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

Prispôsobiteľný Widget
(dokumentácia k 6. - 9. šprintu)

Tím č.10 : the 6_p@ck

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

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

6. šprint (22.2.2011 - 7.3.2011)

Analýza

Pred začatím šiesteho šprintu sme pozorne prešli doterajší výsledok tímovej práce a zhodnotili sme, že je potrebné odstrániť viacero závažných nedostatkov, ktoré bolo potrebné vyriešiť skôr než sa budeme môcť pustiť do ďalšieho vývoja a rozširovania funkcionality.

Medzi tieto nedostatky patrilo okrem iného preklopenie všetkého kódu aplikácie do použitého frameworku (Nette), nakoľko niekoľko súborov bolo naprogramovaných priamo pomocou PHP programovacieho jazyka a nevyužívali možnosti frameworku a zároveň tým vystavovali celú aplikáciu možným konfliktom a aj potenciálnemu nebezpečenstvu v rámci bezpečnosti aplikácie ako celku.

Ďalej je potrebné zlepšiť i samotnú bezpečnosť vytvoreného kódu. V rámci štruktúry aplikácie sa nachádza niekoľko priamo písaných SQL dopytov na databázu, pričom bolo definované, že všetky dopyty na databázu budú vykonané prostredníctvom knižnice DIBI.

Takisto boli objavené nedokonalosti v spôsobe vytvárania unikátneho reťazca pri potvrdzovaní registrácie, ktoré umožňovali "ukradnutie" konta používateľa, resp. umožňovali zneužiť emailové konto iného používateľa na registráciu.

Jednou zo základných úloh v šiestom šprinte bolo aktualizovanie zdrojov uložených v databáze. Je nevyhnutné poskytovať widgetom aktuálne dáta, ktoré sa nachádzajú v príslušných rss zdrojoch. V rámci procesu aktualizácie bolo taktiež relatívne dôležité odstrániť nepoužívané záznamy, aby sa zbytočne nekopili v tabuľke záznamov. Naša databáza potom slúži ako akási medzipamäť (cache) pre všetky spracúvané zdroje. Výhodou môže byť, že informácie ponúkame aj v prípade, že zdrojová stránka je dočasne nedostupná. Okrem toho bolo nutné zabezpečiť pravidelnosť tejto aktualizácie.

V šiestom šprinte sme sa rozhodli umožniť používateľom nášho widgetu pridávať si vlastné RSS zdroje a následne si z nich vygenerovať widget. Externé zdroje, ako sme ich pracovne nazvali, je možné aj navzájom kombinovať. V tom prípade používateľ uvidí vo svojom widgete pomiešané udalosti (správy, informácie ...) zo všetkých zadaných RSS utriedené podľa dátumu ich aktuálnosti. Externé RSS zdroje je možné kombinovať aj s naším preddefinovaným RSS zdrojom.

Problémová oblasť v rámci pridávania externých RSS zdrojov bola voľba kategórií, ktoré sa majú vo widgete zobraziť. My nevieme a ani nie je našim cieľom vedieť rozpoznať resp. vedieť identifikovať aké kategórie sa môžu nachádzať v cudzích RSS zdrojoch. Preto v prípade, kedy používateľ zadá svoj vlastný RSS zdroj sa UC filtrovania kategórií nevolá. Definovanie widgetu pokračuje voľbou štýlu widgetu. Kategórie sa volia iba v prípade ak používateľ použije súčasne náš zdroj spolu s jeho zadanými. V takom prípade sa filtrovanie kategórií vzťahu len na naše RSS.

V šiestom šprinte sme sa rozhodli umožniť používateľom nášho widgetu pridávať si vlastné RSS zdroje a následne si z nich vygenerovať widget. Externé zdroje, ako sme ich pracovne nazvali, je možné aj navzájom kombinovať. V tom prípade používateľ uvidí vo svojom widgete pomiešané udalosti (správy, informácie ...) zo všetkých zadaných RSS utriedené podľa dátumu ich aktuálnosti. Externé RSS zdroje je možné kombinovať aj s naším preddefinovaným RSS zdrojom.

Problémová oblasť v rámci pridávania externých RSS zdrojov bola voľba kategórií, ktoré sa majú vo widgete zobraziť. My nevieme a ani nie je našim cieľom vedieť rozpoznať resp. vedieť identifikovať aké kategórie sa môžu nachádzať v cudzích RSS zdrojoch. Preto v prípade, kedy používateľ zadá svoj vlastný RSS zdroj sa UC filtrovania kategórií nevolá. Definovanie widgetu pokračuje voľbou štýlu widgetu. Kategórie sa volia iba v prípade ak používateľ použije súčasne náš zdroj spolu s jeho zadanými. V takom prípade sa filtrovanie kategórií vzťahu len na naše RSS.

Návrh

Podľa dohody uzavretej na stretnutí tímu bolo definované, že je potrebné preklopiť PHP súbor zabezpečujúci potvrdzovanie registrácie používateľa priamo do použitého frameworku. Na to je nevyhnutné vytvoriť nový Model, ktorý pokryje funkcionálne požiadavky z pôvodného PHP súboru. Taktiež je potrebné vytvoriť príslušný Presenter a k nemu samozrejme aj zodpovedajúci Template. Ideálne by mali mať v názve kľúčoví slovo "Confirmer", aby bolo jasné, že ide o funkcionalitu potvrdenia registrácie.

Ďalej je potrebné zvýšiť bezpečnosť aplikácie pri prístupe do databázy. SQL dopyty na databázu, ktoré boli písané priamo v SQL boli nahradené príkazmi z knižnice DIBI. Zmeny sú na viacerých miestach databázy a v kóde rôznych používateľov.

Takisto problém s možnosťou zneužitia cudzieho mailu pri registrácii je potrebné eliminovať generovaním náhodného reťazca veľkých a malých písmen a čísiel. Ideálne je potrebné vytvoriť nový Model s požadovanou funkcionalitou. V databáze je pri používateľovi potrebné pridať nový riadok.

Zdroje sú uložené v tabuľke ‘rsssource‘. Aktualizovať zdroje znamená postupne ich načítať z ich príslušnej url, preiterovať ich jednotlivé záznamy a v prípade, že sa záznam ešte nenachádza v databáze, treba ho tam vložiť.

Mazanie nepoužívaných zdrojov sa vyrieši pre začiatok tak, že pred aktualizáciou sa zmaže celá tabuľka ‘rssitem’ (aj naviazaná tabuľka ‘categoryitembind’). Toto riešenie nie je veľmi šťastné, ale ako úvodný prototyp postačuje. Jeho nevýhoda spočíva v tom, že po určitú dobu budú dáta nedostupné a widgety budú dostávať neúplné dáta. (riešenie v ďalšom šprinte).

Pravidelnosť aktualizácie je zabezpečená tabuľkou ‘updatelog’, do ktorej sa zaloguje každá aktualizácia dát. V prípade, že je záznam mladší, ako nastavená hodnota (v app/config.ini), tak sa aktualizácia nevykoná. V opačnom prípade sa vykoná. Kontrola sa vykonáva vždy pri otvorení hlavnej stránky widgetizéru.

Pre definovanie widgetu vznikne nová obrazovka, ktorá sa zobrazí pri spustení vytvárania widgetu. Na tejto obrazovke si používateľ bude môcť vybrať buď náš zdroj, pridať vlastný RSS zdroj alebo kombináciu oboch.

