Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Prispôsobiteľný Widget

(dokumentácia k 5. šprintu)

Tím č.10 : the 6_p@ck

Bc. Michal Immer
Bc. Jakub Korch
Bc. Jozef Macho
Bc. Peter Petriľák
Bc. Igor Repka
Bc. Ján Sivuľka

Študijný program: Softvérové inžinierstvo/Informačné systémy
Ročník: 1.ročník inžinierskeho štúdia
Predmet: Tímový projekt
Vedúci projektu: Ing. Tomáš Kuzár
Ak. rok: 2010/11

Piaty šprint

Piaty šprint začínal 29. novembra 2010 a jeho koniec bol stanovený na 13. decembra 2010. V tomto šprinte sme si dali za cieľ dorobiť Widgetizér a widget tak, aby bol funkčný a použiteľný vzhľadom na to, že v minulých šprintoch sa vyskytli chyby, ktoré znemožňovali použitie samotného widgetu. Keďže je tento šprint posledným v prebiehajúcom semestri, bolo jednou z hlavných úloh aj dopracovanie dokumentácií k jednotlivým šprintom a dokumentácie k riadeniu.

„User story“

Zadávateľ chce mať možnosť nastaviť vzhľad widgetu tak, aby vyhovoval veľkej časti stránok – napr. použitím rozumných štýlov. Ďalej požaduje, aby boli vo widgete viditeľné položky názov, dátum a miesto konania udalosti. Cieľom je aj zakomponovať AlchemyAPI do Nette aplikácie tak, aby bolo možné automaticky vygenerovať kategórie zo zadaného RSS zdroja, čo je však iba experimentálna funkcionalita.

Analýza

Filtrovanie dát z RSS

Obsah widgetu je nutné vyplniť na základe používateľom navolených parametrov. Tie sa posielajú volaním funkcií widgetu z používateľovej stránky metódou GET. Funkcie, ktoré sú zavolané vyfiltrujú požadované dáta z RSS a zobrazia ich vo widgete na používateľovej stránke.

Analýza možností cacheovania dát z rss

Hlavným dôvodom, prečo je potrebné uvažovať cachovanie dát pre widget je faktor, že počet používateľov widgetu môže v budúcnosti enormne narásť. V takom prípade by musel náš server obsluhovať veľké množstvo requestov, čo by nemusel celkom dobre zvládať. V súčasnosti je widget naprogramovaný tak, že sa pri každom requeste používateľa na zobrazenie widgetu pošle požidavka na získanie dát podľa vyfiltrovaných kategórií na náš server. Na serveri sa zavolá funkcia, ktorá zakaždým nanovo parsuje požadovaný dátový zdroj vo forme xml rss súboru. Takáto operácia je však neefektívna. Zatiaľ sa jej výhody neprejavujú, avšak s pribúdajúcim počtom používateľov widgetu a tým pádom requestov na

náš server budú viditeľnejšie jej nevýhody. Preto je cieľom analyzovať možnosti vylepšenia súčasného spôsobu implementácie.

Kategorizácia textu

V predošlom šprinte bola jednou z úloh aj kategorizácia zadaného textu, teda na základe zvoleného textu odoslať požiadavku na AlchemyAPI, ktoré vráti odpoveď v podobe zistenej kategórie z textu a informačných elementov. Táto kategorizácia však mala na naše pomery jednu veľkú nevýhodu a tou bola neschopnosť kategorizovať text napísaný v slovenskom jazyku. Táto nevýhoda sa naplno prejavila pri zadaní úlohy kategorizácie správ z dátového zdroja, kde sú uložené udalosti v slovenskom jazyku, a keďže widget je momentálne orientovaný hlavne na udalosti zo Slovenskej republiky, prišlo k požiadavke kategorizovať text priamo zo slovenského jazyka. K tomuto je potrebné previesť si zdrojový text v slovenskom jazyku na text v anglickom jazyku(resp. v inom rozšírenom jazyku, ktorý vie AlchemyAPI kategorizovať). Na prevod zo slovenského jazyka do iného jazyka je vhodným riešením využitie Google Translate API, ktoré poskytuje jednoduchú funkcionalitu využitím, podobne ako tomu bolo pri AlchemyAPI, REST webovej služby. Táto služba je najčastejšie používaná vývojármi, ktorí si chcú preložiť nejaké cudzojazyčné texty na svoju stránku.
Google translate vracia po odoslaní korektnej požiadavky a úspešnom preklade textu odpoveď vo formáte JSON.

