
                                         );
    }
```

Na rozdiel od triedy *Nette\Forms*, metóda *onSubmit* sa zavolá iba v prípade, ak je formulár skutočne validny a skutočne odoslaný a nie je potrebné realizovať kontrolu prostredníctvom metódy *isSubmitted*. K vytvorenému formuláru sa vytvárajú obslužné handlery, ktoré spracovávajú formulár. Napr. obsluha po odoslaní, ktorá môže uložiť odoslané dáta do databázy a vypíše oznam o úspešnom odoslaní.

```
    public function formContactFormSubmitted(AppForm $form)
    {
        // spracovanie formulara ( napr. ulozenie do DB cez model)
        ....
        // oznamenie o uspesnom vlozeni
        $this->flashMessage('Formulár odoslaný.');
        // po príkazu musí nasledovat presmerovanie
        $this->redirect('this');
    }
```

Po úspešnom odoslaní je užívateľ vhodne informovaný (obr. 9). Dôležitý prvok je presmerovanie, ktoré zabezpečí, aby nebol formulár zasa odoslaný po opätovnom načítaní stránky. Na zobrazenie informácie užívateľovi na stránke sa používa metóda *flashMessage* triedy *Control*. Je to miesto, kde sa ukladajú rýchle správy, ktoré je

nezávislé na presenteri a informujú o výsledkoch operácií. Tieto správy sú zobrazované až po presmerovaní stránky, aby nedošlo k chybe pri presmerovaní, keďže http hlavičky už boli odoslané pri zobrazení informácie.



Obr. č.9 Formulár triedy Nette/Application/AppForm

2.2.6. Šablóny

Šablónovací systém frameworku oddeľuje logickú a prezentačnú časť aplikácie. Šablónovací systém systému Nette je možné použiť aj v aplikácii, ktorá nevyužíva návrh MVP. Vytvára tým alternatívu k iných šablónovacím systémom akými sú napríklad Smarty.

V prípade využitia MVP návrhového vzoru a samotného frameworku Nette sa o načítanie šablónovacieho systému postará štartovací súbor aplikácie. Šablónovací systém má na starosti trieda *Nette\Templates\Template*.

Nová šablóna a nastavenie cesty k nej sa vytvorí nasledujúco:

```
$template = new Template;
// nastavi cestu k suboru šablony
$template->setFile('./sablona.phtml');
```

Pri vývoji vo frameworku Nette však nie je potrebné šablóny vytvárať z prostredia kódu, ale stačí ich vytvoriť v textovom editore a uložiť v potrebnom

formáte do adresárovej zložky *templates*. V presenteroch sa potom do prislúchajúcej šablóny nastavujú hodnoty premenných.

Priamo v distribúcii frameworku je zabudovaných niekoľko filtrov, spomedzi ktorých je najvýznamnejší *Latte*. V prípade potreby je možné zaregistrovať aj vlastný filter. Filtre umožňujú pracovať s blokmi šablón alebo s ľubovoľným obsahom šablóny. *Latte* filter obsahuje základné makrá, ktoré predstavujú ekvivalent k nejakej akcii. Napríklad formulár sa prostredníctvom makra *control* v šablóne vykreslí nasledujúco:

```
{control createContactForm}
```

Filter je v šablóne ohraničený prostredníctvom zložených zátvoriek *{ }*. Napríklad vypísanie hodnoty premennej sa vykoná prostredníctvom makra:

```
{$variable}
```

Takisto je možné vykresľovať časti šablón prostredníctvom podmienených výrazov v tvare:

```
{if ?} ... {elseif ?} ... {else} ... {/if}
```

nasledujúco:

```
{if $formSent }
   <h1>Formulár bol odoslaný</h1>
{/if}
```

Nad časťou šablóny je takisto možné vykonávať iterácie prostredníctvom cyklu. Napríklad cyklom foreach je automaticky inicializovaná premenná *$iterator*, ktorá významnou mierou rozširuje informácie o prebiehajúcom cykle.

```
{foreach $rows as $row}
   {if $iterator->isFirst()}
      <table>
            {/if}
            <tr id="row-{$iterator->getCounter()}">
                  <td>{$row->name}</td>
                  <td>{$row->email}</td>
            </tr>
             {if $iterator->isLast()}
      </table>
   {/if}
{/foreach}
```

Metóda premennej *$iterator isFirst()* zistí, či ide o prvý priechod cyklom. Metóda *isLast()* zistí, či ide o posledný priechod cyklom. Metóda *getCounter()* zistí aktuálny priechod cyklom počítaný od jednotky. Metóda *isOdd()* zistí, či ide o nepárny priechod cyklom a metóda *isEven()* zistí, či ide o párny priechod cyklom.

Priamo v distribúcii frameworku sú k dispozícii aj pomocné funkcie, tzv. „helpery“. Do šablóny je možné v prípade potreby zaregistrovať aj vlastné pomocné funkcie. Pomocné funkcie umožňujú napríklad rýchly prevod z malých písmen do veľkých, prevod medzi formátmi dátumu, odstraňovanie bielych znakov a iné. Napríklad priradením textového reťazca do premennej v presenteri:

```
private $vypis = null;
public function renderDefault()
{
   $this->template->vypis = 'premenná do šablóny';
}
```

je možné v šablóne použiť pomocnú funkciu na jej úpravu:

```
{$vypis|capitalize}
```

Textový reťazec je prevedený na kapitálky.

2.2.7. Prihlasovanie užívateľov

Prihlasovanie slúži k overeniu identity užívateľa. V mnohých prípadoch je totiž potrebné obmedziť prístup do určitých častí internetovej aplikácie iba na oprávnených užívateľov. Spôsob autentizácie užívateľov a ošetrenie tohto procesu s ohľadom na bezpečnosť je v kompetencii programátora. Týmto sa vytvára priestor pre rôzne bezpečnostné chyby, ktoré môže vytvoriť aj skúsený programátor.

Nette framework ponúka zjednodušenú možnosť ošetrenia prihlasovania a verifikácie prihlásenia užívateľa. Najprv je potrebné zo šablóny prostredníctvom filtru poslať informáciu do presenteru, aby sa daný komponent vykreslil. V tomto prípade pôjde o prihlasovací formulár.

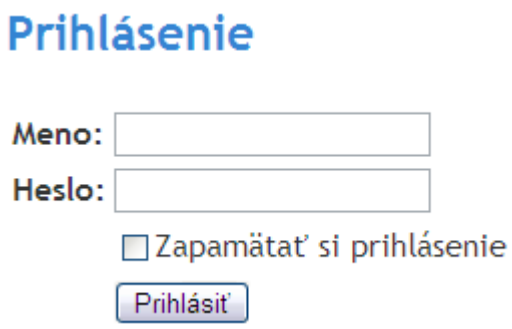
```
{control signInForm}
```

V príslušnom presenteri sa potom komponent vytvorí pomocou verejnej metódy.

```
protected function createComponentSignInForm()
{
   $form = new AppForm;
   $form->addText('username', 'Meno:')
        ->addRule(AppForm::FILLED, 'Prosím zadajte meno.');
   $form->addPassword('password', 'Heslo:')
        ->addRule(AppForm::FILLED, 'Prosím zadajte heslo.');
   $form->addCheckbox('remember', 'Zapamätať si prihlásenie');
   $form->addSubmit('send', 'Prihlásiť');
```

```
        $form->onSubmit[] = callback($this, 'signInFormSubmitted');
        return $form;
    }
```

Vytvorený formulár bude vyzerať nasledujúco (obr. 10):



Obr. 10 Prihlasovací formulár

Po úspešnej validácii na strane klienta je prihlasovací proces spracovávaný verejnou metódou *signInFormSubmitted*. Jej pomenovanie je ľubovoľné. Tu sa nastaví doba automatického odhlásenia užívateľa v závislosti od toho, či zaškrtol zapamätanie prihlásenia alebo nie. Štandardne sa expirácia nastavuje na dvadsať minút, alebo štrnásť dní pre zapamätanie.

```
public function signInFormSubmitted($form)
{
    try {
        $values = $form->getValues();
        if ($values['remember']) {
            $this->getUser()->setExpiration('+ 14 days', FALSE);
        } else {
            $this->getUser()->setExpiration('+ 20 minutes', TRUE);
        }
        $this->getUser()->login($values['username'],
                                $values['password']);
        $this->redirect('Homepage:');
    } catch (AuthenticationException $e) {
        $form->addError($e->getMessage());
    }
}
```

Ďalej sa v tejto funkcii vykoná samotné overovanie prihlásenia na strane serveru a to tak, že sa zavolá metóda *login*. Táto je prednastavene umiestnená v modeli *UsersModel* v triede, ktorá má rovnaký názov a implementuje rozhranie

Nette\Security\IAuthenticator. Overovanie voči databáze sa nachádza vo funkcii *authenticate*.

