Príloha A – Technická dokumentácia

1 Technická dokumentácia

Nasledujúca kapitola prináša hlbší pohľad na fungovanie aplikácie a procesy, ktoré v nej prebiehajú. Skladá sa z dvoch hlavných podkapitol. Prvá opisuje fungovanie samotnej wiki, ktorá zaobaľuje množstvo funkcionality počas každej žiadosti na server a druhá prinesie pohľad na našu prácu a zmeny, ktoré sme vo wiki uskutočnili.

1.1 WikkaWiki

Táto kapitola prináša komplexný pohľad na fungovanie samotnej wiki, na ktorej je naše riešenie postavené. Porozumenie tejto problematiky prinesie čitateľovi lepšie pochopenie ďalších častí dokumentácie.

Samotná wiki sa skladá z niekoľkých dôležitých častí. Patria sem moduly pre správu volaní na server, konfiguráciu, prácu s databázou a prácu so stránkami vo wiki. Všetky sú prehľadne a hierarchicky usporiadané v adresári wiki.

Ako wiki pracuje môžeme vidieť na sekvenčnom diagrame dole (**obr. č. 1**), ktorý zobrazuje priebeh práce wiki. Diagram ukazuje sekvenciu procesov od http žiadosti prijatej od klienta až po odoslanie odpovede klientovi naspäť. Každá žiadosť na server musí smerovať na wikka.php stránku, pretože práve tento php skript vykonáva všetky operácie spojené s wiki a stránkami v nej uloženými. Ďalším údajom, ktorý musí http žiadosť obsahovať je parameter wakka. Jeho hodnota obsahuje názov wiki stránky, ktorú chceme zobraziť. Ďalšie údaje v http žiadosti nie sú povinné ale môžu sa vyskytovať. Môžu byť oddelené znakmi ’&’ alebo ’/’. V nasledujúcej tabuľke sú príklady formátu URL v http žiadosti a akcie, ktoré aplikácia vykoná.

URL	Akcia
http://server/	Automaticky presmeruje žiadosť na URL http://server/wikka.php.
http://server/wikka.php	Automaticky načíta parameter wakka z konfiguračného súboru a presmeruje na http://server/wikka.php?wakka=DefaultPage.
http://server/wikka.php?wakka=HomePage	Zobrazí stránku s názvom „HomePage“.
http://server/wikka.php?wakka=DistIndex&user=UserName	Ukážka viacerých parametrov, jedná sa o stránku s menom „DistIndex“ ktorá v prípade zadania ďalších parametrov zobrazí filtrované údaje na základe týchto

	parametrov. V tomto prípade pre používateľa s menom „UserName“.
http://server/wikka.php?wakka=DistIndex/edit	Ukážka tzv. zachytávača udalosti[1] s menom „edit“, ktorý indikuje pre wiki že danú stránku chce používateľ editovať.

Ako je vidieť, wiki vždy pracuje so súborom wikka.php a v ňom prebiehajú jednotlivé procesy generovania stránky. V prvom rade pri každom volaní skript všetko inicializuje a načíta konfiguráciu, následne vytvorí inštanciu aplikačnej logiky pre stránky lokalizovanú v súbore s názvom „Wakka.class.php“. Potom overí aktuálneho používateľa a získa o ňom údaje z databázy. Nasleduje proces získania údajov z URL v http žiadosti a po ňom skript predá ďalšie úlohy na riešenie už pre inštanciu aplikačnej logiky.

Tá v prvom rade validuje vstupné parametre a autorizuje používateľa. Ak všetko prebehne v poriadku, získa z databázy telo stránky. V ňom sa môžu nachádzať tzv. akcie[2], ktoré sú špeciálnymi akciami wiki. Každá akcia je vlastne ďalším php skriptom a wakka tieto akcie postupne volá podľa toho v akom poradí sú v stránke zapísané. Výsledky týchto akcií vkladá do finálnej stránky, ktorá sa bude odosielať používateľovi v http odpovedi. Po akciách nasledujú už spomínané zachytávače udalostí, ktoré sú uvedené v parametroch URL http žiadosti a ak sa v nej nejaký zachytávač nachádza, wakka uplatní jeho php skript a stránku náležite podľa neho modifikuje. Proces vytvárania stránky je ukončený vložením hlavičky a päty dopredu, resp. dozadu do vygenerovanej stránky. Hlavička a päta každej stránky sú definované v šablónach taktiež ako php skripty.