Dátový zdroj

Keďže sme sa rozhodli v záverečnej časti nášho tímového projektu venovať iba jednému dátovému zdroju, z ktorého budeme získavať dáta, bolo potrebné analyzovať naše možnosti. Podstatné bolo splnenie požiadavky, aby mal používateľ hneď po načítaní našej stránky s widgetom k dispozícii vyparsovaný zoznam kategórií, z ktorých si môže vybrať tie, ktoré sa mu budú zobrazovať vo widgete. Týmto sme takpovediac vynechali úvodnú obrazovku, kde sa volil dátový zdroj a zobrazil používateľovi rovno ďalšiu obrazovku – výber kategórie. Chceli sme docieliť väčšiu kontrolu nad údajmi, ktoré sa používateľovi zobrazujú.

Zamerali sme sa teda hlavne na možnosti ako docieliť čo najlepšiu funkcionalitu a používateľskú jednoduchosť pri získavaní, zobrazovaní a výbere získaných kategórií z nášho zdroja, ktorý predstavuje vopred definovaný RSS súbor umiestnený na webe. Museli sme vyriešiť to, že tento dátový zdroj sa bude určite v budúcnosti meniť a otázne bolo, či budeme mať stále definovaný iba určitý počet kategórií a či sa tieto kategórie budú časom odlišovať. Vopred definovaný maximálny počet kategórií, ako aj ich konštantnosť (napríklad: vždy by sme mali k dispozícii iba dve kategórie – miesto a deň v týždni, z ktorých by si

mohol používateľ vybrať) by boli určite limitujúce a do budúcna nepoužiteľné, preto sme túto variantu zavrhli. Priklonili sme sa k riešeniu, ktoré nám ponúka viacero možností ako aj efektivitu a hlavne väčšiu spokojnosť zákazníka.

Návrh

Databázový návrh

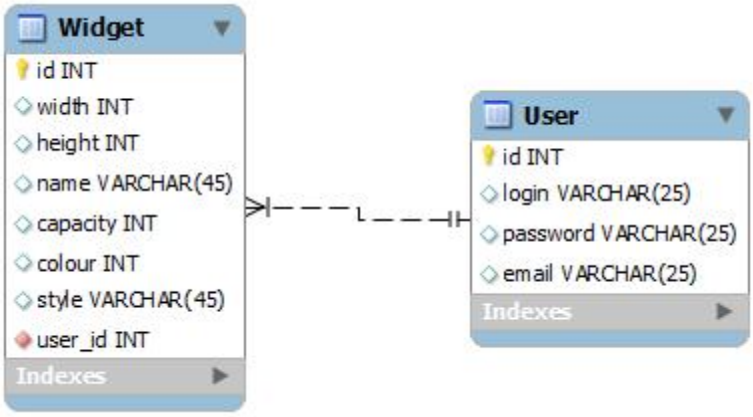
Vzhľadom na analýzu požiadaviek sme dospeli k poznatku, že v projekte bude nutné použiť databázu. Rozsah tohto použitia ešte nie je celkom rozhodnutý, je však istý minimálny rozsah, na čo ju určite budeme potrebovať.

Keďže chceme, aby bolo možné viesť evidenciu používateľov, je nutné zahrnúť ich do databázy. Pre začiatok sme sa dohodli, že každý používateľ je pre začiatok reprezentovaný svojim emailom. Tabuľka používateľov (**User**) ďalej obsahuje používateľské meno a heslo, pretože chceme, aby bolo možné prihlasovať sa do svojho účtu a prezerať svoje vytvorené widgety.