```
class UsersModel extends Nette\Object
                 implements Nette\Security\IAuthenticator
{
   public function authenticate(array $credentials)
   {
     ...
   }
}
```

Overovacia funkcia vracia objekt *Nette\Security\Identity*, v ktorom sú informácie o užívateľovi. Aké informácie sa v objekte nachádzajú, je plne v kompetencii programátora, ktorý ich do objektu priraďuje.

Po úspešnom prihlásení nasleduje metódou *signInFormSubmitted* presmerovanie na ľubovoľné miesto v aplikácii.

Pre overenie prihlásenia užívateľa sa používa metóda *isLoggedIn*. Pomocou tejto metódy sa overuje, či má užívateľ právo vidieť daný obsah. Napokon na odhlásenie sa použije metóda *logout*. V šablóne stačí vytvoriť odkaz:

```
<a href="{plink Sign:out}">odhlasit</a>
```

ktorý presmeruje užívateľa na presenter *Sign*, kde sa nachádza akcia *out*, ktorá užívateľa odhlási.

```
public function actionOut()
{
   $this->getUser()->logout();
   $this->flashMessage('Odhlásenie bolo úspešné.');
   $this->redirect('Homepage:default');
}
```

O úspešnom odhlásení je užívateľa vhodné informovať s využitím zabudovaných flash správ metódy *flashMessage*.

2.2.8. Zabudovanie AJAX technológie [4]

AJAX znamená asynchrónny JavaScript a XML. AJAX nie je nový programovací jazyk, ale technika využívajúca rôzne jazyky. Nie je potrebné striktne použiť XML, nevyžaduje sa ani asynchrónnosť. O spracovanie užívateľskej požiadavky sa zvyčajne

stará JavaScript. O jej spracovanie na strane serveru PHP alebo iný serverový jazyk a odpoveď vygenerovaná serverom môže mať tiež viacero formátov (HTML, XML, JSON).

AJAX komunikácia so serverom prebieha prostredníctvom JavaScriptového objektu XMLHttpRequest. Prostredníctvom tohto objektu môže JavaScript vymieňať dáta priamo so serverom bez potreby opätovného načítania stránky. AJAX využíva asynchrónnu výmenu dát (HTTP requests, t.j. HTTP požiadavku) medzi internetovým prehliadačom a serverom. AJAX umožňuje zmeniť iba časť stránky a získavať aktuálne potrebné informácie zo servera namiesto opätovného načítania celej stránky.

Nette framework umožňuje programátorovi používať súčasne alebo kombinovať aj iné nástroje na vývoj internetových aplikácií, preto nie je problém na tvorbu AJAX aplikácií použiť JavaScriptovú knižnicu jQuery.

2.2.8.1. AJAX/jQuery technológie [5], [6]

jQuery je ľahká croos-browser JavaScriptová knižnica vydaná v roku 2006 Johnom Resigom. Využitím knižnice sa zjednodušuje proces písania JavaScript kódu. Kód sa stáva kompatibilným so všetkými hlavnými webovými prehliadačmi. Prostredníctvom jQuery sa napísanie aplikácie podporujúcej AJAX technológiu veľmi zjednodušuje. Nie je potrebné písať špeciálnu funkciu na vytvorenie XMLHttpRequest objektu, ktorým JavaScript komunikuje so serverom. Objekt sa vytvorí prostredníctvom jQuery funkcie *$.ajax()*.

Takto vytvorenému objektu je možné potom priraďovať vlastnosti a testovať úspešnosť vrátenia dát zo servera. Napríklad odoslanie požiadavky na server metódou „POST“ a následné otestovanie vrátenia dát sa vykoná nasledujúco:

```
$.ajax({
   type: "POST",
   url: "spracuj_poziadavku.php",
   data: "submit=1&text=ahoj",
   //test uspesnosti spracovania
   success: function(){
         //vypisanie informacie uzivatelovi
         alert( "Dáta boli spracované );
   }
```

```
});
```

Príkladom môže byť newsletter mailing list, ktorý umožní užívateľovi sa prihlásiť k odberu noviniek zasielaných e-mailom. Základom je jednoduchý formulár, kam užívateľ vloží svoju e-mailovú adresu a meno. Formulár po upravení kaskádovými štýlmi môže vyzerať napríklad ako na obr. 11.

Obr. 11 Newsletter formulár

Po odoslaní formulára sa spustí v jQuery napísaná JavaScript funkcia, ktorá realizuje samotné spracovanie požiadavky. Pri tvorbe formulára je potrebné formuláru prideliť nejaký identifikátor, aby sa mohlo odosielanie spojiť s JavaScript funkciou. Do HTML sa napríklad zapíše:

```
<form id="ajaxNewsletterForm">
```

a v JavaSript funkcii sa identifikátor využije nasledujúco:

```
$(document).ready(function(){
   $("#ajaxNewsletterForm").submit(function(){
      // 'this' predstavuje aktualny odoslany formular
      var str = $(this).serialize();
      // pokračovanie funkcie .....
    });
});
```

Ďalej sa použije jQuery funkcia *ajax*, ktorá vykoná asynchrónnu HTTP AJAX požiadavku na server. Táto funkcia má viacero argumentov, ktorými sa zadefinuje typ odoslania, cesta k súboru, ktorý spracuje požiadavku, dáta odoslané na spracovanie a iné.

```
$.ajax({
   type: "POST",
   url: "kontakt.php",
   data: str,
   success: function(msg){
      $("#note").ajaxComplete(function(event, request, settings){
         // oznámenie o zrealizovaní požiadavku užívateľovi
      }
   });
```

```
});
```

Funkcia *success* je volaná ak požiadavka prebehla úspešne. Argument *msg* predstavuje dáta, ktoré server odoslal späť po spracovaní požiadavky. Funkcia *ajaxComplete* je globálna udalosť, ktorá vracia odpoveď serveru a je spustená vždy keď AJAX dokončí požiadavku na server.

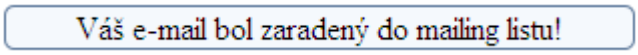
Na základe odpovede vrátenej zo serveru prostredníctvom súboru *kontakt.php* sa potom užívateľovi zobrazí informácia o priebehu požiadavky.

```
// sprava uspesne odoslana => skryje sa hlavny formular
if(msg == 'OK'){
   result = '<div id="ok">Váš e-mail bol zaradený do mailing
              listu!</div>';
   $("#fields").hide('fast',function(){
      // funkcia html() nahradi obsah DIVu "note" spravou (to je
      informacny DIV "result")
      $("#note").html(result);
      // najprv skryje element
      $("#ok").hide();
      // opat zviditelny element s efektom
      $("#ok").show('slow');
   });
}
// chybne vyplnene polia
else{
   $("#note").html(msg);
}
```

V prípade chybne vyplnených informácií užívateľ uvidí varovanie (obr. 12).

Obr. 12 Chybne vyplnený newsletter formulára

jQuery naviac umožňuje vytvoriť pri zobrazovaní informácií užívateľovi pekné efekty. Uvedený kód skryje po úspešnej validácii dát pôvodný formulár a zobrazí namiesto neho informáciu o odoslaní s pekným efektom (obr. 13).



Obr. 13 Newsletter formulár úspešne odoslaný

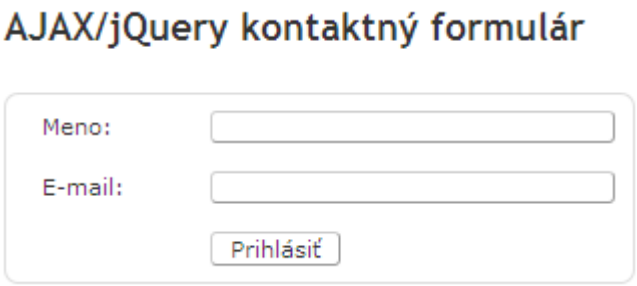
Celý proces verifikácie vložených dát a odosielania formulára prebehol asynchrónne prostredníctvom XMLHttpRequest AJAX objektu. Týmto sa aktualizovala iba časť stránky a nebolo potrebné obnovovať (načítať) celú stránku.

2.2.8.2. AJAX/jQuery technológie v prostredí Nette frameworku

jQuery knižnica sa zahrnie do Nette frameworku jednoduchým nakopírovaním knižnice do zvoleného adresára a nalinkovaním tejto knižnice do aplikácie nasledujúco:

```
<script type="text/javascript" src="{$basePath}/js/jquery.js">
</script>
```

Potom je potrebné v presenteri vytvoriť samotný formulár. Najlepšie prostredníctvom do Nette zabudovaných prostriedkov, čiže pomocou metódy *createComponent*. Takto vytvorený formulár však vyzerá veľmi obyčajne. Formulár je vhodné preto ešte upraviť zadefinovaním kaskádového štýlu, ktorým sa dosiahne, že bude vyzerať podľa potreby. Vytvorený môže teda vyzerať ako na obr. 14:



Obr. 14 jQuery kontaktný formulár

V metóde generujúcej formulár sa nastavia JavaScipt validačné pravidlá. Ak užívateľ zadá dáta v požadovanom formáte, je zavolaná funkcia *onSubmit*, ktorá predá riadenie zadefinovanej funkcii, ktorá spracuje odoslaný formulár.