Po zadaní nových zdrojov používateľom ich musíme spracovať. Zdroje musia byť uložené do databázy. Ukladáme len základné informácie o RSS zdroji, ktoré nesú elementy definované v špecifikácii RSS 2.0. Každý používateľ môže zadávať rôzne RSS zdroje nezávisle od iného používateľa, preto je dôležité zabrániť duplicite dát v databáze. Tá by mohla nastať v prípade ak dvaja používatelia zadajú ten istý RSS zdroj. Tejto situácii prechádzame kontrolou existujúcich RSS zdrojov v našom dátovom úložisku pri pridávaní nového resp. vlastného RSS zdroja používateľom. Používateľ nebude nijako informovaný o tom, že sa pokúša zadať duplicitný zdroj. Táto informácia preňho nie je dôležitá, pretože na ním požadovanú funkcionalitu, výsledok to nemá dopad.

Pri vytváraní widgetu, ktorý bude obsahovať dáta z externého RSS zdroja je nutné tento zdroj namapovať na vznikajúci widget. Preto v databáze vznikne tabuľka „*sourcewidget*“, ktorá prepája widgety so zdrojmi, z ktorých čerpajú dáta. Tak isto pri mazaní widgetov je nutné odstrániť tieto referencie na RSS zdroje.

Pri editácii widgetu a úprave zdrojov musíme odstrániť referencie na staré RSS zdroje a pridať referencie na nové RSS zdroje.

Implementácia

Všetok kód z pôvodného súboru napísaného v čistom PHP kóde bol presunutý do frameworku. Ide konkrétne o model: RegistrationConfirmer.php, o presenter: ConfirmationPresenter.php a template: Confirmation -> default.phtml.

Časť implementácie nového modelu uvedieme aj v nasledujúcej ukážke:

```
<?php
class RegistrationConfirmer
{
	public static function confirmRegistration()
	{
		//$user = urldecode($_REQUEST['user']);
		$control = urldecode($_REQUEST['control']);

		////////////////////////////////////
		$res = dibi::query('SELECT * FROM [user] WHERE [control] =
%s',$control,' AND [confirmed] = %i', 0);
		////////////////////////////////////
```

```
            $numrow = $res->rowCount();

            if($numrow != 1)
            {
                  return "Neplatný odkaz - používateľ neexistuje alebo už
bola registrácia potvrdená!";
            }
            else
            {
                  $value = $res->fetchSingle();
                  {
                        $confirmed = 1;

                        dibi::query('UPDATE [user] SET [confirmed] = %s',
1, 'WHERE [control] = %s', $control,' AND [confirmed] = %i', 0);

                        $user = dibi::select('username')->from('user')-
>where('control=%s', $control)->fetch();

                        return "Registrácia používateľa
'".$user['username']."' úspešne dokončená!";
                  }
            }
      }
}
```

Na generovanie náhodného reťazca zloženého z čísiel a malých a veľkých písmen abecedy bol vytvorený model RandomGenerator.php, ktorý generuje reťazce dĺžky 64 znakov, čo by malo byť viac než dostatočné na zabránenie zneužitia cudzích emailov. Zdrojový kód modelu je uvedený nižšie:

```
<?php

class RandomGenerator
{
      public $ranStr = '';

      public static function randomString()
      {
            $rs = '';
            for ($i=0; $i<64; $i++)
            {
            $typeOfSymbol=rand(1,111)%3;
                  if($typeOfSymbol == 0)
            {
            //adds numeric symbol between 0 - 9
            $rs .= chr(rand(48,57));
            }
            if($typeOfSymbol == 1)
            {
            //adds small alpha symbol between a - z
            $rs .= chr(rand(97,122));
            }
            if($typeOfSymbol == 2)
            {
            //adds capital alpha symbol between A - Z
            $rs .= chr(rand(65,90));
            }
            }
```

```
            return $rs;
      }

      public static function getRandomString()
      {
            $rs = RandomGenerator::randomString();
            $res = dibi::query('SELECT control FROM [user] WHERE [control]
= %s',$rs);
            while($res->rowCount() > 0)
            {
                  $rs = RandomGenerator::randomString();
                  $res = dibi::query('SELECT control FROM [user] WHERE
[control] = %s',$rs);
            }
            return $rs;
      }
}
```

Ako vidno, tak je zabezpečené aj to, aby sa žiaden reťazec neopakoval viac než raz. A to už priamo pri jeho generovaní. Dĺžka 64 znakov zabezpečuje, že nikto nedokáže reťazec len tak uhádnuť a zároveň vzhľadom na malý počet používateľov je nepravdepodobné, že by aj pri útoku hrubou silou dokázal program nájsť práve reťazec, ktorý bol použitý. A aj tento fakt by mu nepomohol, nakoľko by nepoznal prihlasovací login používateľa, ktorý sa už v reťazci nevyskytuje. Tým je zabezpečená a zlepšená bezpečnosť pri registrácii.

Pre prácu so zdrojmi v databáze sa vytvorila trieda RssSourceManager – oddedená je od triedy DBManager. Na načítanie všetkých zdrojov z databázy sa zavolá funkcia getRssSourcesFromDb() v triede RssSourceManager. Funkcia je implementovaná nasledovne:

```
public function getRssSourcesFromDb()
    {
      $result =  dibi::query('SELECT [url] FROM [rsssource]')->fetchAll();
      $returnArray = array();
      foreach ($result as $n => $row) {
            $single = $row->toArray();
            $returnArray[$n] = $single['url'];

      }
      return $returnArray;
}
```

Pre každý zdroj, ktorý je vo vytvorenom poli, sa následne zavolá funkcia processRss() z triedy DataProcessor. Pred tým sa ešte zmaže tabuľka categoryitembind a rssitem pomocou sql príkazu truncate vykonaného cez Dibi.

Celá táto funkcionalita sa volá len v prípade, že je nutná aktualizácia – to sa zistí overením v tabuľke ‘updatelog’ – volaním funkcie isUpdateNeeded() v triede DataProcessor. Táto kontrolná metóda vyzerá nasledovne.

```
public function isUpdateNeeded($RssItemManager)
      {
            $lastUpdateTime = $RssItemManager->getLastUpdateTime();
            if ($lastUpdateTime == NULL)
            {
                  return TRUE;
```

```
            }
            $actualTime = time();
            $timeSinceLastUpdate = $actualTime - $lastUpdateTime;
            $updatePeriod = NEnvironment::getVariable('updatePeriod') * 60;

            if ($updatePeriod < $timeSinceLastUpdate)
                  return TRUE;
            else
                  return FALSE;
      }
```

Kontroluje premennú updatePeriod, ktorá sa nastavuje v konfiguračnom súbore aplikácie – config.ini. Po vykonanú aktualizácie sa táto akcia zaznamená v spomenutej logovacej tabuľke.

Ako je spomínané už v kapitole návrhu pre pridávanie nových externých RSS zdrojov vznikla nová obrazovka, ktorá je implementovaná v prezenteri „*WidgetPresenter.php*“. Obrazovka sa zobrazuje ako prvá pri vytváraní a editácii widgetu a jej šablóna sa volá „*addRssSource.phtml*“.