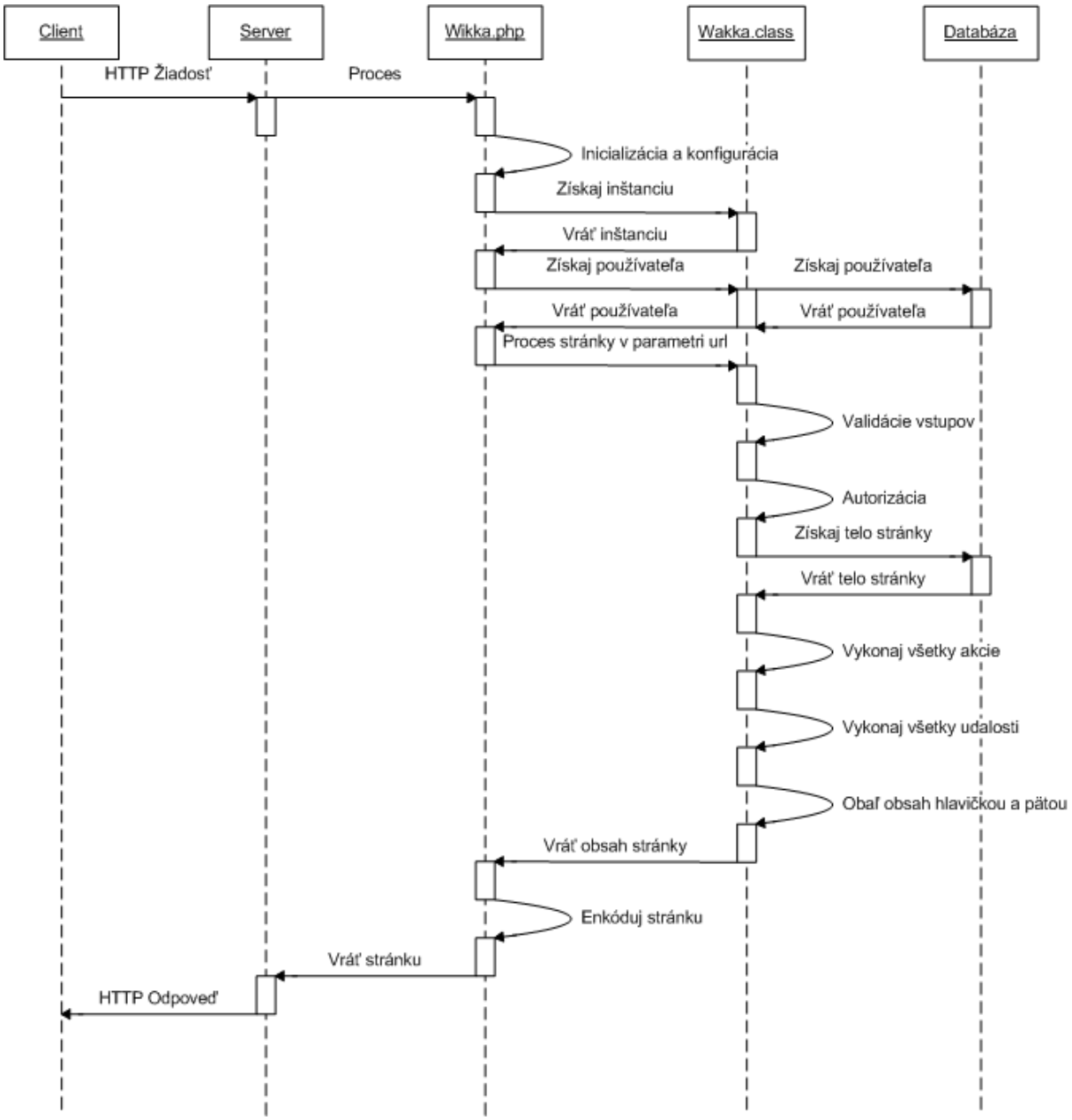
Po skončení tohto procesu wakka vráti výsledok pre wikka.php. Tá už len stránku zakóduje[3] a vráti serveru. Následne server vráti stránku pre používateľa.

---

[1] angl. „event handlers“

[2] angl. „actions“

[3] angl. „encode“



Obr. č. 1 - Sekvenčný diagram práce wiki od žiadosti až po odpoveď.

Nasledujúce podkapitoly sú ukážkou práce jednotlivých častí wiki, ktorých všeobecný a skrátený opis sme mohli vidieť v predchádzajúcom sekvenčnom diagrame.

1.1.1 Wikka.php

Ako už bolo spomínané, skript umiestnený v súbore **wikka.php** je srdcom celej aplikácie. Všetka funkcionalita sa vykonáva prostredníctvom nej. Stará sa o správne inicializovanie globálnych

premenných, konfiguráciu aplikácie a taktiež o report chýb, ktoré sa vyskytnú počas generovania stránky.