```
$form->onSubmit[] = array($this, 'submitJqueryForm');
```

Vo funkcii *submitJqueryForm* sa zvyčajne zavolá funkcia zvoleného modelu, ktorá napríklad uloží dáta do databázy a vráti odpoveď o priebehu. Napríklad:

```
public function odoslatAjaxMail($entry)
{
   // … spracovanie dat …
```

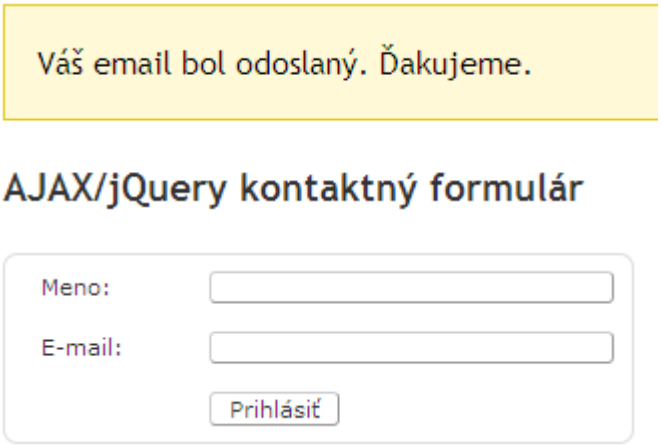
```
    if($spracovanie_prebehlo){
          return true;
    }else{
          return false;
    }
}
```

Predaním spracovania modelu sa dodržiava model MVP, kde sa o komunikáciu s databázou má starať model. Funkcia v presenteri spracovávajúca odpoveď môže vyzerať nasledujúco:

```
public function submitJqueryForm(Form $form)
{
    $entry = $form->getValues();

    if( ContactMailModel::odoslatAjaxMail($entry) )
    {
          $this->flashMessage('Váš email bol odoslaný. Ďakujeme.');
          if (!$this->isAjax()){
                $this->redirect('this');
          }
          else {
                $this->invalidateControl('formular');
                $form->setValues(array(), TRUE);
          }
    }else{
          $this->flashMessage('Váš email nebol odoslaný.');
    }
}
```

Jej úlohou vhodne informovať užívateľa o spracovaní požiadavky, najlepšie prostredníctvom zabudovaných rýchlych flash správ (obr. 15).



Obr. 15 jQuery kontaktný formulár odoslaný

Doteraz nebolo nikde zadefinované, aby komunikácia so serverom prebiehala prostredníctvom XMLHttpRequest AJAX objektu. Je potrebné si prostredníctvom jQuery napísať funkciu a túto uložiť do JavaScript súboru a prilinkovať do aplikácie.

```
$("form.ajaxjQueryForm").live("submit", function (callback) {
    $(this).ajaxSubmit();
    return false;
});
```

Vo funkcii sa teda iba zadefinuje aký formulár sa má spracovať prostredníctvom CSS selektoru *class*. jQuery funkcia *live* zabezpečí, že aj ďalšie pribudnuté elementy na stránke sa odošlú AJAXom. Táto samotná funkcia by však asynchrónne formulár na server neodoslala. Do aplikácie je potrebné prilinkovať dva súbory, ktoré sú k dispozícii na stiahnutie na oficiálnych stránkach Nette. Tieto súbory (*jquery.nette.js* a *jquery.ajaxform.js*) zabezpečia správnu funkčnosť jQuery pod frameworkom Nette a odosielanie formulárov metódou AJAX.

3. Praktická časť

3.1. Analýza problému

Cieľom diplomovej práce je vytvoriť internetovú aplikáciu, ktorá umožní prihlásenému administrátorovi zverejňovať dokumenty v elektronickej podobe na internetových stránkach.

Požiadavkou je vytvoriť štyri samostatné sekcie, v ktorých sú zverejňované zmluvy, objednávky, faktúry a výzvy na predkladanie návrhov v procese verejného obstarávania. Administrátor prostredníctvom príslušného formulára môže zverejniť dokument do príslušnej sekcie.

Možnosť zverejňovať zmluvy má mať iba prihlásený administrátor, ktorému má aplikácia umožniť si heslo meniť, prípadne obnoviť v prípade straty.

Administrátor má možnosť v prípade potreby dodatočne zverejnené informácie editovať, prípadne úplne odstrániť. V sekcii verejného obstarávania má aplikácia umožniť ukončené výberové konanie presunúť do archívu.

V procese tvorby aplikácie je možnosť zvoliť si viacero prístupov:

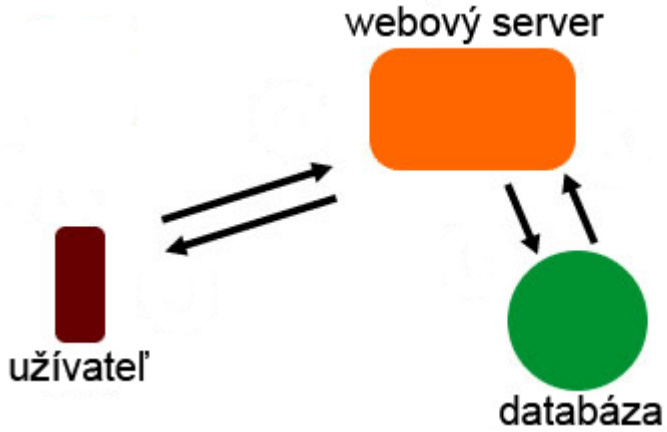
- Najzákladnejším prístupom je použiť techniku procedurálneho programovania a s využitím programovacieho jazyka PHP aplikáciu vytvoriť. Keďže PHP je možné ľahko začleniť do HTML, vznikne program, ktorý kombinuje skriptovací a programovací jazyk. Ako však mohutnosť aplikácie narastá, udržiavanie takto napísanej aplikácie sa stáva čoraz ťažšie. Takisto je tu eliminovaná možnosť práce viacerých ľudí na projekte súčasne, prípadne rozdeliť práce medzi dizajnéra a programátora.
- Ďalším možným prístupom je použiť šablónovací systém, ktorý má eliminovať práve problém udržiavania aplikácie. Napríklad systém Smarty umožňuje oddeliť logickú a prezentačnú časť aplikácie. Týmto umožňuje pracovať súčasne na projekte aj dizajnérovi a aj programátorovi. Programátor však môže použiť procedurálne programovanie, šablónovací systém programátora nenúti využiť techniku objektovo – orientovaného programovania. Týmto sa eliminuje možnosť práce viacerých programátorov na jedno projekte súčasne.

- Ďalšou možnosťou je použiť niektorý z dostupných frameworkov. Framework zabezpečí oddelenie prezentačnej a aplikačnej časti aplikácie a vedie programátora k písaniu kódu objektovo – orientovanou technikou. Výhodou využitia frameworku je možnosť zapojiť do práce paralelne viacero programátorov súčasne, ktorí pracujú na jednotlivých moduloch aplikácie.

Z uvedených možností bola využitá tretia možnosť. Na tvorbu aplikácie bol použitý framework Nette.

3.2. Databáza aplikácie

Databáza tvorí základ aplikácie. Databázy predstavujú sklad informácií, kam sú dáta ukladané a odkiaľ sú v prípade potreby spätne zobrazované užívateľovi. Aplikácie, ktoré ako úložisko dát používajú databázový systém sa vo všeobecnosti nazývajú dynamicky generované. Na obr. 16 vidieť všeobecný graf životného cyklu interakcie medzi užívateľom a aplikáciou.



Obr. 16 Životný cyklus interakcie

3.2.1. Rozhranie PDO [7],[8]

V aplikácii je použité rozhranie PDO. PDO je konzistentné rozhranie pre prístup k databázam prostredníctvom PHP. PDO vytvára dátovú abstraktnú vrstvu, ktorá umožní používať rovnaké funkcie pre prístup k dátam v databáze bez ohľadu na jej typ

[7]. PDO v súčasnosti podporuje najvýznamnejšie databázové systémy akými sú: MS SQL Server, MySQL, PostgreSQL, SQLite, Oracle, IBM, Informix a iné [8].

PDO umožňuje oddeliť logiku aplikácie od logiky komunikácie s databázou. Všetky funkcie pre prístup k dátam sú realizované prostredníctvom tohto rozhrania, jedinou podmienkou je, aby abstraktná vrstva obsahovala ovládač kompatibilný s daným databázovým systémom.

V samotnej aplikácii sa potom v abstraktnej triede modelu zadefinuje chránená metóda, ktorá vytvorí PDO objekt a vloží ju do privátnej dátovej premennej.