Zadaný zdroj sa ukladá do databázy hneď pri prechode na ďalšiu obrazovku. Uloženie RSS zdroja je implementované vo funkcii „*saveRssSourceIntoDB()*“ v triede „*RssItemManager.php*“.

```
      /**
       * Save Item from Rss to DB
       */
      public function saveRssItemIntoDB($title, $link, $date, $description,
$rssSourceId)
      {
            $queryResult = dibi::query ( 'INSERT INTO [rssitem] SET [title]
= %s, [link] = %s, [date] = %t, [description] = %s, [RSSSource_idRSSSource]
= %i', $title, $link, $date, $description, $rssSourceId);
            $idRSSItem = dibi::insertId ();
            return $idRSSItem;
      }
```

Mapovanie RSS zdroja na widget prebieha pri ukladaní widgetu do databázy. Konkrétne funkcionalitu mapovania implementuje funkcia „*mapSourcesToWidget()*“ v triede „*DBManager.php*“. Volaná je vo funkcii, ktorá ukladá widget „*saveWidgetIntoDbs()*“ v tej istej triede.

```
      /*
       * Function map widget to his rssSources
       */
      private function mapSourcesToWidget($rssSourcesId, $idWidget)
      {
            foreach($rssSourcesId as $sourceId)
            {
                  $insertQuery = dibi::query ( 'INSERT INTO [sourcewidget]
SET [widget_id]= %i,[rsssource_idRSSSource]= %i', $idWidget, $sourceId );
            }
      }
```

Implementácia zobrazovania widgetu v Nette, refaktoring

V našom projekte sme používali na získavanie údajov a ich následne zobrazovanie vo forme widgetu na klientskej stránke po vložení vygenerovaného zdrojového kódu triedy, ktoré neboli priamo súčasťou zvoleného frameworku. Z toho dôvodu sme používali dopyty na databázu, ktoré nevyužívali vrstvu dibii. Pre potrebu zjednotenia sme reimplementovali triedy a potrebné funkcie tak, aby sme všetko mali pod jednou strechou a pre získavanie dát z dibii používali výlučne vrstvu dibii, ktorá zjednodušuje zápis SQL príkazov a ponúka automatickú podporu konvencií (escapovanie / slashovanie, úvodzovkovanie identifikátorov a pod.). Taktiež sme metódy writeWidget()a writeWidgetPreview() slúžiace pre zobrazenie náhľadu vytváraného widgetu a pre zobrazenie vytvoreného widgetu po vložení vygenerovaného zdrojového kódu do klientskych stránok zjednotili do jednej pod názvom writeWidget() a nachádza sa v súbore *WidgetModel.php*. Táto zmena bola potrebná pretože metódy slúžili na vykonávanie rovnakých úloh, čo bol príklad duplicity v kóde, ktorú sme vyriešili.

Ako je spomínané už v kapitole návrhu pre pridávanie nových externých RSS zdrojov vznikla nová obrazovka, ktorá je implementovaná v prezenteri „*WidgetPresenter.php*“. Obrazovka sa zobrazuje ako prvá pri vytváraní a editácii widgetu a jej šablóna sa volá „*addRssSource.phtml*“.

Zadaný zdroj sa ukladá do databázy hneď pri prechode na ďalšiu obrazovku. Uloženie RSS zdroja je implementované vo funkcii „*saveRssSourceIntoDB()*“ v triede „*RssItemManager.php*“.

```
	/**
	 * Save Item from Rss to DB
	 */
	public function saveRssItemIntoDB($title, $link, $date, $description,
$rssSourceId)
	{
		$queryResult = dibi::query ( 'INSERT INTO [rssitem] SET [title]
= %s, [link] = %s, [date] = %t, [description] = %s, [RSSSource_idRSSSource]
= %i', $title, $link, $date, $description, $rssSourceId);
		$idRSSItem = dibi::insertId ();
		return $idRSSItem;
	}
```

Mapovanie RSS zdroja na widget prebieha pri ukladaní widgetu do databázy. Konkrétne funkcionalitu mapovania implementuje funkcia „*mapSourcesToWidget()*“ v triede „*DBManager.php*“. Volaná je vo funkcii, ktorá ukladá widget „*saveWidgetIntoDbs()*“ v tej istej triede.

```
	/*
	 * Function map widget to his rssSources
	 */
	private function mapSourcesToWidget($rssSourcesId, $idWidget)
	{
		foreach($rssSourcesId as $sourceId)
		{
			$insertQuery = dibi::query ( 'INSERT INTO [sourcewidget]
SET [widget_id]= %i,[rsssource_idRSSSource]= %i', $idWidget, $sourceId );
```

```
            }
        }
```

7. šprint (8.3.2011 - 21.3.2011)

Analýza

Nakoľko nám bola vytknutá veľká denormalizácia v databázovom modeli, je potrebné tento problém odstrániť. Je nutné odstrániť prebytočné tabuľky pre používateľov a zjednotiť dáta do jednej tabuľky.

Taktiež je potrebné vytvoriť SQL dopyt, ktorý dokáže z databázy získať zoznam všetkých článkov z viacerých zdrojov zoradené podľa dátumu publikácie. To doteraz nebolo zrealizované, hoci to má pre samotnú aplikáciu zásadný význam.

Ďalšou funkcionalitou bolo experimentálne zapracovanie AlchemyAPI, ktoré by slúžilo na rozširujúcu identifikáciu kategórií špeciálneho zdroja. Čo je však dôležité, jeho potenciál by sa dal teoreticky využiť aj pri kategorizácií všeobecných zdrojov. AlchemyAPI pracuje na princípe webovej služby, ktorej sa pošle určitým spôsobom sformovaná požiadavka obsahujúca neznámy text v anglickom jazyku a služba vráti kategóriu pre daný text.

Validácia externých zdrojov RSS

V predchádzajúcom šprinte sme dopracovali funkcionalitu, ktorá umožní používateľovi okrem základného zdroja používať aj ďalšie externé rss zdroje dát. Avšak je potrebné ošetriť túto funkcionalitu, aby v prípade zlých vstupov aplikácia nepadala. Preto sme sa rozhodli validovať zadané rss zdroje.

Úprava zobrazovania widgetu

Widget, ktorý si používateľ definuje sa mu zobrazí prvýkrát pri definovaní jeho vzhľadu a napokon po vložení vygenerovaného kódu do svojich stránok. Tento náhľad sme sa rozhodli spolu s definovaním určitých vlastností ako názov, výška, šírka ako aj samotné zobrazenie widgetu po jeho vložení na stránku klienta upraviť, aby nedovoľoval používateľovi definovať rozmery obsahujúce veľmi vysoké čísla a nekonečne dlhý názov. Takisto sme sa rozhodli zmeniť grafiku pre zobrazovania widgetu.

Mobilná webová aplikácia

Keďže rozvoj informačných technológií v oblasti mobilných zariadení ma zvyšujúcu tendenciu a mobilný telefón má drvivá väčšina neustále pri sebe, rozhodli sme sa používateľom poskytnúť zobrazovanie definovaného widgetu aj pre mobilné zariadenia. Takto majú v prípade mobilného zariadenia s pripojením na internet prístup k aktuálnym informáciam v oblastiach a štýle, ktoré si sami definovali z akéhokoľvek miesta a nie sú obmedzovaní na nutnosť prístupu k počítaču s internetom.

Návrh

Tabuľky "user" a "unconfirmed_user" bolo potrebné a žiadané zjednotiť. Z toho vyplynulo viacero problémov, ktoré však je možné vyriešiť. Je potrebné rozlíšiť používateľov, ktorí už majú potvrdenú registráciu od tých, čo ju potvrdenú nemajú. Pretože podľa toho je nutné vyhodnotiť ich pokus o prihlásenie. Je samozrejme jasné, že nemožno prihlásiť používateľa, ktorý nemá potvrdenú registráciu, pretože by tým toto potvrdenie stratilo význam. Je preto nutné pridať do tabuľky používateľov riadok, ktorý určuje túto vlastnosť pre každého používateľa.

S pomocou DIBI knižnice je tiež nutné napísať SQL dopyt do databázy, ktorý získa všetky články zo všetkých zdrojov, ktoré sú pre konkrétny widget špecifikované. Ide o prípad, kedy sú zdroje nie zo štandardného zdroja z vyveska.sk, ale z používateľom definovaných zdrojov.

Kategórie, ktoré sa pomocou AlchemyAPI priradia jednotlivým záznamom z rss zdroja, budú uložené rovnakým spôsobom, ako kategórie určené zo špeciálneho zdroja „chrustikovci“. Primárna kategória bude „Alchemy“ a sekundárna budú jednotlivé reťazce, ktoré vracia Alchemy služba. Toto nám zabezpečí aj kompatibilitu vo filtrácii a už hotový kód bude treba meniť len minimálne.