Tým sa dostávame k druhej tabuľke počiatočného primitívneho dátového modelu – **Widget**. Tá obsahuje informácie o danom widgete, všetky možné parametre (farba, veľkosť). Obsahuje tiež aj stĺpec s odkazom na vlastníka daného widgetu (používateľa). Túto tabuľku bude treba postupne rozširovať o ďalšie stĺpce, podľa toho, ako budeme rozširovať možnosti konfigurácie widgetu.

Prvotný fyzický dátový model bude preto veľmi jednoduchý, avšak postačujúci našej veci. Je zobrazený na nasledujúcom obrázku.

Obrázok č.1: Fyzický dátový model



Vzťah medzi týmito dvoma tabuľkami je 1:n, pretože používateľ môže mať niekoľko svojich widgetov.

Filtrovanie dát z RSS

Cieľom je, aby sa používateľom navolené dáta vyfiltrovali z RSS a zobrazili sa vo widgete na používateľovej stránke. Widget si zo stránky používateľa zavolá funkcionalitu na našom serveri. Tu sa na základe parametrov v GET metóde vyfiltrujú požadované dáta a sú vrátené na zobrazenie. Na serveri funkcie prechádzajú všetky parametre, ktoré prišli v metóde GET. Tie, ktoré sa týkajú dát widgetu sú polia. Vzhľadom na to, že premenná GET je tiež pole sa ňou môžeme iterovať. Pri každej iterácii sa pýtame, či premenná v poly GET je takisto pole. Ak áno spúšťa sa logika filtrovania dát z RSS. Premenná sa vyberie z poľa GET a začíname iteráciu celým RSS súborom. Konkrétne nad elementmi item. Z tohto elementu vyberieme všetky elementy category a porovnávame, či typ dát, ktorý nesú je zhodný s typom dát obsiahnutých v poly získanom z poľa GET a na základe ktorého prebehne filtrovanie. V prípade, že sa typy zhodujú dôjde k porovnaniu samotných hodnôt v elemente category daného typu a v poly získaného z poľa GET. Ak sa obsahy zhodujú do poľa indexov sa uloží index (miesto výskytu) elementu item. Cyklus prejde do ďalšieho kroku nad novým poľom dát získaného z metódy GET, ale tento krát sa už neprechádza celý RSS súbor, ale len tie elementy item, ktorých indexy sme odchytili a uložili do poľa indexov (elementy item, v ktorých už boli nájdené požadované filtrované parametre). Porovnanie prebieha rovnako ako v prvom kroku. To znamená, že z poľa indexov vzniká užšie pole indexov, ktoré obsahuje indexy takých elementov item, ktoré obsahujú dáta zhodné s aktuálne porovnávaným poľom, získaným z poľa GET. V podstate máme teraz v poly indexov také elementy item, ktoré obsahujú dáta požadované v predchádzajúcich iteráciách a súčasne v poslednej iterácii. Takýmto postupom prejdeme všetky polia z poľa GET a z elementov item vyfiltrujeme tie, ktoré nesú informácie, o ktoré má používateľ záujem (vybral si ich vo filtroch pri tvorbe widgetu).

Dátový zdroj

Cieľom je vytvoriť riešenie, ktoré v našom súbore, z ktorého aktuálne získavame údaje, umožňuje definovať viacero kategórií a samozrejme tie potom zohľadniť pri úvodnej obrazovke, kde sa používateľovi zobrazuje zoznam oblastí (kategórií), ktoré si môže zvoliť.
Tieto oblasti musia odrážať aktuálne kategórie v našom zdroji, aby mal tak používateľ k dispozícii vždy najnovšie údaje. Tzn., že vytvoríme generickú triedu, ktorá bude parsovať údaje z nášho dátového zdroja a zobrazovať ich tak, aby sa rovnaká možnosť nezobrazovala viac krát a aby si mohol používateľ zvoliť viacero oblasti záujmu. V prípade, že sa kategórie zmenia (názov alebo počet), naše riešenie zobrazí aktuálne údaje bez ďalších zásahov do kódu