```
protected function getPDO(){
   // pripojenie sa k databaze
   try {
      $this->pdoAbstractDb = new PDO("mysql:dbname=$this->databaza;
                                      host=$this->server_db",
                                      $this->uzivatel_db,   $this-
                                      >heslo_db,   $this->array_opt
                                      );
      // nastavi tvorbu "exceptions" namiesto chyb
      $this->pdoAbstractDb    ->    setAttribute(PDO::ATTR_ERRMODE,
                                        PDO::ERRMODE_EXCEPTION);
      return $this->pdoAbstractDb;
   }catch(PDOException $e){
      throw new Exception('PDO DB zlyhala: '.$e->getMessage());
   }
}
```

V detských triedach potom dedením tejto metódy je PDO objekt pre prístup do databázy k dispozícii.

```
$this->pdo_db = parent::getPDO();
```

Výhodou vyseparovania spoločného predka do abstraktnej triedy je možnosť editovať prístupové informácie do databázy z jedného miesta.

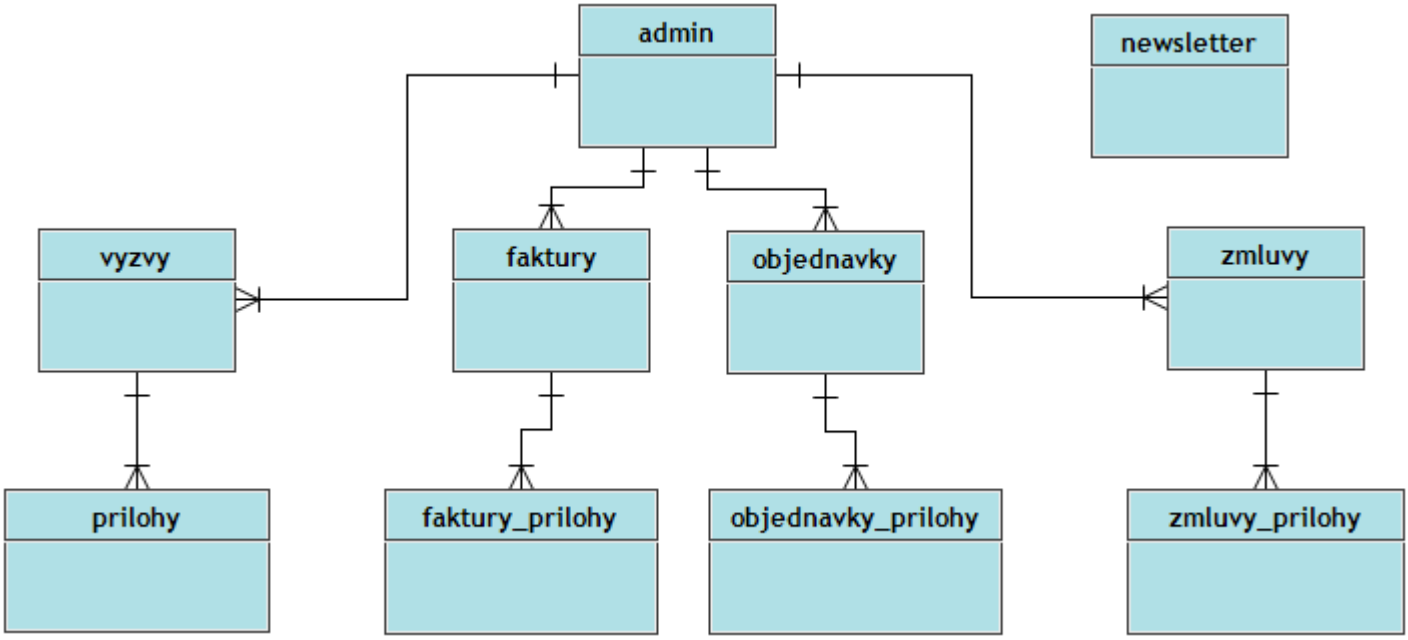
3.2.2. MySQL [9], [10]

Aplikácia využíva relačný databázový systém MySQL, ktorý je voľne dostupný, relačný databázový systém, postavený na jazyku SQL [9]. Prostredníctvom tohto systému je možné ukladať, meniť a mazať dáta.

MySQL je optimalizované pre viacero platforiem ako sú: Linux, Microsoft Windows, Mac OS X, Solaris, OpenBSD a iné [10]. MySQL takisto ponúka široké možnosti konfigurácie databáz a ich zabezpečenia.

Pre vytvorenie žiadaných funkcií aplikácie je potrebné použiť desať tabuliek, ktoré sú medzi sebou v rôznom relačnom vzťahu prostredníctvom kľúčov.

Na obr. 17 je entitno – relačný diagram pre celú databázu aplikácie.



Obr. 17 Entitno-relačný diagram databázy

Z obr. 17 vidieť, že tabuľka *admin* je v relačnom vzťahu 1:N so štyrmi sekciami aplikácie. To znamená, že aplikácia si ukladá údaje o tom, aký administrátor dokumenty zverejnil.

Každá sekcia má vlastnú tabuľku na ukladanie dokumentov v relačnom vzťahu 1:N. To znamená, že ku každému oznamu je možné nahrať minimálne jeden dokument. Maximálne množstvo nie je limitované.

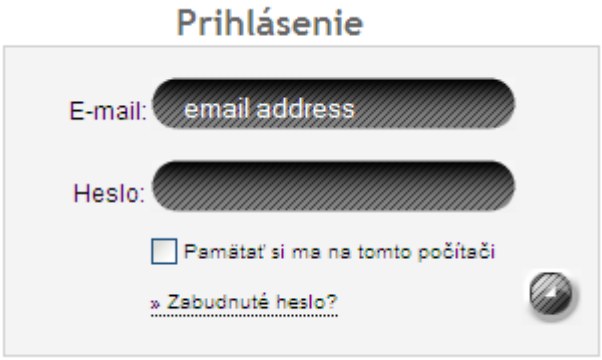
Samostatne stojaca tabuľka *newsletter* obsahuje informácie o užívateľoch, ktorý sa prihlásili k odberu noviniek prostredníctvom e-mailu.

3.3. Administrácia aplikácie

Administračné rozhranie by malo byť pokiaľ možno čo najprehľadnejšie a intuitívne. Rozhranie umožňuje administrátorovi zverejňovať dokumenty do zvolených sekcií aplikácie.

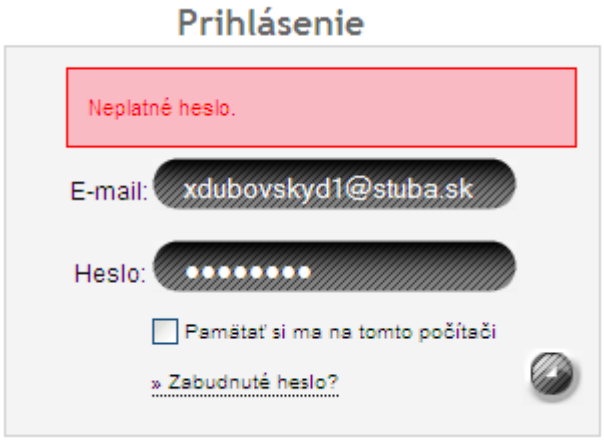
3.3.1. Prihlásenie administrátora

Dokumenty môže zverejňovať iba prihlásený administrátor. Administrátor sa prihlasuje prostredníctvom formulára (obr. 18)



Obr. 18 Prihlásenie administrátora

Pri prihlasovaní má možnosť zaškrtnúť políčko, aby zostal prihlásený dlhšie ako pol hodinu. Po pol hodine bez aktivity, je totiž automaticky odhlásený. Ak políčko zaškrtne, je prihlásený na dobu štrnásť dní a ostáva prihlásený, aj keď zatvorí prehliadač. V prípade chybne zadaných prihlasovacích údajov je administrátor upozornený (obr. 19)



Obr. 19 Chybné prihlásenie administrátora

Po úspešnom prihlásení je administrátor automaticky presmerovaný na úvodnú stránku rozhrania (obr. 20).