Validácia externých zdrojov RSS

Prípady v ktorých sa môže vyskytnúť problém a aplikácia sa by sa mohla správať nečakane:

- Nie je zvolený základný zdroj ani žiaden externý zdroj
- Ako externý zdroj je zadaný reťazec, ktorý nepredstavuje platnú URL adresu k xml súboru
- Zadaný externý zdroj nie je validný, teda nespĺňa špecifikáciu RSS 2.0

Tieto všetky prípady je potrebné ošetriť a v prípade, že nastanú je nutné používateľovi oznámiť aký problém nastal, aby nebol prekvapený v prípade, že by mu aplikácia nespracovala ním zadané hodnoty.

Pre validáciu rss sme sa rozhodli použiť externý validátor http://feedvalidator.org/, čím sa nám zjednodušila práca a nepotrebovali sme implementovať vlastný validátor, ktorý by kontroloval štruktúru súboru voči rss špecifikácii.

Úprava zobrazovania widgetu

Pri špecifikácii atribútov ako výška, šírka ako aj pri názve je potrebné definovať obmedzujúci počet vložených znakov tak, aby používateľ nemohol vložiť údaje, ktoré by mohli spôsobiť zlé zobrazovanie widgetu. Predpokladá sa, že názov widgetu bude krátky a výstižný, preto nie je žiadúce, aby bol na viacero riadkov a uberal miesto hlavnému obsahu – zobrazovaným informáciám. Taktiež možnosti upraviť si rozmery widgetu treba upraviť tak, aby boli v takých rozmedziach, že pri náhľade nepresahuje plochu pre ktorú je určený. V poslednom rade treba aktualizovať možnosti výberu veľkosti písma tak, aby boli v rozumných medziach (od 8px do 14px) aby veľkosť bola prijateľná vzhľadom k ploche určenej pre zobrazovanie záznamu. Takisto treba pridať tieňovanie do znázornenia widgetu.

Mobilná webová aplikácia

Mobilná webová aplikácia by mala byť prepojená z hlavným systémom, čo znamená, že používateľovi sa po tom, ako si definuje dátové zdroje pre widget a určí štýl zobrazovania, zobrazí kód reprezentujúci widget, ktorý sa po vložení na jeho stránku bude zobrazovať v tvare v akom bol definovaný, no zároveň sa mu zobrazí zdrojový kód určený pre mobilnú webovú verziu definovaného widgetu. Táto verzia widgetu je určená pre používateľov prehliadajúcich si daný widget prostredníctvom mobilných zariadení.

Implementácia

Tabuľka "unconfirmed_user" bola odstránená a do tabuľky "user" bol pridaný stĺpec CONFIRM, ktorý definuje či bol používateľ už potvrdený alebo nie. Následne sa podľa tejto hodnoty overuje jeho prihlásenie.

SLQ dopyt do databázy, ktorý zabezpečuje, že sa všetky články bez ohľadu na pôvod zobrazia podľa svojho dátumu publikovania:

```
$arrayOfContent = array(); //array with content of widget
```

```
            $rssAllSourceItemsID = array(); //tu ulozim vsetky items
vsetkych zdrojov, na kt. je widget namapovany

            $rssAllSourceItemsID = dibi::query('SELECT [idRSSItem] FROM
[rssitem] WHERE [RSSSource_idRSSSource] IN %l',$widgetSources,' ORDER BY
[date] DESC')->fetchAll();
```

Bolo nutné naprogramovať rozhodovaciu logiku, ktorá podľa toho, aký typ zdroja sa práve spracúva, určí spôsob parsovania a ukladania položiek do databázy (či určiť aj kategórie, či použiť AlchemyAPI, ak je zapnuté v config.ini a pod.).

Okrem toho sa naprogramovala vlastná trieda, ktorá slúži na analýzu zdrojov pomocou ALchemyAPI – AlchemyAnalyser.

Postup, ktorým AlchemyAnalyser identifikuje kategóriu je zobrazený na nasledujpcom diagrame:

Reťazec v slovenčine, ktorý pozostáva z názvu RSS položky a popisu položky sa preloží do angličtiny pomocou služby Google Translate. Návratové dáta sa spracujú pomocou JSON. Reťazec v angličtine sa pošle do AlchemyAPI pomocou cURL a výsledok (XML) sa spracuje pomocou DOM. Následne sa kategória preloží naším interným podmieňovacím mechanizmom do slovenčiny.

Vytvorenie cURL požiadavky na Alchemy API:

```
function getAlchemyResponseData($data)
      {
          $outputMode = "json";
          $txt = str_replace(" ", "%20", $data);
          $key = NEnvironment::getVariable('alchemyKey');
            $url                                                    =
"http://access.alchemyapi.com/calls/text/TextGetCategory?apikey=$key&text=$
txt&outputMode=$outputMode";
          $ch = curl_init();
          curl_setopt($ch, CURLOPT_URL, $url);
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
            curl_setopt($ch, CURLOPT_CONNECTTIMEOUT,0);

          $responseData = curl_exec($ch);
          $info = curl_getinfo($ch);
          curl_close($ch);
          if($info['http_code']!=200)
          {
              return "Error";
```

```
            }
            else{
                return $responseData;
            }
        }
```

Okrem očakávaných stavov bolo treba mierne ošetriť aj rôzne chybové návratové hodnoty – išlo predovšetkým o Alchemy, ktoré sa niekedy správa nestabilne. Napríklad napriek tomu, že sa od neho vyžadujú dáta v JSON formáte, vracia XML. S týmto v Alchemy analyzátore počítame.

Validácia externých zdrojov RSS

Pri validácii zadaných zdrojov bolo potrebné ošetriť prípady, ktoré sú popísané v časti návrh. Prvotnou myšlienkou bolo namapovať vlastný validátor na každé textové políčko. Avšak počas implementácie sa vyskytol problém v tom, že hodnota ktorá sa posielala z textového poľa do validačnej funkcie sa správala čudne. V prípade, že sme s ňou chceli zaobchádzať ako s reťazcom tvárila sa ako objekt a naopak v prípade, že sme s ňou pracovali ako s objektom, dostávali sme chybové oznámenie, že sa nejedná o objekt.

Chyba bola spôsobená frameworkom, preto sme tento spôsob museli zavrhnúť a vyriešiť to iným spôsobom. Riešenie spočíva v tom, že po odoslaní formulára sa pre každú hodnotu zadanú v textových poliach zavolá naša validačná funkcia. Ak vráti chybu pre niektorú hodnotu, používateľovi sa zobrazí oznámenie o probléme. V prípade, že sú všetky zdroje validné, používateľ je presmerovaný na ďalšiu stránku. Ak si zvolil aj základný zdroj, je presmerovaný na výber kategórií, ak nie je presmerovaný na voľbu štýlov. Na nasledujúcich riadkoch je časť kódu, ktorá sa zaoberá validáciou zadanej hodnoty z textového poľa po odoslaní formulára.

```
if($rssSourcesForm['source2']->getValue() != null)

{

      $rssSourcesUrl[] = $rssSourcesForm['source2']->getValue();

      $isValidField = $this->isValidFeed($rssSourcesForm['source2']-
            >getValue());

      if(!$isValidField){

            $errorMessage = 'Zdroj č.2 nie je validný';

            $this->redirect('Widget:addRssSources',  $this->isEdited,
                  $this->widgetId, $rssSourcesForm['defaultSource']-
                  >getValue(), $errorMessage , $rssSourcesForm['source2']-
                  >getValue(), $rssSourcesForm['source3']->getValue(),
```

```
                    $rssSourcesForm['source4']->getValue(),
                    $rssSourcesForm['source5']->getValue());

        }

}
```