a prípadných úprav. Pre identifikovanie názvu kategórie a jej hodnoty použijeme oddeľovač, ktorým bude prvý výskyt znaku „:", kedy reťazec do tohto oddeľovacieho znaku bude predstavovať názov kategórie a reťazec od miesta výskytu bude predstavovať jej hodnotu. Napríklad zo zobrazeného kódu nižšie definujeme kategóriu „Kraj" a jej hodnota bude „Bratislava":

```
<category><![CDATA[Kraj: Bratislava]]></category>
```

Je viacero možných riešení vylepšujúcich aktuálny spôsob. Sú to napríklad uloženie zdrojových dát v podobe xml súborov na našom serveri, ukladanie vyparsovaných dát z rss súborov do databázy alebo ich ukladanie do cache na serveri.

Kategorizácia textu

Keďže logika prekladu textu je už vyhotovená, je potrebné už len zakomponovať do existujúceho riešenia preklad textu tak, aby bolo možné načítať slovenský text, tento poslať Google Translate API na preklad a odpoveď vo forme JSON rozparsovať a vo forme voľného textu následne poslať ako požiadavku na kategorizáciu textu. Vrátenú kategóriu je potrebné ešte namapovať na slovenský ekvivalent, pretože vrátená kategória je v anglickom jazyku. Takto získané kategórie ďalej zobrazíme vo forme checkboxov, aby si používateľ mohol vybrať z jednotlivých kategórií filter, podľa ktorého sa mu udalosti prefiltrujú.

Databáza

Ukladanie dát do xml súboru by nám postačovalo, avšak ak uvažujeme, že budeme mať veľké množstvo rss dátových zdrojov rôznych druhov a v nich veľa dát, tak bude výhodnejšie použiť databázu. Databáza je väčšinou rýchlejšia ako načítavanie z xml súboru. Takisto ak bude požadovať prístup k dátam množstvo používateľov, bude rýchlejšie posielať dopyty na databázu ako parsovať xml súbory. Taktiež je otázne či by bolo viditeľné zlepšenie parsovania xml súboru priamo na serveri oproti jeho stiahnutiu z externého zdroja a jeho následného spracovania.

Cacheovanie

Ďalší zo spôsobov je, že budeme spracovávať a parsovať xml súbory a získané dáta následne ukadať do cache. Pri požiadavkách používateľov o dáta do widgetov by sa tieto dáta už nemuseli získavať opakovaným spracovávaním rss súborov, ale by sa už získali z dát uložených v cache. Úlohou by bolo v určitých intervaloch updateovať dáta uložené v cache,

pretože samotné rss zdroje sa menia a používateľ chce aktuálne dáta. Na pravidelný update týchto dát by sme museli napísať nejaký skript alebo využiť inú podobnú možnosť.

XML súbor

Ďalší prístup k danému problému je, že si vytvoríme xml súbor zo všetkých dát, ktoré nám prídu a budeme to zakaždým parsovať a posielať klientovi dáta. Avšak to nie je veľmi dobrý spôsob najmä v prípade, že tých dát bude veľa.

Ukladanie kategórií do číselníka

Keďže rss súbory budú môcť obsahovať rôzne typy kategórií podľa ktorých sa budú filtrovať údaje, bude vhodné mať ich uložené v číselníkoch. Tie je možné mať uložené v databáze alebo aj v xml súbore. Použitie xml súboru pripadá do úvahy aj preto, že by neobsahoval veľké množstvo dát a mal by jednoduchú štruktúru, čiže by bolo jednoduché ho spracovávať. Výhodou by bolo taktiež oddelenie štruktúry kategórií od samotných dát. Štruktúra by sa nemala meniť až tak často ako samotné dáta, preto použitie číselníkov by malo byť vhodné.