Obr. 20 Administračné rozhranie

V prípade ak si administrátor nemôže spomenúť na heslo, aplikácia ponúka možnosť si heslo obnoviť (obr. 21).

Obr. 21 Obnovenie hesla

Do obnovovacieho políčka je potrebné zadať meno, priezvisko a e-mail administrátora. Prebehne „case sensitive“ porovnanie voči údajom v databáze a v prípade zhody všetkých troch údajov, je na zadaný e-mail odoslané nové heslo. Administrátor je o zmene informovaný (obr. 22).

Obr. 22 Potvrdenie obnovy hesla

Po otvorení svojej e-mailovej schránky, si nájde v doručenej pošte nové heslo.

Administrátor si môže heslo aj zmeniť. Je potrebné zadať pôvodné heslo a potom dvakrát heslo nové (obr. 23). Potvrdenie nového hesla je ochrana proti preklepom.

Obr. 23 Zmena hesla

O úspešnej zmene hesla je administrátor opäť informovaný (obr. 24).

Obr. 24 Potvrdenie zmeny hesla

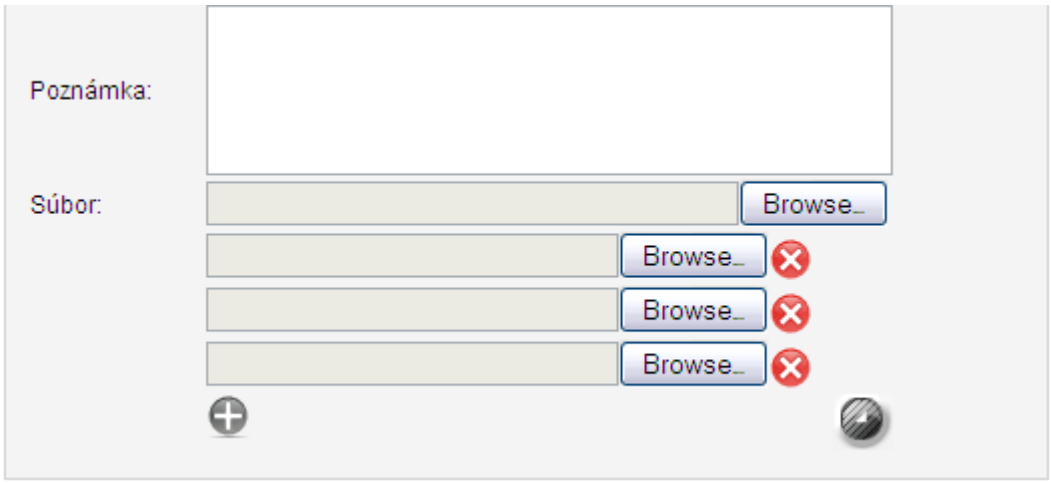
3.3.2. Publikovanie zmluvy, faktúry, objednávky

Administrátor môže zverejniť informácie aj s príslušnými dokumentmi o zrealizovaných objednávkach, vyplatených faktúrach alebo uzatvorených zmluvách. V základnom administračnom rozhraní si kliknutím na ikonu zvolí do akej sekcie chce údaje zverejniť. Otvorí sa príslušný formulár (obr. 25)

Obr. 25 Formulár na zverejnenie faktúry

Formuláre pre jednotlivé typy zverejňovaných informácií sa mierne líšia, ale ich podstata ostáva rovnaká. Vždy je potrebné vyplniť všetky informácie, okrem poznámky. Poznámka má len doplňujúci informačný charakter pre návštevníka stránok.

Pre všetky tri prípady je povinné nahrať aspoň jeden súbor, v ktorom sú zvyčajne zhrnuté informácie, ktoré administrátor vložil do internetovej stránky. Prípadne tento dokument reprezentuje samotnú faktúru, zmluvu alebo objednávku, keďže dnes sú takéto dokumenty vyhotovované takmer vždy v elektronickej podobe (súbor typu pdf, doc, xls, txt a iné.). Návštevník si potom tieto informácie môže stiahnuť a nemusí si ukladať alebo tlačiť webovú stránku. Ako prílohu je možné nahrať aj viacero súborov naraz a to jednoduchým kliknutím na ikonu „plus“ (obr. 26).



Obr. 26 Formulár na zverejnenie faktúry s viacerými prílohami

Po vložení je administrátor informovaný o úspešnom zverejnení. Zverejnené informácie sú automaticky zobrazené v príslušnej časti internetovej aplikácie, kde si ich každý návštevník môže pozrieť (obr. 27).

Obr. 27 Zverejnené faktúry

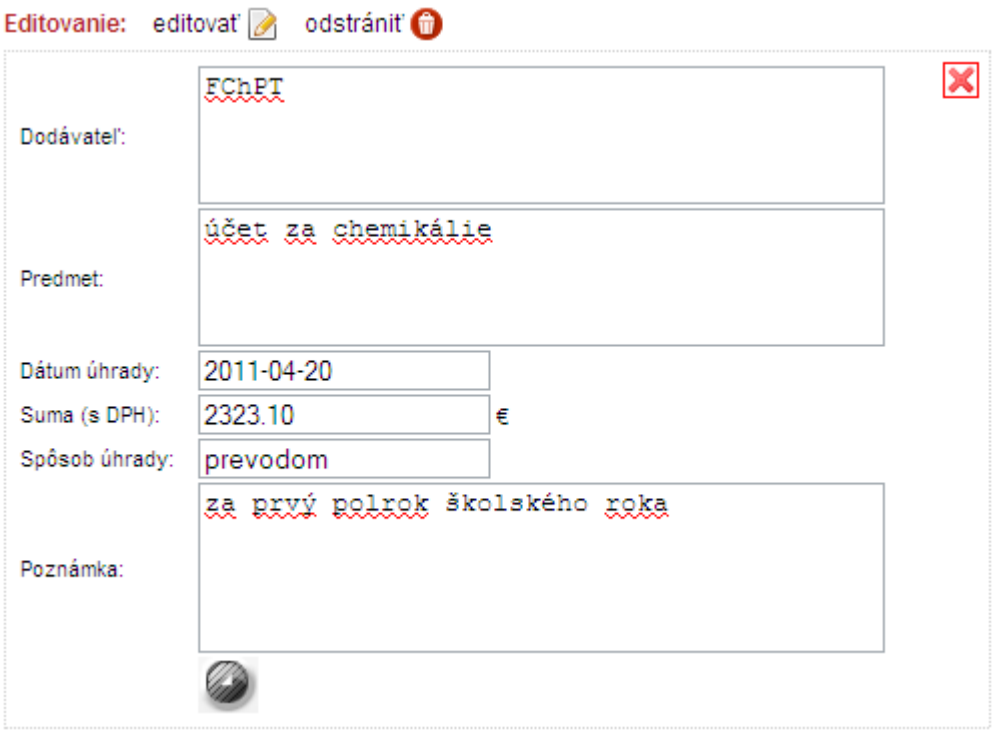
Aplikácia limituje množstvo zverejnených záznamov na päť na jednu stránku a potom dôjde k automatickému stránkovaniu, aby nevznikala veľmi dlhá stránka.

Zverejnený záznam je možné v prípade preklepov alebo zmien údajov dodatočne editovať kliknutím na ikonu (obr. 28). Otváranie a zatváranie editačného formulára je zrealizované prostredníctvom knižnice jQuery, aby nedochádzalo k zbytočnému obnovovaniu stavu aplikácie. V šablóne je umiestnený HTML kód aj s identifikátormi:

```
<span class="editovanie_signal">
   <img src="{$basePath}/obr/edit.png" alt="editovať"
        title="editovať" class="editovat_pic"
        id="{$iterator->getCounter()}" />
</span>
```

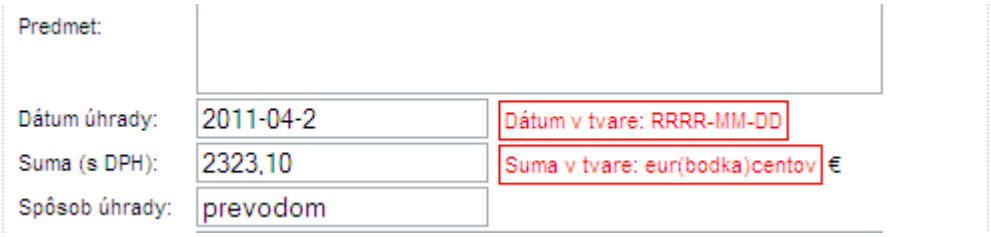
a jQuery prostredníctvom identifikátorov pristupuje k jednotlivým prvkom šablóny:

```
// zobrazi editacny formular
$(".editovat_pic").click(function(){
   var id = $(this).attr("id");
   // zobrazi formular
   $("#edit_form_"+id).slideDown('slow');
});
// zatvori editacny formular
$(".editovat_close").click(function(){
   var id = $(this).attr("id");
   $("#edit_form_"+id).hide('slow');
});
```



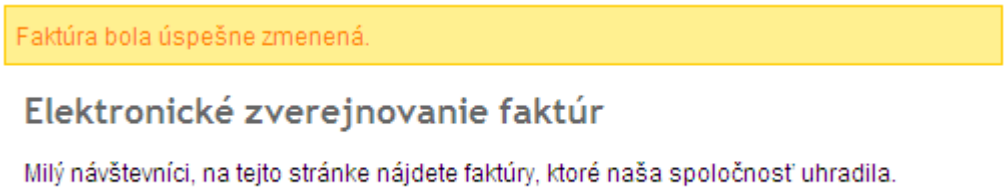
Obr. 28 Editačný formulár faktúry

Aj v prípade editovania je pomocou jQuery zrealizovaná kontrola editovaných údajov na strane klienta, aby sa do databázy nedostali dáta v nevhodnom tvare (obr. 29)



Obr. 29 Kontrola editovaných údajov

O spracovanie formulára sa postará metóda, na ktorú sa odkazuje ako na signál presenteru zo šablóny. Metóda odovzdá dáta do modelu, ktorý aktualizuje údaje v databáze a vracia návratovú hodnotu metóde. Táto na jej základe informuje administrátora o dokončení požiadavky (obr. 30).



Obr. 30 Potvrdenie o aktualizácii faktúry

Zverejnený záznam je možné aj odstrániť kliknutím na príslušnú ikonu. Požiadavka na odstránenie je opäť signál na presenter. Tuto je však vhodné vyzvať administrátora na potvrdenie odstránenia záznamu.

```
<a href="{link vymazat!, id => $zmluva['zmluvy_id'], co => 2}"
   n:confirm = "Naozaj chcete zmazať zmluvu:
                {$zmluva['zmluvy_predmet']} ?" >
   <img src="{$basePath}/obr/remove_item.png" alt="vymazať výzvu"
        title="vymazať výzvu" />
</a>
```

Spustenie potvrdzovacieho okienka (obr. 31) zabezpečí jQuery funkcia *trigger*, ktorá je aktivovaná po kliknutí na príslušnú ikonku.

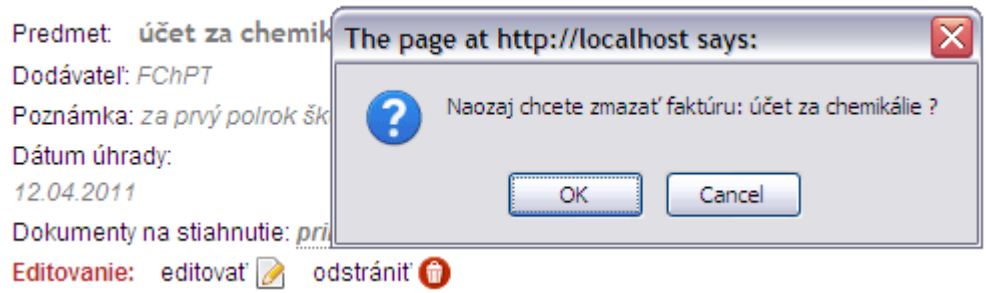
```
// potvrdenie vymazania
$.fn.extend({
   triggerAndReturn: function (name, data) {
      var event = new $.Event(name);
      this.trigger(event, data);
      return event.result !== false;
   }
```

```
});

$('a[data-confirm], button[data-confirm],
   input[data-confirm]').live('click', function (e)
{
   var el = $(this);
   if (el.triggerAndReturn('confirm')) {
      if (!confirm(el.attr('data-confirm'))) {
         return false;
      }
   }
});
```

Do šablóny je potrebné si zadefinovať vlastné makro a to najlepšie v *startup* funkcii, ktorá je vyvolaná na začiatku životného cyklu presenteru.