Je potrebné si zvoliť aspoň jeden zdroj, preto ak nie je zvolený žiaden ani nie je zaškrtnutý checkbox pre základný zdroj, je používateľ oboznámený o povinnosti zadať aspoň jeden zdroj. Časť zdrojového kódu obstarávajúca túto funkcionalitu je zobrazená nižšie.

```
// show error message if no source was entered and checkbox for
defaultsource was not checked

if(empty($rssSourcesUrl) && !$rssSourcesForm['defaultSource']->getValue())

{

        $errorMessage = 'Zvoľte si aspoň jeden zdroj.';

        $this->redirect('Widget:addRssSources',  $this->isEdited,
              $this->widgetId, false, $errorMessage, null, null, null, null);

}
```

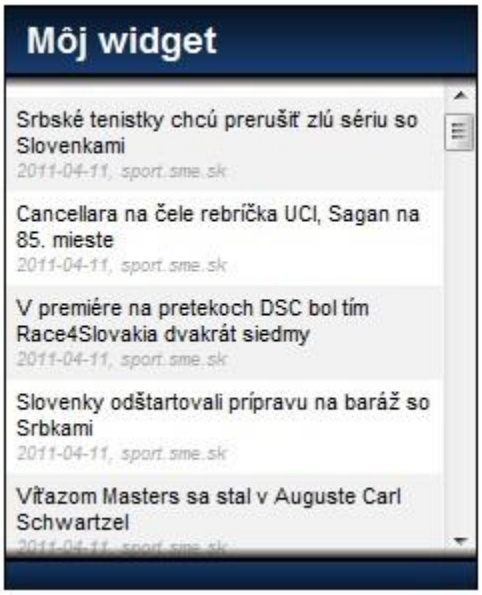
Ako bolo spomenuté v časti Návrh, na validáciu rss zdrojov používame externý validátor http://feedvalidator.org/. Jeho použitie je založené na odoslaní requestu na túto stránku s adresou rss, ktoré chceme validovať. Naspäť dostaneme response s stránkou, ktorá obsahuje informácie o validácii. Následne zistíme, či obsahuje informáciu o tom, že rss je validné. Nie je to ideálny spôsob, avšak bohužiaľ sme nenašli externú knižnicu do php, ktorá by nám pomohla validovať rss podľa špecifikácie 2.0. Existujú knižnice, ktoré slúžia na validáciu súborov voči xml špecifikácii, avšak nestretli sme sa so žiadnou, ktorá by to vedela spraviť pre rss.

Pri testovaní tejto funkcionality sme zistili, že viacero rss zdrojov, ktoré sme chceli používať nespĺňa práve špecifikáciu. Preto sa v budúcnosti možno budeme zaoberať implementáciou vlastného validátora, ktorý nebude taký prísny a umožní používateľom použiť aj zdroje, ktoré nespĺňajú úplne celú špecifikáciu.

Úprava zobrazovania widgetu

Pri implementácii sa zapracovali potrebné obmedzenia pre definovanie názvu, veľkosti písma a rozmerov priamo v HTML template (*createStyle.phtml*), kde sa jednotlivým poliam definoval maximálny počet vložených znakov. Plocha pod nadpisom sa vytieňovala a takisto

sa pridalo tieňovanie na konci a na začiatku plochy určenej pre zobrazovanie jednotlivých informácií widgetu. Titulok článku sa farebne odlíšil a zväčšil oproti údajom o dátume a zdroji daného záznamu. Aktuálny štýl je zobrazený na nasledujúcom obrázku.



Obr. 1. Ukážka štýlu widgetu

Mobilná webová aplikácia

Pri implementácii mobilnej webovej verzie sme sa pokúšali zabezpečiť dodržanie definovaných vlastností widgetu (definovaná farba pozadia, názov widgetu, počet zobrazovaných článkov...) takisto s funkcionalitou widgetu určeného pre klientske stránky (zobrazenie rovnakých článkov, rovnakých dát). Dôraz sme kládli na to, aby bola táto mobilná webová verzia použiteľná pre viacero mobilných zariadení a nie len určitým platformám. Umožnili sme zobrazenie widgetu pre rôzne veľkosti obrazoviek, čo umožňuje prezerať vygenerovaný widget na nie len na mobilných, ale aj iných zariadeniach s prehliadačom a prístupom na internet. Výhodou takto vytvorenej mobilnej webovej aplikácie je, že ak niečo zmeníme, používateľ si nemusí inštalovať aktualizácie ale má ihneď prístupnú aktuálnu verziu. Náhľad mobilnej webovej aplikácie zobrazujeme na nasledujúcom obrázku.

Obr. 2. Ukážka mobilnej webovej aplikácie pre widget

8. šprint (22.3.2011 - 4.4.2011)

Analýza

Úvahami o smerovaní celého projektu sa dospelo k názoru, že by bolo vhodné celý template aktuálnej stránky Widgetizéra pretvoriť do dizajnu búducej stránky vyveska.sk, na ktorú bol aj primárne tento projekt smerovaný.

Okrem tohto sa objavili aj problémy so zobrazeným widgetom, ktoré bolo nutné odstrániť.

Požiadavkou bolo, aby sa aktualizácia dát v databáze spúšťala asynchrónne, v pravidelných intervaloch a aby sa dáta počas aktualizácie nestrácali. To znamená, že sa nemôže mazať celá tabuľka, ale iba nepoužívané dáta s RSS záznamami. Tiež sa požadovalo, aby sa kategórie, ktoré boli určené pomocou Alchemy analyzátora, zobrazovali vo filtri. Odmazávanie neaktívnych dát sa deje počas aktualizácie dát v databáze.

Okrem toho, že sa zmení funkcionalita aktualizácie dát, bude vhodné, ak sa zavedie možnosť logovania dôležitých udalostí. Predovšetkým kvôli odhaľovaniu chýb a ladeniu.

Mobilná verzia pre platformu Android

V predchádzajúcom šprinte sme vytvorili mobilnú webovú aplikáciu pre náš widgetizér, ktorý umožňuje používateľom mobilných zariadení vzhliadnuť webovú mobilnú verziu vytvoreného widgetu. Táto služba umožňuje prispôsobenie widgetu mobilným zariadeniam. Napriek mnohým výhodám, takáto verzia obsahuje aj určité obmedzenia, ku ktorým patrí to, že používateľ musí do svojho mobilného prehliadača. Toto obmedzenie sme sa rozhodli eliminovať pre operačný systém Android (ďalej len OS Android), ktorý v poslednom období zasadol na post kráľa mobilných operačných systémov. Túto aplikáciu je potrebné prispôsobiť tak, aby sme nestratili možnosť aktualizácie widgetu a zároveň aby sme používateľom mobilných zariadení s OS Androidom ponúkli aplikáciu, ktorá im zobrazí widget spĺňajúci špecifiká definované pri jeho vytváraní na našej stránke.

Návrh

Podľa dodaných zdrojových kódov, ktoré dodal náš vedúci sme vytvorili upravili existujúci template stránky Widgetizéru, resp. do dodaného template vývesky sme vložili a prispôsobili logiku našej aplikácie.

Problém so zobrazovaním widgetu pri jeho náhľade aj pri konečnom zobrazení zrejme majú pôvod v chýbajúcich tagoch ukončujúcich alebozačínajúcich odseky na stránke. Je potrebné ich preveriť.