Možnosti cache-ovania v nette

Samotný framework Nette, ktorý používame na implementáciu nám ponúka možnosti na cacheovanie rôznych dát. Je prístupná knižnica Cache.php, ktorá slúži na ukladanie dát do cache. Jej výhoda je, že umožňuje ukladanie akýchkoľvek štruktúr na ukladanie nielen textových reťazcov. Výhodou je rýchla a jednoduchá práca s cacheovanými dátami.
Uvediem jednoduchý príklad použitia tejto knižnice:

```
// získnie inštancie cache
$storage = new FileStorage('tmp');
$cache = new Cache($storage); // alebo $cache = Environment::getCache()

// zápis do cache
$cache['data'] = $myData;

// čítanie z cache
$cachedData = $cache['data'];

// vymazávanie z cache
unset($cache['data']);
// alebo
$cache['data'] = NULL;

// overenie, či je položka v cache
if (isset($cache['data'])) ...
// alebo
```

```
$cachedData = $cache['data'];
if ($cachedData !== NULL) ...
```

Pri ukladaní položiek je možné špecifikovať ďalšie parametre a podmienky pre invalidáciu. Pre ich špecifikáciu je potrebné použiť funkciu `$cache->save($key, $data, $options)`, kde parameter $options je pole, ktoré môže obsahovať tieto kľúče:

```
expire => (int) čas, kedy obsah vyexpiruje
sliding => (bool) má sa expirácia predlžovať?
files => (array) zoznam súborov, na ktorých cache závisí
items => (array) zoznam kľúčov v cache, na ktorých táto položka závisí
tags => (array) zoznam vlastných tagov
priority => (int) priorita
consts => (array) zoznam konštant, na ktorých cache závisí
```

Implementácia

Filtrovanie dát z RSS

Algoritmus popísaný v kapitole Návrh je implementovaný v súbore functionWidget.php a to konkrétne vo funkciách doFirstIteration() (spúšťa sa len raz a to pri prvej iterácii resp. ak je pole indexov obsahujúcich odkazy na elementy item prázdne)a doNextIteration(). Náš widget volá súbor widget.php, ktorý je prepojený s functionWidget.php.

Funkcia doFirstIteration():

```
function doFirstIteration($index, $keys, $doc, $category)
{
	$itemsIndexes = array();
	$iterator = 0;
	foreach ($doc->getElementsByTagName('item') as $node) //zoberiem si elementy item
	{
		if($node->getElementsByTagName('category')->length != 0) //pre kazdy item zistim ci obsahuje element category
		{
			//porovnat vsetky category v elemente item
			//s hodnotami v poli category
			foreach ($node->getElementsByTagName('category') as $catElement) //prejdem vsetky categories v item elemente
			{
				$typeOfElement = substr($catElement->nodeValue, 0, strpos($catElement->nodeValue, ":"));
				if(strcmp(normalizeString($typeOfElement), (string)$keys[$index]) == 0)
				{
					$contentOfElement = substr($catElement->nodeValue, (strpos($catElement->nodeValue, ":")+2));
					foreach ($category as $catHelp)
					{
						$pos1 = strpos($catHelp, "'");
						$pos2 = strrpos($catHelp, "'");
						$catHelp = substr($catHelp, $pos1 + 1, $pos2 - 1);
```

```
      if(strcmp(normalizeString(utf8_encode((string)$catHelp)),
normalizeString((string)$contentOfElement)) == 0)
                                    {
                                          array_push($itemsIndexes,
$iterator);
                                    }
                              }
                        }
                  }
            }
            $iterator++;
      }
      return $itemsIndexes;
}
```