```
public function startup(){
   parent::startup();
   // zaregistruje vlastne makro
   LatteMacros::$defaultMacros['@confirm'] =
      'data-confirm="<?php echo %:formatString%; ?>"';
}
```



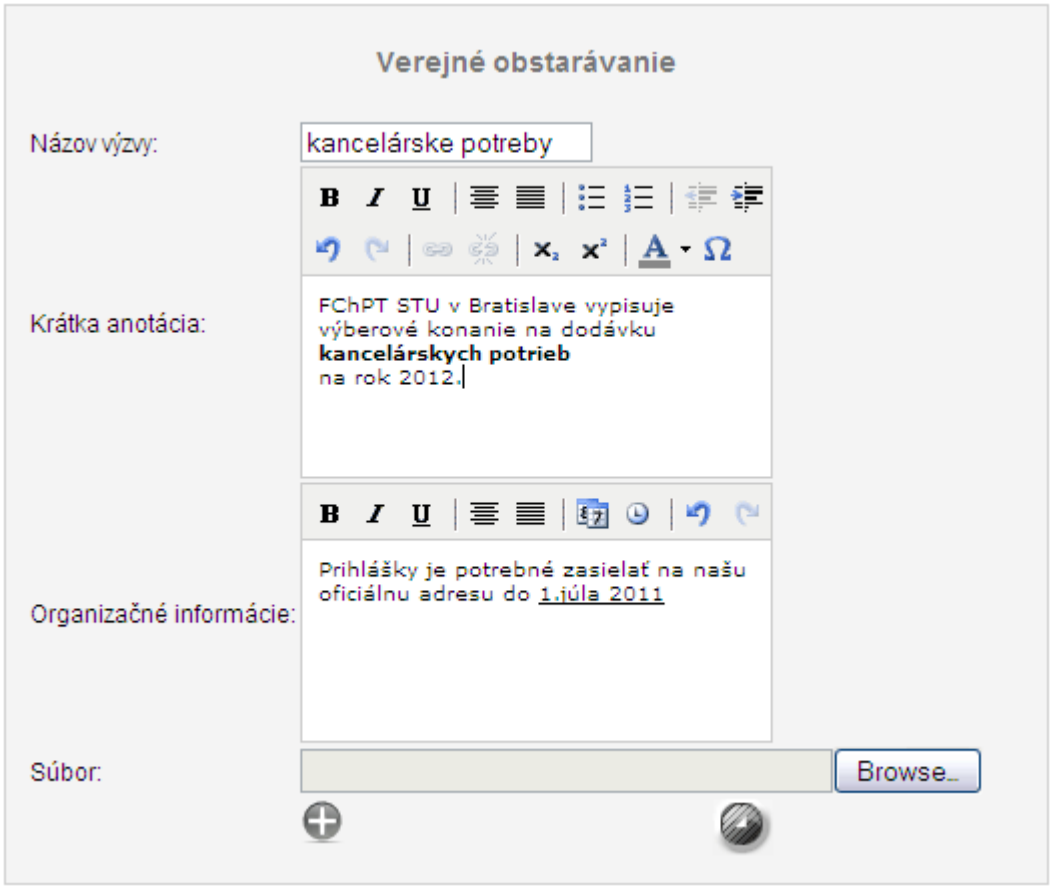
Obr. 31 Odstránenie faktúry

Rovnakým spôsobom, akým sa pracuje s faktúrami sa pracuje v ostatných sekciách aj so zmluvami a objednávkami.

3.3.3. Verejné obstarávanie [11]

Samostatne stojacou sekciou je verejné obstarávanie. Po kliknutí na ikonu v administračnej zóne sa otvorí formulár (obr. 32), do ktorého administrátor zadáva názov výzvy, krátku anotáciu a rôzne technicko-organizačné informácie. Takisto je

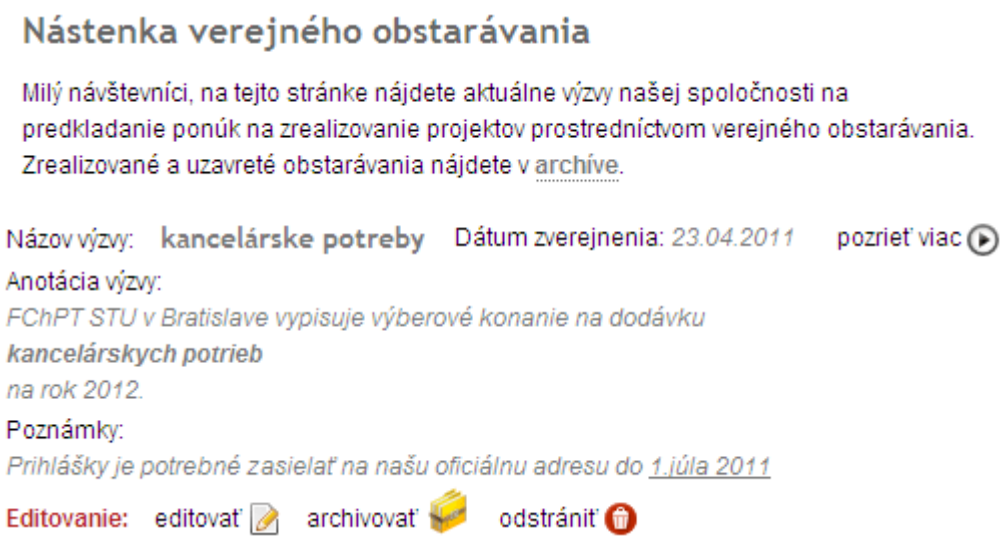
povinný nahrať minimálne jeden súbor, kde by mali byť zhrnuté podmienky súťaže a ktorý si bude môcť záujemca stiahnuť.



Obr. 32 Formulár pre verejné obstarávanie

V prípade zadávania informácií do výzvy má administrátor možnosť vkladané údaje editovať pomocou menu, ktoré svojou podobou pripomína štandardné kancelárske softvérové balíky (obr. 32). Editovacie menu je javascriptová trieda s názvom TinyMCE od spoločnosti moxiecode, ktorá je voľne dostupná a je ľahko zakomponovateľná do šablóny.

Po vložení údajov a ich nahraní na server sa v sekcii verejné obstarávanie automaticky táto výzva objaví (obr. 33).

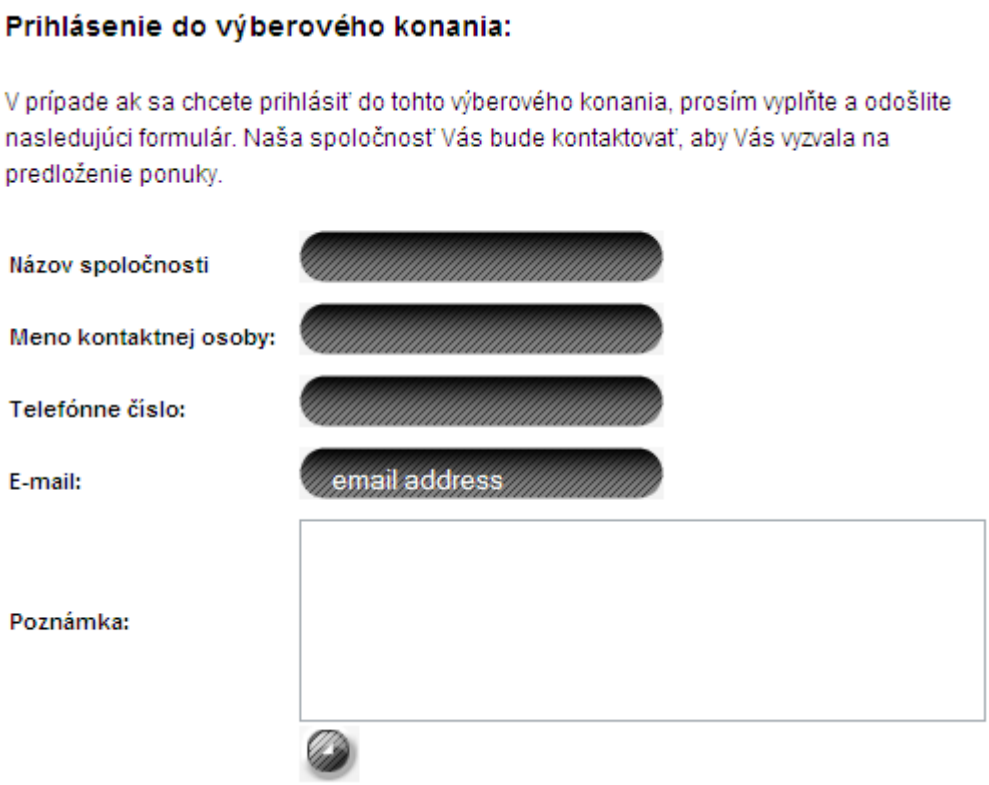


Obr. 33 Zverejnená výzva verejného obstarávania

Prihlásený administrátor môže aj v tomto prípade výzvu dodatočne editovať, prípadne odstrániť. V prípade ukončenia výberového konania, kliknutím na ikonu „archivovať“ sa výzva presunie do archívu. Archív je zoznam prebehnutých výberových konaní. V prípade odstraňovania aj archivácie musí administrátor akciu potvrdiť prostredníctvom javascriptového okienka.

Kliknutím na ikonu detail sa návštevník dostane na stránku s kompletnými informáciami o výzve, kde sa nachádzajú súbory na stiahnutie a aj registračný formulár (obr. 34).

Po vyplnení a odoslaní prihlášky príde na zvolenú poštovú adresu poverenému zamestnancovi správa s informáciami o prihlásenom účastníkovi. Potvrdzujúca správa príde aj prihlásenému účastníkovi. Poverený zamestnanec si potom vedie agendu prihlásených účastníkov, komunikuje s nimi a stará sa o celý proces obstarávania podľa vlastného uváženia.



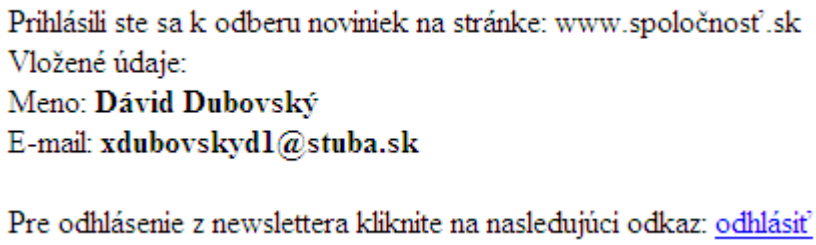
Obr. 34 Prihlásenie do výberového konania

3.3.4. Hromadná korešpondencia

Akýkoľvek návštevník stránok má možnosť prihlásiť sa k odberu noviniek, ktoré ho môžu informovať napríklad o plánovaných alebo aktuálne vypísaných súťaží, ktoré je potrebné zrealizovať formou verejného obstarávania. Forma a spôsob informovania je v kompetencii administrátora aplikácie. Návštevník vyplní na stránkach registračný formulár (obr. 35).

Obr. 35 Prihlásenie k odberu noviniek

O úspešnom zaradení do zoznamu je informovaný potvrdzujúcou správou, ktorá sa zobrazí na stránke a ešte aj emailovou správou na zadaný email, kde je k dispozícii aj odkaz na odhlásenie sa z odberu noviniek (obr. 36).

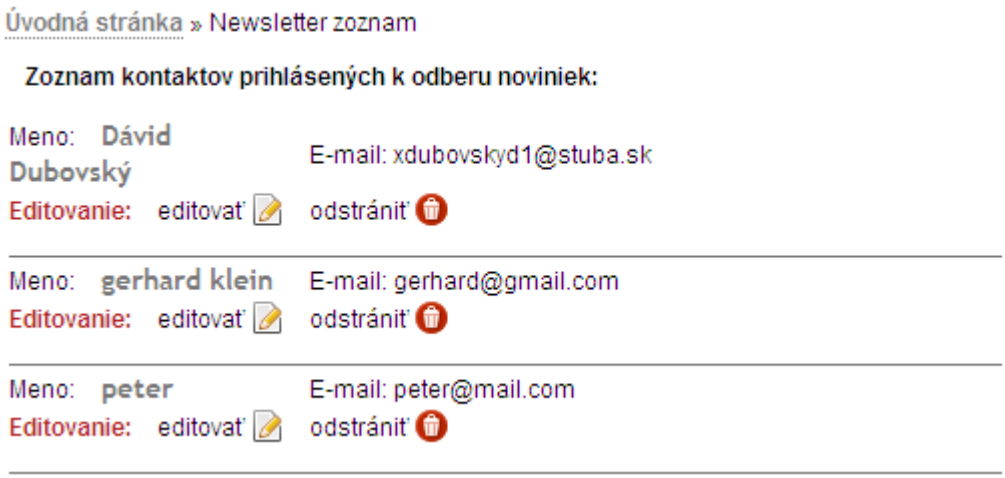

Prihlásili ste sa k odberu noviniek na stránke: www.spoločnosť.sk
Vložené údaje:
Meno: **Dávid Dubovský**
E-mail: **xdubovskyd1@stuba.sk**

Pre odhlásenie z newslettera kliknite na nasledujúci odkaz: odhlásiť

Obr. 36 Potvrdzujúci email

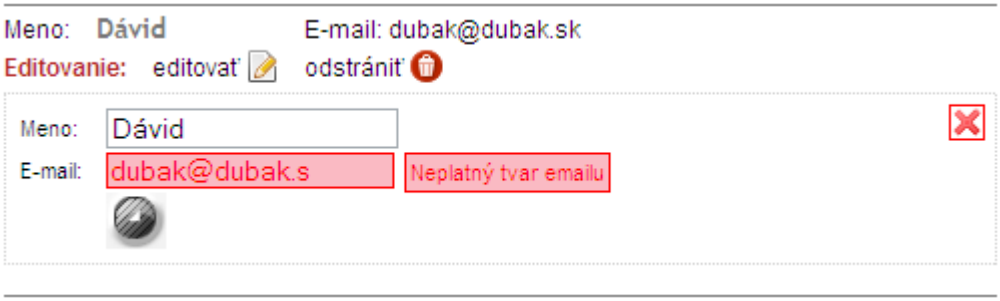
Ak sa užívateľ rozhodne odhlásiť z odberu noviniek, kliknutím na odkaz v potvrdzujúcom emaily je presmerovaný na stránku aplikácie, kde sa daný úkon vykoná a užívateľovi sa zobrazí potvrdzujúca informácia.