Aby sa zabezpečilo asynchrónne a „bezpečné“ aktualizovanie databázy, použije sa Cron. (plánovací program na unixových OS). Stratia sa tým síce možnosti prenositeľnosti aplikácie. Ale tie neboli nikdy vyžadované, preto to nepovažujeme sa vážnejší problém. Cronom sa bude spúšťať skript, ktorý bude aktualizovať dáta. Keďže sa nebude mazať pred aktualizáciou celá tabuľka, treba zabezpečiť mazanie neaktívnych položiek z tabuľky ‘rssitem’. Riešením je vždy po prebehnutí aktualizácie kontrolovať, či sa niektoré položky, ktoré sú v databáze, už viac nenachádzajú v k nim príslušnom rss zdroji. Ak je to tak, tieto položky sa zmažú (aj z tabuľky naviazania na kategóriu).

Filter je nutné zobrazovať aj pre tie zdroje, ktoré majú priradené nejaké Alchemy kategórie. Doteraz sa zobrazoval len pre špeciálny výveskový RSS zdroj.

Logovať sa bude do databázy – toto považujeme za najjednoduchšie riešenie, pričom tiež ponúka možnosti vyhľadávania v databáze.

Mobilná verzia pre platformu Android

Aplikáciu pre OS Anroid je vhodné vytvoriť ako natívno-webovú aplikáciu, pretože tým nestratíme kontrolu nad widgetom (pre prípadné zmeny vo widgete si nemusí používateľ aktualizovať aplikáciu) a zároveň bude mať používateľ aplikáciu, ktorú si nainštaluje a spustí kliknutím na jej ikonu. Tým pádom má rýchli prístup k widgetu, ktorý bol definovaný na našej webovej stránke. Je potrebné zabezpečiť, aby aplikácia bola univerzálna pre všetky vygenerované widgety a ak si ju niekto stiahne, tak má zaručené, že si na nej môže pozrieť akékoľvek widgety vygenerované prostredníctvom našich služieb. Musíme avšak nejako rozlišovať medzi jednotlivými aplikáciami, pretože aplikáciu nebudeme kompilovať pri vytváraní widgetu, ale bude k dispozícii už vopred. Na určenie verzie widgetu, ktorý sa má zobraziť použijeme jeho identifikátor (id v databáze), ktoré je unikátne, čím je zabezpečená jeho jednoznačnosť.

Implementácia

Nový dizajn stránky Widgetizéra bol úspešne aplikovaný a teraz pôsobí dojmom ako v budúcne aktívna stránka vyveska.sk. Do modrej a hnedej farby ladený dizajn pôsobí profesionálnejšie a je aj prepracovanejší ako predošlý dizajn, ktorý bol získaný bezplatne z internetu.

Oprava dizajnu pri jeho konečnom zobrazení ako aj odstránenie neznámej bielej čiary a náprava dizajnu widgetu pri vzhľade si vyžiadala revíziu celého kódu - hlavne DIV elementov v HTML kóde. Nakoniec bol identifikovaný prebytočný/chýbajúci element, ktorý spôsobil nesprávne zobrazovanie. Tiež sa našiel aj nepoužívaný element DIV, ktorý spôsoboval bielu čiaru narúšajúcu dizajn stránky. Všetky nedostatky v náhľade aj v konečnom vzhľade widgetu boli opravené.

Na podporu spúšťania aktualizačného skriptu cronom bolo nutné vytvoriť nový presenter, ktorý sa pomocou wgetu v crone bude otvárať. Vytvoril sa preto CronJobPresenter, ktorý volá spracovanie všetkých zdrojov v databáze – teda aktualizáciu. Adresa k presenteru je …/cronjob. Je tak možné vykonať aktualizáciu zdrojov aj explicitne – „na požiadanie“. Na nastavenie Cronu sa použil unixovský program Crontab, ktorým sa editujú dané užívateľské skripty.

Príkaz pre Cron:

```
0 */1 * * * wget -q -O /dev/null http://localhost/team10issi/kuzar/widgetizer/document_root/cronjob/
```

Mazanie položiek funguje tak, že pre daný zdroj sa vytiahnu záznamy z databázy a porovnajú sa s položkami v rss zdroji. Ak sa v databáze nachádzajú také, ktoré v zdroji nie sú, tieto sa zmažú. Príslušný zdrojový kód:

```
public function removeInactiveRssItems($rssSource,$RssItemManager)
```

```
{
            $itemsFeed = Array();
            $doc = new DOMDocument();
            $doc->load(trim($rssSource));
            foreach ($doc->getElementsByTagName('item') as $node)
            {
                  array_push($itemsFeed,$node-
>getElementsByTagName('description')->item(0)->nodeValue);
            }

            $rssSourceID                  =                  $RssItemManager-
>getIDByValue('rsssource','url',$rssSource,'idRSSSource');
            $itemsRows                    =                  $RssItemManager-
>getRssItemsBySourceId($rssSourceID);

            foreach ($itemsRows as $item)
            {
                  $description = $item['description'];
                  if (!in_array($description, $itemsFeed))
                  {
                        $RssItemManager->deleteRssItem($item['idRSSItem']);
                  }
            }
}
```

Na to, aby bolo možné zobrazovať filter aj pre iné zdroje, ako Vývesku, je nutné pozmeniť podmienku, ktorá rozhoduje o tom, či sa filtračná obrazovka zobrazí, alebo nie. Podmienka funguje tak, že sa overí, že či pre daný zdroj existujú nejaký záznamy, ktoré majú určenú kategóriu. Ak existujú, zobrazí sa filtrácia. (pozn.: Aktuálny stav tejto funkcionality nie je ešte funkčný a je nutné vykonať ďalšie úpravy kódu).

Logovanie, ktoré bolo zavedené predovšetkým kvôli ladeniu aplikácie, funguje nasledovne:

```
$logger = new Logger();
$logger->log("debug","retazec");
```

Uvedený zdrojový kód vytvorí tzv. Logger, čo je logovací objekt. Tento následne volá funkciu log(), ktorá ukladá daný záznam s určitou úrovňou (napr. debug, error, info a pod.) a reťazec, ktorý chceme logovať.

Mobilná verzia pre platformu Android

Aplikáciu sme vyvíjali v prostredí Eclipse s SDK (Software Development Kit) pre mobilnú platformu Android. Táto aplikácia sa skladá z dvoch hlavných obrazoviek. Prvou je obrazovka, v ktorej používateľ definuje id widgetu, ktorý chce zobraziť. Potom ako raz definuje používateľ id widgetu, sa tento údaj uloží do súboru. Pri novom spustení aplikácie sa id widgetu načíta zo súboru, aby bol používateľ odbremenený od jeho opätovného zadávania a hneď prejde na obrazovku č.2, ktorou je samotná mobilná webová verzia widgetu. Keďže je táto aplikácia natívno-webová, máme možnosť používateľovi ponúknuť menu, v ktorom mu ponúkneme možnosť zmeniť aktuálne id widgetu, čím umožníme prehľadávanie viacerých

widgetov. Taktiež po kliknutí na titulok pri zobrazení záznamov sa dostávame na externú stránku, z ktorej záznam pochádza, ale na widget sa môžeme vrátiť stlačením buttonu „späť“ na mobilnom telefóne. Ukážku obidvoch obrazoviek zobrazujem na obrázkoch nižšie.

Obr. 1. Úvodná obrazovka aplikácie

Obr. 2. Druhá obrazovka aplikácie

9. šprint (5.4.2011 - 18.4.2011)

Analýza

Je potrebné vytvoriť nový dizajn pre štandardnú stránku Widgetizéra. Rozhodlo sa, že vzniknú dve verzie Widgetizéra. Jedna špeciálne uspôsobená na stránku vyveska.sk a druhá všeobecne zameraná. Z tohto dôvodu je potrebné ich odlíšiť nielen funkčne, ale aj vzhľadom, čo uľahči používateľom orientáciu. Okrem toho je potrebné upraviť aj vzhľad widgetu pre špeciálnu výveskovú verziu Widgetizéru.