Konfigurácia sa skladá z dvoch častí. Prvá je definovanie najdôležitejších premenných bez ktorých wiki nemôže fungovať a ich uloženie do pamäte ako pole konfiguračných hodnôt. Táto definícia je napísaná natvrdo v kóde. Následne je načítaný konfiguračný súbor a údaje z neho sú zlúčené s týmito premennými a výsledkom je kompletná konfigurácia. Pri zlučovaní platí pravidlo, že ak existuje konfiguračný záznam natvrdo napísaný v kóde a taktiež existuje preň záznam aj v konfiguračnom súbore tak wiki použije záznam z konfiguračného súboru. Pre zlučovanie využíva wiki php funkciu **array_merge()**, ukážka zdrojového kódu je na obrázku **č. 2**.

```
257  $wakkaDefaultConfig = array(
258      'mysql_host'                => 'localhost',
259      'mysql_database'            => 'wikka',
260      'mysql_user'                => 'wikka',
261      'table_prefix'              => 'wikka_',
262      'root_page'                 => 'HomePage'
263  );
264
265  // load config
266  $wakkaConfig = array();
267  if (file_exists($configfile)) include($configfile);
268
269  $wakkaConfig = array_merge($wakkaDefaultConfig, $wakkaConfig);  // merge defaults with config from file
```

Obr. č. 2 - Ukážka zlučovania konfigurácií pomocou metódy array_merge().

1.1.2 Wakka.class.php

Tento skript obsahuje triedu tvoriacu logiku aplikácie a to najmä pre správu stránok a prácu s nimi. Kľúčovou je jej metóda **Run()**, ktorá je vstupným bodom celej logiky a ktorá je volaná zo skriptu **wikka.php**. Stará sa o validácie vstupných údajov, overenie autorizácie používateľa a o doplnenie chýbajúcich údajov v prípade, že neboli zadané. Tieto údaje berie z konfigurácie aplikácie. Tú vytvoril už predtým práve skript wikka.php a konfiguráciu predal ako vstupný parameter pre konštruktor triedy aplikačnej logiky **Wakka.class**.

Zároveň táto trieda obsahuje množstvo pred pripravených metód, ktoré sú využívané samotnou wiki alebo rôznymi akciami a zachytávačmi udalostí počas generovania stránok.

1.1.2.1 Používateľ

Wakka.class má k dispozícií množstvo metód aj pre správu používateľov. Nasleduje zoznam tých najdôležitejších alebo často využívaných metód, určené hlavne na získavanie údajov o používateľovi, keď je potrebné rozhodnúť, či je používateľ autorizovaný na určitú činnosť alebo ktoré dáta sa mu môžu zobraziť a ktoré nie.

LoadUser() je metóda, ktorá vráti všetky potrebné údaje o aktuálnom používateľovi, ktoré o ňom existujú v databáze. Využíva sa najmä pri prvotnej autorizácií, neskôr pri zobrazovaní používateľových nastavení. Počas behu wiki aplikácia často overuje, či je nejaký používateľ autentifikovaný alebo nie, vtedy využíva jednoduchší variant tejto metódy **GetUser()**. Táto metóda pracuje s reláciou[4] a údaje o používateľovi číta z nej, čiže priamo z RAM serveru a nemusí preto volať databázu.

```
1694    function LoadUser($name, $password = 0)
1695    {
1696        return $this->LoadSingle(
1697            "select * from ".$this->config['table_prefix'].
1698            "users where name = '".mysql_real_escape_string($name)."' ".
1699            ($password === 0 ? "" : "and password = '".
1700            mysql_real_escape_string($password)."'")." limit 1");
1701    }
```

IsAdmin() je metóda na určenie, či aktuálny používateľ systému má administrátorské privilégiá v systéme. Je veľmi často využívaná a obsiahnutá v množstve akcií, pretože administrátori môžu vykonávať všetky úkony v systéme.

```
1816    function IsAdmin($user='') {
1817        $adminstring = $this->config["admin_users"];
1818        $adminarray = explode(',' , $adminstring);
1819
1820        if(TRUE===empty($user))
1821        {
1822            $user = $this->GetUserName();
1823        }
1824        else if(is_array($user))
1825        {
1826            $user = $user['name'];
1827        }
1828        foreach ($adminarray as $admin) {
1829            if (trim($admin) == $user) return true;
1830        }
1831    }
```

UserIsOwner() je metóda, ktorá vráti informáciu, či je aktuálny používateľ vlastníkom stránky, ktorú si práve prezerá. Táto metóda je dôležitá napríklad pri určení, či používateľ môže editovať stránku alebo nastavovať práva pre prístup iných používateľov k nej.