O prihlásení nového odberateľa je emailom informovaný aj administrátor stránok, aby mohol prípadne skontrolovať vložené dáta o užívateľovi. Prihlásený administrátor má teda právo daný zoznam čistiť od neplatných záznamov alebo upravovať v prípade preklepov, ako napríklad „meno@gmail.sk" (obr. 37).



Obr. 37 Zoznam kontaktov v hromadnej pošte

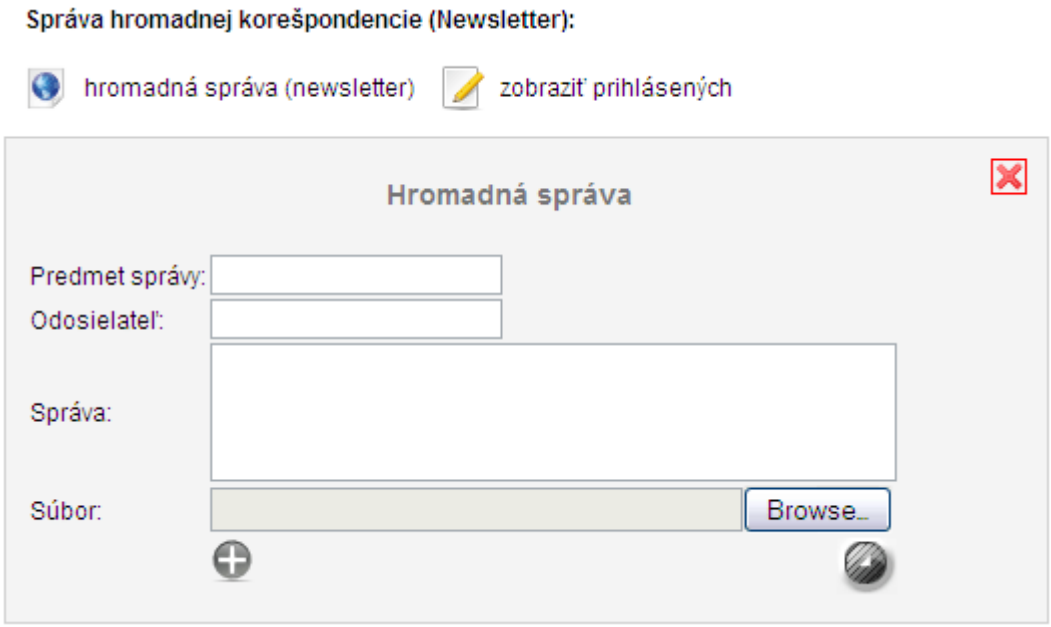
Pred vymazaním kontaktu zo zoznamu je administrátor vyzvaný na potvrdenia akcie, aby nedošlo k náhodnému odstráneniu. Editovanie je zrealizované prostredníctvom formulára, ktorý sa zobrazí po kliknutí na ikonu „editovať" (obr. 38).



Obr. 38 Editovanie kontaktu v hromadnej pošte

Pred odoslaním formulára sú dáta upravené administrátorom skontrolované pomocou javascriptových funkcií, aby nedošlo k vloženiu nevhodných dát.

Administrátor má možnosť odosielať hromadné správy na všetky kontakty, ktoré sú uložené v databáze. Hromadná korešpondencia je veľmi dobrý spôsob ako informovať užívateľov prihlásených k odberu noviniek. Po kliknutí na príslušnú ikonu sa administrátorovi otvorí formulár, prostredníctvom ktorého hromadný email odošle (obr. 39).



Obr. 39 Odosielanie hromadného emailu

Administrátor opäť môže odoslať poslať jednu alebo viacero príloh, pričom v tomto prípade nieje povinnosť nahrať aspoň jednu prílohu. Správnosť vložených údajov je pred odoslaním opäť skontrolovaná pomocou javascriptu. Do položky „Odosielateľ“ sa uvádza emailová adresa, ktorá sa užívateľovi zobrazí ako tá, z ktorej bol email

odoslaný. Po úspešnom odoslaní hromadnej správy je administrátorovi zobrazená potvrdzujúca informácia (obr. 40).

Obr. 40 Potvrdenie odoslania hromadnej pošty

4. Záver

Cieľom diplomovej práce bolo vytvoriť internetovú aplikáciu, ktorá umožní administrátorovi publikovať na Internete uzavreté zmluvy, zrealizované objednávky, vyplatené faktúry a najmä výzvy na predkladanie návrhov v procese verejného obstarávania. Administrátor prostredníctvom príslušného formulára môže zverejniť dokument do príslušnej sekcie aplikácie.

Samotné programátorské práce mohli byť zrealizované viacerými spôsobmi. Napokon bol využitý PHP framework Nette, ktorý oddeľuje logickú a prezentačnú časť aplikácie a podporuje písanie aplikácie technikou objektovo orientovaného programovania.

Po výbere vývojového prostredia nasledovalo zadefinovanie funkčnosti aplikácie a na základe toho aj výber a návrh databázového systému. Bol použitý systém MySQL.

Aplikácia umožňuje prihlásenému administrátorovi publikovať dokumenty do štyroch základných sekcií: verejné obstarávanie, zmluvy, faktúry a objednávky. Administrátor vloží krátky opis zverejňovaného dokumentu a nahrá na server samotný dokument, aby si ho mohol užívateľ v prípade potreby stiahnuť. Aplikácia umožňuje dodatočné editovanie, prípadné úplné odstránenie zverejneného dokumentu.

V sekcii verejného obstarávania sa nachádza formulár, prostredníctvom ktorého sa môže záujemca prihlásiť do výberového konania ku konkrétnej výzve. Výberové konania, ktoré už prebehli, má možnosť administrátor presunúť do archívu.

Aplikácia tiež umožňuje zber kontaktov od užívateľov a následne môže administrátor na tieto kontakty odoslať hromadnú správu. Takisto môže dodatočne zoznam kontaktov editovať.

Zoznam použitej literatúry

[1] http://zdrojak.root.cz/clanky/prezentacni-vzory-zrodiny-mvc

[2] http://doc.nette.org/cs/model-view-presenter

[3] http://davidgrudl.com

[4] http://en.wikipedia.org/wiki/JQuery

[5] http://api.jquery.com/

[6] http://api.jquery.com/jQuery.ajax

[7] http://sk.php.net/manual/en/intro.php

[8] http://sk.php.net/manual/en/pdo.drivers.php

[9] http://en.wikipedia.org/wiki/Mysql

[10] Velká kniha PHP a MySQL 5, W. J. Gilmore, Zoner Press, 2006, ISBN: 80-86815-53-6, 845 s.

[11] http://tinymce.moxiecode.com