Dátový zdroj

Ako sme v návrhu uviedli, definovali sme si novú triedu RSSParser, ktorá bude slúžiť na získavanie aktuálnych údajov. Táto trieda obsahuje 2 premenné : *$categoryFromRss* a *$categoryValuesFromRss*. *$CategoryFromRss* predstavuje pole kategórií, získaných z dátového zdroja (napr. Kraj, Deň v týždni) a *$CategoryValuesFromRss* predstavuje pole získaných hodnôt pre jednotlivé kategórie (napr. pre Kraj to môže byť Bratislava, Košice, Stará Ľubovňa ...). Ďalej sa tu okrem metód pre získavanie a defnovanie spomínaných premenných (set a get metódy) a konštruktora nachádza metóda *getCategory*, ktorá sa stará o získanie potrebných dát a ich odovzdanie (ako návratovú hodnotu). Návratovou hodnotou je pole, ktoré na prvom indexe obsahuje pole kateórií a na druhom pole získaných hodnôt pre konkrétne kategórie. Ako parameter dostáva dátový zdroj, z ktorého dáta má získať.

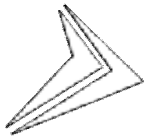
Ukážka hlavnej metódy:

```
function getCategory($rssSource)
{
   $doc = new DOMDocument();
   $doc->load(trim($rssSource));

   $arrCategories = array();
   $specificCategory = array();
   $counter=0;

   foreach ($doc->getElementsByTagName('item') as $node)
   {

      foreach ($node->getElementsByTagName('category') as $category)
      {
        $pole    =    substr($category->nodeValue,    0,    strpos($category-
>nodeValue, ":"));
        $obsah    =       substr($category->nodeValue,    (strpos($category-
>nodeValue, ":")+2));

        if (!in_array($pole, $arrCategories))
        {
```

```
            array_push($arrCategories,$pole);
            $specificCategory[array_search($pole,     $arrCategories)]     =
array();
            array_push($specificCategory[array_search($pole,
$arrCategories)],$obsah);            }
         else
        {
            if        (!in_array($obsah,$specificCategory[array_search($pole,
$arrCategories)]))
            {
                array_push($specificCategory[array_search($pole,
$arrCategories)],$obsah);
            }
        }

       }
    }

    $categoryArrayAndValues=array();
    $categoryArrayAndValues[0]=$arrCategories;
    $categoryArrayAndValues[1]=$specificCategory;

    return $categoryArrayAndValues;
}
```

Ukážka funkcionality:

```
  <author>mail@mail.sk (Organizátor)</author>
  <category><![CDATA[Organizátor: Organizátor]]></category>
  <category><![CDATA[Kraj: Bratislava]]></category>
  <category><![CDATA[Tag: Stiv-ov tag]]></category>
  <dc:creator>Organizátor</dc:creator>
</item>
<item>
  <title>Duchovné cvičenia Komunity Blahoslavenstiev</title>
  <description><![CDATA[DC o viere a vydávaní svedectva viery
  <pubDate>Sun, 31 Oct 2010 18:24:41 +0100</pubDate>
  <link>http://vyveska.kaweb.sk/index/details/id/73</link>
  <guid>http://vyveska.kaweb.sk/index/details/id/73</guid>
  <author>blahoslavenstva@alconet.sk (Komunita blahoslavenstie
  <category><![CDATA[Organizátor: Komunita blahoslavenstiev]]>
  <category><![CDATA[Kraj: Bratislava]]></category>
  <category><![CDATA[Kategória: Duchovné]]></category>
  <dc:creator>Komunita blahoslavenstiev</dc:creator>
</item>
<item>
  <title>Duchovné cvičenia Komunity Blahoslavenstiev</title>
  <description><![CDATA[Seminár o sebaprijatí a vzťahu k sebe
  <pubDate>Sun, 31 Oct 2010 18:24:38 +0100</pubDate>
```