---

[4] angl. „session“

```
1803    function UserIsOwner($tag = "")
1804    {
1805        // if not logged in, user can't be owner!
1806        if (!$this->GetUser()) return false;
1807
1808        // if user is admin, return true. Admin can do anything!
1809        if ($this->IsAdmin()) return true;
1810
1811        // set default tag & check if user is owner
1812        if (!$tag = trim($tag)) $tag = $this->GetPageTag();
1813        if ($this->GetPageOwner($tag) == $this->GetUserName()) return true;
1814    }
```

LoadACL() je metóda slúžiaca na načítanie používateľových práv, ktoré pre aktuálnu stránku má. Využíva sa pri autorizácií používateľa a taktiež pri zobrazovaní listov existujúcich stránok, aby sa v listoch nachádzali len údaje o stránkach, na ktoré má používateľ prístupové práva.

```
1856    function LoadACL($tag, $privilege, $useDefaults = 1)
1857    {
1858        if ((!$acl = $this->LoadSingle(
1859                        "SELECT ".mysql_real_escape_string($privilege).
1860                        "_acl FROM ".$this->config["table_prefix"].
1861                        "acls WHERE page_tag = '"
1862                        .mysql_real_escape_string($tag)."' LIMIT 1"))
1863                        && $useDefaults)
1864        {
1865            $acl = array("page_tag" => $tag, $privilege."_acl" =>
1866                    $this->GetConfigValue("default_".$privilege."_acl"));
1867        }
1868        return $acl;
1869    }
```

LogoutUser() je metóda na odhlásenie používateľa, ktorá sa stará o odstránenie všetkých stôp, ktoré používateľ v systéme zanechal a následne ho z neho odhlási.

```
1706    function LogoutUser()
1707    {
1708        $this->DeleteCookie("user_name");
1709        $this->DeleteCookie("pass");
1710        // Delete this session from sessions table
1711        $this->Query(
1712            "DELETE FROM ".$this->config['table_prefix']."sessions
1713            WHERE userid='".$this->GetUserName()."' AND sessionid='".
1714            session_id()."'");
1715        $_SESSION["user"] = "";
1716        // This seems a good as place as any to purge all session records
1717        // older than PERSISTENT_COOKIE_EXPIRY, as this is not a
1718        // time-critical function for the user.  The assumption here
1719        // is that  server-side sessions have long ago been cleaned up by PHP.
1720        $this->Query(
1721            "DELETE FROM ".$this->config['table_prefix'].
1722            "sessions WHERE DATE_SUB(NOW(), INTERVAL ".
1723            PERSISTENT_COOKIE_EXPIRY." SECOND) > session_start");
1724    }
```

1.1.2.2 Stránky

Nasledujúca kapitola opisuje metódy pre správu stránok, ktoré sú neodmysliteľnou súčasťou správy stránok wiki.

SavePage() je metóda využívaná na ukladanie stránok do databázy. Každá nová stránka alebo každá zmenená existujúca stránka sa pomocou tejto metódy vkladá do databázy. Pri zmenených stránkach metóda najprv označí všetky predchádzajúce verzie stránok ako neaktuálne a následne vytvorí nový záznam v databáze s aktuálnou podobou stránky. Takýmto spôsobom vytvára verzie stránok a pomocou wiki nie je problém vrátiť stránku do predchádzajúcich stavov. Príklad zdrojového kódu nižšie je aj so zmenami vykonanými počas našej práce, kde táto metóda bola rozšírená o nepovinný parameter **is_package_page**, ktorý definuje typ vytváranej stránky. Tie sme v rámci návrhu rozdelili na wiki stránky a stránky zobrazujúce informácie o balíčkoch. Parameter udáva či stránka ktorú ukladáme do databázy je onou spomínanou stránkou pre balíček.

```
888     function SavePage($tag, $body, $note, $owner=null, $is_package_page='N')
889     {
890         // get current user
891         $user = $this->GetUserName();
892
893         // TODO: check write privilege
894         if ($this->HasAccess("write", $tag))
895         {
896             // If $owner is specified, don't do an owner check
897             if(empty($owner))
898             {
899                 // is page new?
900                 if (!$oldPage = $this->LoadPage($tag))
901                 {
902                     // current user is owner if user is logged in, otherwise, no owner.
903                     if ($this->GetUser()) $owner = $user;
904                 }
905                 else
906                 {
907                     // aha! page isn't new. keep owner!
908                     $owner = $oldPage["owner"];
909                     $is_package_page = $oldPage["ispackagepage"];
910                 }
911             }
912             // set all other revisions to old
913             $this->Query(
914                 "update ".$this->config["table_prefix"]."pages set latest = 'N' where tag = '".
915                 mysql_real_escape_string($tag)."'");
916             // add new revision
917             $this->Query("insert into ".$this->config["table_prefix"]."pages set ".
918                 "tag = '".mysql_real_escape_string($tag)."', ".
919                 "time = now(), ".
920                 "owner = '".mysql_real_escape_string($owner)."', ".
921                 "user = '".mysql_real_escape_string($user)."', ".
922                 "note = '".mysql_real_escape_string($note)."', ".
923                 "latest = 'Y', ".
924                 "body = '".mysql_real_escape_string($body)."', ".
925                 "ispackagepage = '".mysql_real_escape_string($is_package_page)."'");
926
927             if ($pingdata =
928                 $this->GetPingParams($this->config["wikiping_server"], $tag, $user, $note))
929                     $this->WikiPing($pingdata);
930         }
931     }
```