Pre úplnosť pri registrácii a prihlasovaní používateľov je potrebné umožniť používateľovi aj nanovo si môcť vygenerovať heslo v prípade, že jeho pôvodné niekto zistí.

Pre lepšiu prehľadnosť, prívetivosť a orientáciu používateľa pri tvorbe widgetu by sa malo na stránke zobrazovať aj v ktorej fáze aktuálne je tvorba Widgetu. To by malo zmenšiť frustráciu nového používateľa, ktorý dopredu nevie, koľko krokov ho očakáva pri tvorbe widgeu.

Špeciálna verzia projektu bude autonómna od riadnej verzie. Mala by na nej fungovať automatická aktualizácia dát. Počítať treba aj s inštaláciou tohto špeciálneho projektu a teda aj so skriptom, ktorý ju bude čo najviac zjednodušovať.

Mobilná aplikácia a webová mobilná verzia

Po tom, ako si používateľ zadefinuje zdroje pre widget a definuje jeho vzhľad, získava zdrojové kódy reprezentujúce definované widgety pre webovú stránku ako aj pre mobilnú webovú verziu a aplikáciu pre OS Android. Mobilná webová verzia ako aj verzia pre klientske stránky obsahuje v kóde definované id widgetu, na základe ktorého sa používateľovi zobrazí widget, ktorý si vytvoril. Taktiež číslo reprezentujúce id widgetu je potrebné pre androidovú aplikáciu, pri prvotnom definovaní widgetu, ktorý chceme zobraziť. Avšak id je reprezentované číslom, ktoré sa dá jednoducho zmeniť a tým má používateĺ k dispozícii widgety, ktoré vytvoril niekot iný. Preto treba tento údaj zmeniť, aby sa tomu zabránilo.

Webová mobilná časť widgetu doposiaľ obsahuje titulok, ktorý predstavuje linku, na ktorú treba kliknúť, aby sme prešli na hlavný zdroj a pri načítavaní stránky používateľ ostáva na pôvodnej stránke a kým ho nepresmeruje nevie, čo sa aktuálne vykonáva, čo môže vyvolať dojem zlej implementácie. Tento problém je takisto potrebné vyriešiť.

Úprava filtrácie a zobrazovania dát pri použití alchemy

Aplikovaním alchemy API sme dokázali zisťovať kategórie aj pre externé zdroje zadané používateľom. Tieto kategórie následne zobrazujeme pri voľbe kategórií vo forme checkboxov. Avšak je potrebné upraviť funkcionalitu výberu dát do widgetu na základe zvolených alechemy kategórií, pretože doteraz sa filtrovali kategórie iba pre náš defaultný zdroj. Je potrebné zmeniť samotné zobrazovanie kategórií ako checkboxov, následne výber dát podľa filtra do zobrazeného widgetu, mapovanie itemov k widgetu pri jeho ukladaní a filtrovanie dát pri zobrazení widgetu vloženého do stránky.

Návrh

Nový template by mal farebne jasne naznačovať, že ide o samostatnú verziu aplikácie, pričom však musí aj poukazovať na príbuznosť medzi verziami. Nový template by mal byť inou farebnou variáciou vývesovej verzie.

Podľa zadanej špecifikácie a obrázku by sa mal widget vo výveskovej verzii zobrazovať podľa iným spôsobom ako klasický všeobecný widget.

V záujme bezpečnosti je potrebné, aby vygenerovanie nového hesla bolo možné iba v prípade, že ide o naozajstného používateľa, ktorý si konto aj vytvoril. Preto by sa mala každá zmena hesla potvrdzovať pomocou linky poslanej v maile, ktorá bude obsahovať náhodný reťazec, ktorý nemožno uhádnuť. Len v prípade, že používateľ prihlásením do mailového klienta a kliknutím na linku priloženú v maile potvrdí záujem o zmenu hesla, sa toto naozaj zmení. Istou nevýhodou môže byť fakt, že týmto spôsobom si používateľ nemôže heslo sám určiť, avšak zabráni sa tým prípadu, kedy heslo zmení niekto, kto sa dostal k starému heslu, čím by originálny vlastník konta oň prišiel.

Do jednotlivých templatov je potrebné pridať textový reťazec obsahujúci zoznam krokov pri vytváraní widgetu a aktuálnu fázu zvýrazniť, aby používateľ vedel, koľko krokov musí urobiť na dokončenie tvorby widgetu.

Keďže sa vymedzila špeciálna verzia projektu, ktorá bude fungovať ako časť iného projektu – Vývеska, je nutné vytvoriť špeciálnu databázu, nad ktorou bude táto špeciálna inštancia projektu pracovať.

Okrem toho treba vytvoriť inštalačný skript pre tento špeciálny projekt a tiež aj pridať aktualizáciu tohto projektu do Cronu.

Mobilná aplikácia a webová mobilná verzia

Problém s id widgetom, ktoré môže používateľ jednoducho zmeniť vyriešime tým, že v databáze vytvoríme nový záznam, ktorý nazveme *activationCode* a bude predstavovať tzv. aktivačný kód, ktorý v prípade aplikácie pre OS Android používateľ zadá v úvodnej obrazovke. Taktiež ho využijeme pri ostatných verziách widgetu, kedy namiesto id budeme zobrazovať tento aktivačný kód reprezentujúci widget. Aby sme docielili unikátnosť aktivačného kódu a odstránili problém s jednoduchým prístupom k iným widgetom, definujeme ho ako zloženie id widgetu + náhodne generovaného trojznakového reťazca. To náš problém vyrieši.

Za cieľom zlepšiť mobilnú webovú prezentáciu widgetu sme upravili zobrazovanie jednotlivých záznamov tak, aby bola kliknuteľná celá plocha jednotlivých záznamov. Po následnom kliknutí na konkrétny záznam (plochu záznamu) sa používateľovi zobrazí informácia o načítavaní stránky, až pokiaľ nebude presmerovaný na stránku obsahujúcu detaily záznamu.

Úprava filtrácie a zobrazovania dát pri použití alchemy

Pri prechode zo stránky na zadanie rss zdrojov sa bude zisťovať, či niektorý zo zadaných zdrojov má namapované kategórie v databáze, či už alchemy kategórie alebo

kategórie z defaultného zdroja. V prípade, že je táto podmienka splnená, zobrazí sa obrazovka s výberom filtrov. Ak nie používateľ je presmerovaný na stránku štýlov, a do widgetu sa mapujú všetky dáta pre dané zdroje.

Pri zobrazení filtrov sa musí prehodnocovať, či bol medzi zvolenými zdrojmi aj defaultný zdroj. V prípade, že bol, tak sa zobrazia aj kategórie špecifické iba pre daný zdroj. Ak nie tak sa zobrazia len kategórie získané z alchemy API, pretože ostatné kategórie by používateľa mohli zmiasť.

Po zaškrtnutí kategórií sa k widgetu namapujú zvolené kategórie. Ak však nejaký zdroj nemá alchemy kategórie, nevieme tieto dáta filtrovať preto je potrebné zachovať predchádzajúci prístup, a teda pre daný zdroj sa zobrazia všetky dáta. Pri zobrazovaní widgetu sa zisťuje či k danému zdroju existujú kategórie. Ak áno zobrazia sa dáta filtrované na základe zvolených kategórií, ak nie zobrazia sa všetky dáta.

Avšak alchemy kategórie sa načítajú až po určitom čase od vloženia nového zdroja do databázy, pri spustení cron-u. Naskytá sa tu otázka, ako ošetriť situáciu, že pre novo vložený zdroj nie sú namapované kategórie a zobrazujú sa používateľovi všetky itemy z daného zdroja, avšak po určitom čase, ak sú pre daný zdroj zistené kategórie, pri zobrazení toho widgetu sa prehodnotí podmienka na existenciu kategórií, ktorá je splnená, ale keďže k danému widgetu nie sú uložené kategórie z filtrov, nezobrazia sa mu žiadne dáta.