Organizátor: Organizátor, Komunita blahoslavenstiev

Kraj: Bratislava, Trnava, Presov

Tag: Stiv-ov tag, Dobre, Zle

Kategória: Duchovné

Kategorizácia textu

Implementácia tejto časti nebola vypracovaná podľa požiadaviek do konca tohto šprintu z dôvodu, že sa jej dokončenie nestihlo. V predbežnej pracovnej verzii tejto kategorizácie textu je možné zadať text v slovenskom jazyku, odoslať požiadavku na Google Translate API a toto API nám vo forme JSON vráti odpoveď s preloženým textom v anglickom jazyku. Pri odoslaní požiadavky na Google Translate API bolo potrebné vykonať úpravy nad týmto textom a to z dôvodu, že anglický jazyk nemá diakritiku a pri odoslaní požiadavky s diakritikou sa slová, ktoré túto diakritiku obsahovali zle preložili, čo spôsobovalo obrovskú

nečitateľnosť textu, pretože diakritika sa nachádzala takmer v každom druhom slove. Okrem nečitateľnosti nastával aj problém so zlou kategorizáciou textu.

Volanie služby bolo vykonané pomocou php funkcie `file_get_contents($url)`, kde premenná $url obsahovala vyskladané volanie na Google Translate API v podobe http://ajax.googleapis.com/ajax/services/language/translate?v=1.0&$params. Za otáznikom sa nachádzajú parametre tohto volania, pričom parameter *v* označuje verziu použitého protokolu, tu verzia 1.0 a zvyšné parametre boli špecifikované v podobe premennej $param, ktorej obsah je "q={$txt}&langpair={$srcLang}|{$destLang}", kde za parametrom *q* nasleduje textový reťazec, ktorý chceme preložiť, a parameter *langpair* označuje dvojicu jazykov, pričom prvý z dvojice je nepovinný jazyk a je ním zdrojový jazyk. Druhý z dvojice je už povinný a je pred neho potrebné dať i oddeľovač jazykov v podobe |. V našom prípade je touto dvojicou *'sk'|'en'*. V nasledujúcom kóde je zobrazená funkcia *translate($txt, $destLang, $srcLang)*, ktorá je určená práve na preloženie textu z jedného jazyka do druhého po úspešnom preklade(teda ak návratový status je rovný 200 OK):

```
function translate($txt, $destLang, $srcLang)
{
    $txt = normalizeString($txt);
    $txt = urlencode( $txt );
    $destLang = urlencode( $destLang );
    $srcLang = urlencode( $srcLang );
    $params = "q={$txt}&langpair={$srcLang}|{$destLang}";
    $url=
"http://ajax.googleapis.com/ajax/services/language/translate?v=1.0&$params
";
    $translate = file_get_contents($url);
    $json = json_decode( $translate, true );
    if( $json['responseStatus'] != '200' )
        return false;
    else
        return $json['responseData']['translatedText'];
}
```

Databázový návrh

Úvodom upozorňujeme, že daná funkcionalita ešte nie je zavedená v hlavnej vývojovej vetve, je to zatiaľ experimentálna funkcionalita, ktorá sa vyvíja.

Najprv bolo nutné preskúmať možnosti komunikácie s databázou, ktoré ponúka Nette framework. Existujú dve cesty, ktorými sme sa mohli vydať: komunikácia pomocou ORM Doctrine alebo komunikácia pomocou databázovej vrstvy Dibi. Rozhodli sme sa použiť knižnicu Dibi, pretože je ľahšia na pochopenie, Doctrine je oproti nej príliš zložitá, aj keď ponúka širšie možnosti. Rozhodli sme sa použiť databázu MySQL, keďže je dobre zdokumentovaná, jej kvalita je potvrdená používaním v množstve webových riešení a členovia tímu s ňou už majú predošlé skúsenosti. Databázu (resp. SQL skript na vytvorenie

tabuliek) sme vytvorili pomocou prostredia MySQL Workbench, ktoré model priamo synchronizuje s databázovým serverom.

Nasleduje ukážka práce s knižnicou Dibi pri jej používaní vo frameworku Nette.