GetPageTag() je ďalšou veľmi vyťažovanou metódou, ktorá vráti aktuálny názov stránky, ktorý je taktiež uložený v databáze.

```
666     function GetPageTag()
667     {
668         return $this->tag;
669     }
```

1.1.2.3 Databáza

Wakka.class má veľa metód na získavanie údajov z databázy ale v tejto sekcii si predstavíme implementovanú logiku, ktorá pomáha týmto funkciám a zaobaľuje množstvo funkcionality. Tým robí kód znovu použiteľným. Hovoríme o nasledujúcich troch metódach:

Query() je metóda na vykonávanie všetkých dotazov v databáze. Zapuzdruje najzákladnejšie volanie na databázu – funkciu **mysql_query()**. Metóda sa stará aj o ošetrenie výnimiek a taktiež výpis

do debugu ak je povolený v konfiguračnom súbore. Metóda Query() je teda metódou, pomocou ktorej poľahky môžete volať SQL dotazy kdekoľvek v kóde a nemusíte sa starať o rôznu funkcionalitu spojenú s prácou s databázou.

```
77      function Query($query, $dblink='')
78      {
79          // init - detect if called from objct or externally
80          if ('' == $dblink)
81          {
82              $dblink = $this->dblink;
83              $object = TRUE;
84              $start = $this->GetMicroTime();
85          }
86          else
87          {
88              $object = FALSE;
89          }
90          if (!$result = mysql_query($query, $dblink))
91          {
92              ob_end_clean();
93              die("Query failed: ".$query." (".mysql_error().")");
94          }
95          if ($object && $this->config['sql_debugging'])
96          {
97              $time = $this->GetMicroTime() - $start;
98              $this->queryLog[] = array(
99                  "query"      => $query,
100                 "time"       => $time);
101         }
102         return $result;
103     }
```

LoadAll() je metóda, ktorá vráti pole výsledkov z databázy. Vstupný parameter predstavuje dotaz na databázu v jazyku SQL. Následne je tento dotaz vykonaný pomocou vyššie spomínanej metódy **Query()**. Výsledky dotazu sú vrátené ako pole riadkov databázy.

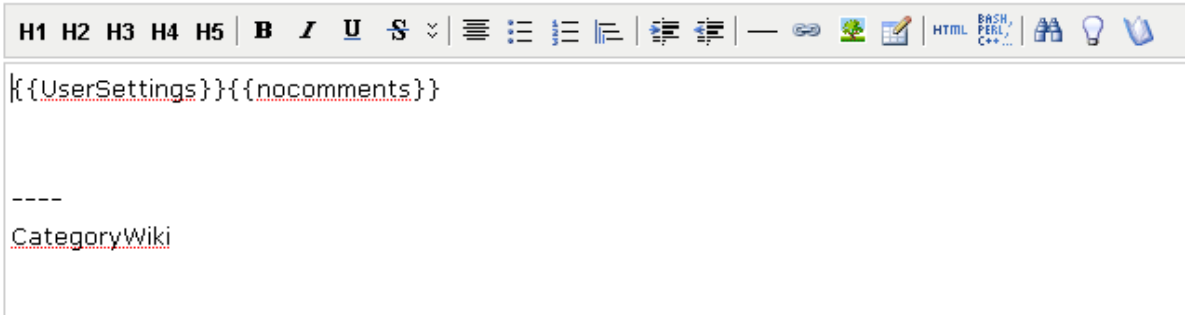
```
105     function LoadAll($query)
106     {
107         $data = array();
108         if ($r = $this->Query($query))
109         {
110             while ($row = mysql_fetch_assoc($r)) $data[] = $row;
111             mysql_free_result($r);
112         }
113         return $data;
114     }
```