Implementácia

Nový template v inej farebnej kombinácii a mierne zmenený layout zabezpečujú, že obe verzie aplikácie sú ľahko identifikovateľné a hlavne bez problémov rozlíšiteľné.

Dizajn vývesk ového widgetu je mierne odlišný od toho pôvodného no jeho nasadenie ešte viac rozširuje univerzálnosť celej aplikácie a ukazuje potenciál smerovať aplikáciu na rôzne domény záujmu.

Zmena hesla v prípade jeho zistenia inou osobou je funkčné a umožňuje používateľovi po vložení starého hesla a vpísaní nového hesla zmeniť si heslo podľa ľubovôle. Keďže heslo sa zmení až po kliknutí na linku odoslanú na kontaktný email použitý pri registrácii, zároveň je zachovaná aj bezpečnosť. Pre tento účel bol do tabuľky "user" pridaný ďalší stĺpec nevyhnutný na zmenu hesla.

Do jednotlivých templatov boli pridané reťazce, ktoré poukazujú na aktuálny priebeh pri vytváraní vlastného widgetu. Aktuálna fáza je zvýraznená boldom a o niečo väčšia ako ostatné časti.

Bola vytvorená nová databáza s názvom ‘vyveskaWidgetDB‘, ktorá ma identický dátový model, ako primárna databáza.

Do inštalačného skriptu bolo nutné zahrnúť pozmenenie konfiguračného súboru našej aplikácie, ktorý obsahuje o.i. názov databázy, na ktorú sa treba pripájať.

Do Cronu bola pridaná url:

```
http://localhost/team10issi/vyveska/widgetizer/document_root/cronjob/
```

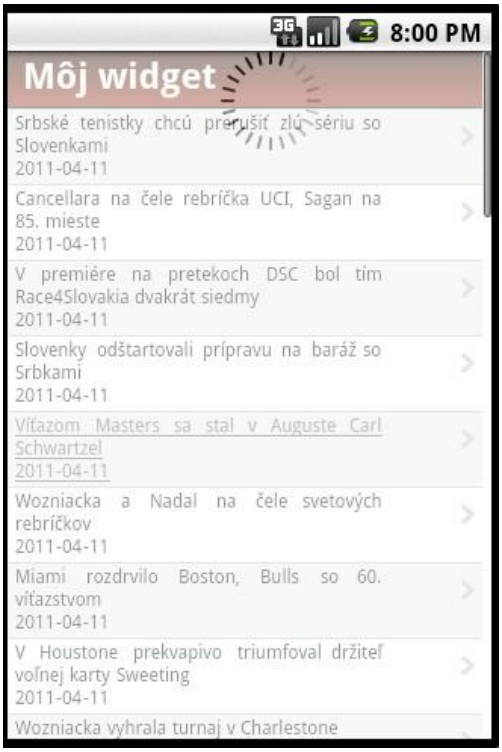
Mobilná verzia pre platformu Android

Pre definovanie aktivačného kódu pre widget sme implementovali funkciu, ktorá nám vráti identifikátor pozostávajúci z id widgetu a náhodne generovaného trojznakového reťazca. Funkcia na generovanie náhodného reťazca je zobrazená nižšie:

```
// Generate a random character string

function rand_str($length, $chars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz')

{

    // Length of character list

    $chars_length = (strlen($chars) - 1);

    // Start our string

    $string = $chars{rand(0, $chars_length)};

    // Generate random string

    for ($i = 1; $i < $length; $i = strlen($string))

    {

        // Grab a random character from our list

        $r = $chars{rand(0, $chars_length)};

    // Make sure the same two characters don't appear next to each other

          if ($r != $string{$i - 1}) $string .=  $r;
```

```
    }

    return $string;

}
```

Pre zmenu zobrazovania mobilnej webovej aplikácie za účelom vyššie uvedených cieľov sme vymenili kliknuteľnosť titulu záznamu za kliknuteľnosť celej plochy záznamu jej obalením do tagu *<a href>*. Pre zobrazenie správy o načítavaní sme použili javascript, ktorý sa aktivuje kliknutím na akýkoľvek záznam a trvá dovtedy, kým sa nedostaneme na zdrojovú stránku záznamu. Taktiež sme pridali šípku na pravú časť každéh záznamu symbolizujúcu možnosť kliknutia a zobrazenia celého záznamu. Náhľad spomínanej obrazovky po kliknutí zobrazujeme na obrázku nižšie.



Obr. 1. Obrazovka po kliknutí na niektorú z položiek

Úprava filtrácie a zobrazovania dát pri použití alchemy

Pre zisťovanie, či určitý zdroj má kategórie slúžia nasledujúce dve funkcie: hasCategoryItem a hasCategorySource. Prvá menovaná zisťuje či pre dané id rss itemu existuje namapovaná kategória. Druhá funkcia zisťuje, či pre daný zdroj existuje nejaká kategória a to tak, že pre každý item z daného zdroja zavolá funkciu hasCategoryItem.

```
public function hasCategoryItem($rssItemID)

	{

		$queryResult = dibi::query('SELECT * FROM [categoryitembind] 
WHERE [RSSItem_idRSSItem] = %i ',$rssItemID);
```

```
            $queryResult = $queryResult->fetchAll();
            if (count($queryResult) > 0)
            {
                  return TRUE;
            }
            else return FALSE;
      }

      public function hasCategorySource($rssSourceId)
      {
            $items = $this->getRssItemsBySourceId($rssSourceId);
            foreach ($items as $item)
            {
                  if ($this->hasCategoryItem($item['idRSSItem']))
                        return TRUE;
            }
            return FALSE;
      }
```

Bolo nutné upraviť funkciu getArrayOfRssItemId, ktorá vracia id-čka itemov, pre zvolené kategórie. Jej úprava bola potrebná z toho dôvodu, že vracala všetky itemy pre kategóriu bez ohľadu na zdroj. Problém je v tom, že k danej kategórii napr. „Umenie“ môžu byť namapované itemy aj z defaultného zdroja aj z iného externého zdroja. Preto sme museli uvažovať pri výbere itemov do widgetu aj zadefinované zdroje. Na nasledujúcom úryvku zdrojového kódu, je select do databázy, ktorý vracia itemy pre zadané kategórie a zdroje.

```
$queryToGetRssId = dibi::query(
                              'SELECT RSSItem_idRSSItem
                              FROM categoryitembind
                              JOIN rssitem, rsssource
                              WHERE RSSItem_idRSSItem = idRSSItem
                              AND RSSSource_idRSSSource = idRSSSource
```

```
                                AND CategoryData_idCategoryData = %i

                                AND idRSSSource IN %l ',

                                $arrayOfCategoryDataId[$i],
$sourcesWithCategory)->fetchAll();
```

Aby sme vedeli, pre ktoré zdroje chceme získať všetky dáta a pre ktoré potrebujeme získať itemy na základe filtrácie kategórií, boli potrebné taktiež zmeny. Nasledujúci kód zobrazuje jednu z úprav, ktorú bolo treba urobiť, pre identifikáciu zdrojov s kategóriami a bez alchemy kategórií.

```
$sourcesWithCategory = array();

$sourcesWithoutCategory = array();

foreach( $this->rssSourcesId as $source) {

        if($rssItemManager->hasCategorySource($source)) {

                array_push($sourcesWithCategory, $source);

        }

        else {

                array_push($sourcesWithoutCategory, $source);

        }

}
```

Dátový model

Aktuálny dátový model systému.