Nastavenia pre pripojenie do databázy (typ ovládača – mysql, host, meno, heslo a pod.) sa pridajú do súboru config.ini. Z neho sa načítajú v súbore bootstrap.php príkazom

```
dibi::connect(NEnvironment::getConfig('database'));
```

kde ‘database’ je meno premennej v súbore config.ini.

Ďalej sa zadefinuje nová trieda v modeloch (DBManager.php), ktorá vykonáva nad databázou jednoduché dopyty. Funkcia na nájdenie používateľa podľa jeho ID vyzerá nasledovne:

```
public function findUser($id)
{
    return dibi::query('SELECT * FROM [user] WHERE [id]=%i LIMIT 1', $id)
                ->setRowClass('User')
                ->fetch();
}
```

Pre každú tabuľku sa zadefinuje trieda v modelovej časti, ktorá dedí od DibiRow a ktorá reprezentuje entitu z tejto tabuľky. Vytvorená trieda User napríklad obsahuje aj funkciu na uloženie usera do tabuľky (databázy):

```
public function save()
{
      return  dibi::query('UPDATE  [user]  SET',  (array)  $this,  'WHERE
      [id]=%i', $this->id);
}
```

Teraz si stačí už len vytvoriť presenter, ktorý bude mať okrem iného aj funkciu, ktorá vytvorí DBManager a ten môže napríklad vytvoriť, alebo zmazať nejakého Usera – toto sa môže volať napríklad vo formulári.

Funkcia na vrátenie DBManagera:

```
public function getModel()
{
        if(!isset($this->dbManager))
            $this->dbManager = new DBManager;

        return $this->dbManager;
```

Testovanie

Testovanie výslednej podoby widgetu prebehlo bez problémov a widget je aplikovateľný na stránku. Testovanie kategorizácie pomocou REST služby Alchemy API neprebehlo, pretože táto nebola doimplementovaná do riešenia. Takisto neprebehlo testovanie časti úloh ohľadom rozšírenia štýlov pre kustomizáciu widgetu z dôvodu nedoriešenia tejto úlohy do času odovzdania dokumentácie.

Tabuľka č.1: Akceptačný test pre zobrazenie údajov o udalosti vo widgete

ID	1	**Názov**	Zobrazenie údajov o udalosti vo widgete		
Úroveň splnenia testu			Musí – ~~Mal by~~ – ~~Mohol by~~	**Autor**	Jozef Macho
Rozhranie		Používateľská stránka alebo kustomizácia widgetu			
Účel		Overenie správnej funkčnosti zobrazovania údajov o udalosti.			
Vstupné podmienky		Aspoň jedna udalosť v dátovom zdroji.			
Výstupné podmienky		Údaje o udalosti zobrazené vo widgete.			
Krok	**Akcia**		**Očakávaná reakcia**		**Skutočná reakcia**
1	Otvorenie stránky s widgetom alebo kustomizácia widgetu		Widget zobrazuje udalosti: názov, linka, dátum, miesto.		Rovnaká ako očakávaná.

Tabuľka č.2: Akceptačný test pre filtrovanie dát z RSS.

ID	2	**Názov**	Filtrovanie dát z RSS		
Úroveň splnenia testu			Musí – ~~Mal by~~ – ~~Mohol by~~	**Autor**	Jozef Macho
Rozhranie		Stránka widgetizéru			
Účel		Overenie správnej funkčnosti filtrovania dát podľa označených kategórií			
Vstupné podmienky		Získané kategórie z dátového zdroja vo widgetizéri			
Výstupné podmienky		Zobrazené záznamy podľa kategórií v náhľade widgete			
Krok	**Akcia**		**Očakávaná reakcia**		**Skutočná reakcia**
1	Označenie kategórií na filtrovanie pomocou zakliknutia CheckBoxu		Zobrazenie záznamov s danou kategóriou podľa výberu v CheckBoxoch.		Rovnaká ako očakávaná.