LoadSingle() je metóda na získanie prvého vráteného riadku z databázy. Používa sa najmä v prípadoch, kde je potrebný len jeden výsledok z databázy a na ďalšie možnosti nie je potrebné prihliadať, napríklad pri načítavaní stránky z databázy, kde sa môže nachádzať viacero verzií stránky a pod. Využíva metódu **LoadAll()**, ktorá vráti pole výsledkov, následne zoberie prvý z nich a vráti ho ako výsledok volania.

```
104     function LoadSingle($query)
105     {
106         if ($data = $this->LoadAll($query))
107         {
108             return $data[0];
109         }
110     }
```

1.1.3 Akcie

Akcie sú špeciálnymi skriptami nachádzajúcimi sa v tele stránok a poskytujúcich rôznu funkcionalitu stránok. Na obrázku **č. 3** je ukážka tela stránky ako ho vidí pri editovaní používateľ. Do tela stránky vložil dve akcie, ktoré poznáme podľa toho že sa nachádzajú uprostred dvojitých zložených zátvoriek. Výsledkom takejto stránky po uložení do databázy a následnom zobrazení je, že sa najprv vykoná skript akcie **UserSettings**, ktorý zobrazí používateľove nastavenia a následne sa vykoná skript **nocomments**, ktorý zo stránky odoberie možnosť pridávať komentáre, pretože pre túto stránku si to používateľ neželá. Zdrojový kód skriptov akcií sa nachádza vždy v rovnomennom súbore, čiže napr. „Akcia.php“.



Obr. č. 3 - Ukážka použitia akcií v stránkach.

1.1.4 Udalosti

Udalosti[5] sa týkajú najmä operácií nad stránkami, nezasahujú priamo do textu (tela) stránky

[5] najvýstižnejší preklad angl. „handlers“ – „event handler“ je zachytávač udalostí, čiže niečo, čo čaká kým nenastane nejaká udalosť

ale slúžia na vyvolanie udalostí nad stránkami. Príkladom takejto udalosti, ktorá je najčastejšie vyvolávaná je udalosť „**edit**“. Po zavolaní stránky s touto udalosťou sa používateľovi otvorí okno pre editovanie stránky a zobrazí sa neformátovaná stránka ako editovateľný text.

Ďalšou typickou udalosťou je udalosť s menom „**acl**“. ACL[6] je určená na správu prístupových práv k stránke. Tie sú uložené v databáze. Každá stránka môže mať štyri operácie, ktoré môžu byť pomocou ACL limitované:

Operácia	Opis
Write	Slúži na označenie práv používateľov meniť stránku a priamo zasahovať do jej obsahu.
Read	Slúži na označenie práv používateľov vidieť obsah stránky.
Comment	Slúži na označenie práv používateľov vložiť svoj komentár k stránke ak stránka má možnosť komentáre vkladať.
Set Page Owner	Umožňuje zmeniť majiteľa stránky a predať práva vlastníka stránky na iného používateľa.

Vyvolanie nejakej udalosti je možné prostredníctvom menu v päte každej stránky alebo manuálnym dopísaním do URL žiadosti. Základné štyri udalosti sú viditeľné pre všetkých registrovaných používateľov a administrátorov. Patria medzi ne už spomínané udalosti „edit“ a „ACL“ a ďalšie dve, slúžiace na zobrazenie referencií na stránku (udalosť „**referrers**“) a zobrazenie histórie stránky a posledných zmien (udalosť „**history**“). Kompletný zoznam udalostí a ich podrobný opis sa nachádza v dokumentácií k WikaWikki.

1.1.5 Šablóny

Nie je na škodu priblížiť aj túto časť wiki a tou sú šablóny. Vo wiki sú použité dve šablóny a to pre hlavičku a pätu každej stránky.

Hlavička stránky obsahuje meno stránky a používateľské menu, ktoré sa generuje podľa toho či je používateľ prihlásený alebo nie. Odkazy v menu pre prihlásených a neprihlásených sa dajú konfigurovať v konfiguračnom súbore.

---

[6] skratka od „Access Control Lists“

Päta stránky slúži na rôzne udalosti a operácie nad stránkou, ktorá je práve zobrazovaná. Okrem základných štyroch udalostí sa tu nachádza čas poslednej zmeny, odkaz na tzv. „**rss feed**“ sledujúci zmeny stránky, ďalej informácie o vlastníkovi stránky a možnosť vyhľadávať text v stránke. V pravom dolnom rohu sa tu taktiež vyskytujú informácie pre vývojárov, ako sú napríklad informácie o validnosti stránky a jej štýlov podľa noriem **W3C**, údaj o čase potrebnom na vygenerovanie stránky a pri zapnutom ladení[7] SQL dotazov (cez konfiguračný súbor) aj údaje o všetkých SQL dotazoch, uskutočnených počas generovania stránky.

1.1.6 Konfigurácia

Konfigurácia bola už čiastočne spomínaná v časti **1.1.1** o wikka.php, kde sa jednalo hlavne o načítavanie a použitie konfigurácie. Nasledujúca kapitola má za úlohu priblížiť konfiguračný súbor, ktorý je voľne dostupný pre správcov systému a je priamo určený na konfiguráciu aplikácie.

Nasledujúca tabuľka uvádza zoznam najdôležitejších konfiguračných záznamov, ktoré sú pre používateľa systému k dispozícií a zlepšujú škálovateľnosť[8] aplikácie.

Kľúč => hodnota	**Vysvetlenie**
'mysql_host' => 'luadist.org'	Adresa servera kde sa nachádza databáza wiki.
'root_page' => 'HomePage'	Názov stránky, ktorá je nastavená ako hlavná stránka a zobrazuje sa na začiatku alebo v prípade, že žiadna stránka nebola zadaná v parametri wakka v URL.
'action_path' => 'actions'	Cesta, kde sa nachádzajú skripty akcií wiki.
'handler_path' => 'handlers'	Cesta, kde sa nachádzajú skripty udalostí wiki.
'stylesheet' => 'wikka.css?d7f30'	Názov štýlu, ktorý sa použije ako základný štýl wiki.
'navigation_links' => 'PageIndex :: RecentChanges :: RecentlyCommented'	Odkazy v menu pre neprihlásených používateľov. Wiki pri ho pri načítavaní formátuje a vytvára z neho linky na existujúce stránky.
'logged_in_navigation_links' => 'PageIndex :: RecentChanges :: RecentlyCommented :: RecentlyPublished'	Odkazy v menu pre prihlásených používateľov. Wiki pri ho pri načítavaní formátuje a vytvára z neho linky na existujúce stránky.

[7] angl. „debug“

[8] angl. „scalability“

'hide_comments' => '0'	Premenná indikujúca možnosti komentovať stránky používateľmi ak stránky komentovanie umožňujú.
'sql_debugging' => '0'	Premenná indikujúca možnosť zobraziť všetky SQL dotazy použité pri generovaní wiki stránky. Účinné najmä pre vývojárov.
'admin_users' => 'KajoMarton, HakiBaki, RimmoN, Ma5, KubKub'	Označenie jednotlivých používateľov systému ako administrátorov. Používateľ samozrejme musí byť korektne zaregistrovaný a údaje o ňom musia existovať v databáze.
'default_write_acl' => '!*'	Predvolené nastavenie práv na zápis stránok. Nastaví obmedzenie, ktoré zakáže neprihláseným používateľom meniť obsah stránok, táto možnosť ostane len prihláseným a administrátorom.
'default_read_acl' => '*'	Predvolené nastavenie práv na čítanie stránok. Nastavenie povoľuje vidieť všetky vytvorené stránky všetkým používateľom, vrátane neregistrovaných
'default_comment_acl' => '+'	Predvolené nastavenie práv na pridávanie komentárov stránok. Nastaví obmedzenie, ktoré zakáže neprihláseným používateľom komentovať stránky, táto možnosť ostane len prihláseným a administrátorom.
'allow_user_registration' => '1'	Premenná, ktorá povoľuje neprihláseným používateľom registrovať sa.

1.2 Luadist

V predchádzajúcej kapitole sme sa pozreli na funkcionalitu, ktorú wiki poskytuje po svojom nainštalovaní. Samozrejme, pre náš projekt potrebujeme ďalšie, ktoré budú priamo pracovať s balíčkami a repozitármi našich používateľov.

1.2.1 Registrácia používateľa

Registrácia používateľa bola z časti ponechaná ale bolo pridané vytváranie používateľského repozitára, čiže adresára kde budú uložené jeho balíčky. Tieto zmeny boli prevedené do skriptu pre akciu **UserSettings**. Ak tento skript overí všetky vstupné údaje, ktoré zadal používateľ pri registrácií,

vytvorí následne jeho používateľský adresár. Na tento úkon používa metódu **mkdir_r()**, ktorá rekurzívne vytvorí tento adresár.

```
131 if (!function_exists('mkdir_r')) {
132     function mkdir_r($dir)
133     {
134         if (strlen($dir) == 0)
135             return 0;
136         if (is_dir($dir))
137             return 1;
138         elseif (dirname($dir) == $dir)
139             return 1;
140         return (mkdir_r(dirname($dir)) and mkdir($dir, 0775));
141     }
142 }
```

Samotné volanie sa nachádza za ostatnými validáciami vstupných údajov a po úspešnom vložení používateľa do databázy. V prvom rade sa získa názov adresára pomocou „hash“ funkcie **md5()**, ktorá ako vstupný parameter zoberie používateľské meno a priradí k nemu náhodnú premennú. Vytvorením takéhoto „hash-u“ dosiahneme vyššiu bezpečnosť aby nebolo možné lokalizovať používateľský repozitár, ktorý je podľa špecifikácie privátny a tým pádom skrytý pre ostatných.

```
519 $path = "./repo/".md5($this->GetUserName().mt_rand());
520
521 if (!is_dir($path))
522 {
523     if (!mkdir_r($path))
524     {
525         echo "<strong>An error has been occured during the action mkdir("
526             . $path . ")!</strong>";
527     }
528 }
529 else
530 {
531     die("Couldn't create directory ".$path.
532         ", directory exists soon. Please contact system administrator.");
533 }
```

1.2.2 Generovanie manifestu

Manifest je generovaný 2 spôsobmi. Z databázy a pridaním záznamu na koniec existujúceho. Funkcie, ktoré implementujú túto funkcionalitu sú umiestnené v súbore luadist.lib.php.

```php
function readDistArchive($dist) {
    $unzip = unzip($dist, "tmp");
    if($unzip <= 0){
        return $dist . " is not a valid dist file.";
    }
    else {
        $distName = getFileName($dist);
        $distinfo_lines = file("tmp/" . $distName . "/dist.info", FILE_IGNORE_NEW_LINES | FILE_TEXT);
        if(is_null($distinfo_lines))
            return false;
        $toRet = array();
        doPairs($distinfo_lines, $toRet);
        $toRet["file_count"] = GetNumberOfFilesInArchive("tmp/" . $distName, true);
        rmdir_recursive("tmp/" . $distName);
        return $toRet;
    }
}
```

Funkcia **readDistArchive()** najprv rozbalí daný dist súbor na danej ceste do tmp adresára, podadresára s menom súboru. O toto sa stará funkcia unzip php pclzip.lib knižnice. Následne z danej cesty si zistí meno samotného dist súboru, a podľa neho nájde v rozbalenom priečinku dist.info súbor a načíta ho celý do jedného reťazca. Následne je tento reťazec rozobratý funkciou doPairs(), ktorá vráti asociatívne pole hodnôt z dist.info súboru. Funkcia doPairs() prechádza reťazec pričom hľadá oddeľovače lua zápisu objektov – tabuliek.

```php
function createManifestFromDb($page, $repository){
    $sqlResultData = $page->LoadAllPackageInfosFromRepo($repository);
    $manifest = fopen($repository . "/dist.manifest", 'w');
    if ($manifest) {
        foreach ($sqlResultData as $dist_info) {
            if(is_array($dist_info)) {
                $dist_info["file_name"] = substr($dist_info["page_tag"], strpos($dist_info["page_tag"], "_") + 1);
                writeDistInfoToManifest($manifest, $dist_info);
            }
        }
        fclose($manifest);
        return true;
    }
    return false;
}

/**
```

```php
function addToManifest($dist_info, $file_info){
    //open the manifest file
    $repository = $file_info["repo"];
    $manifest = fopen($repository . "/dist.manifest", 'a');
    if ($manifest) {
        if(is_array($dist_info)){
            $dist_info["file_name"] = $file_info["file_name"];
            writeDistInfoToManifest($manifest, $dist_info);
        }
        else{
            echo "error reading dist info of the ".$file_info["file_name"];
        }

        fclose($manifest);
        return true;
    }
    return false;
}
```

Funkcia **addToManifest()** pridáva informáciu o dist balíčku do existujúceho manifestu. Funkcia **createManifestFromDb()** zmaže existujúci manifest súbor a vytvorí nový na základe údajov získaných z databázy.

Obe tieto funkcie používajú funkciu **writeDidstInfoToManfiest(),** ktorá vytvorí zapíše formátovaný výstup do daného manifest súboru. Tento súbor otvorený na zapisovanie vo volajúcich funkciách.

```
150     function writeDistInfoToManifest($manifest, $distInfo) {
151         fwrite($manifest, $distInfo["name"] . " = {\n");
152         fwrite($manifest, "\t['" . $distInfo["version"] . "'] = {\n");
153         fwrite($manifest, "\t\t['" . $distInfo["arch"] . "'] = {\n");
154         fwrite($manifest, "\t\t\t['" . $distInfo["type"] . "'] = \t[[" . $distInfo["file_name"] . "]],\n");
155         fwrite($manifest, "\t\t},\n");
156         fwrite($manifest, "\t},\n");
157         fwrite($manifest, "}\n\n");
158     }
```

1.2.3 Wakka.class.php

Wakka.class ako miesto na lokalizáciu aplikačnej logiky na správu používateľov a stránok prešla taktiež viacerými zmenami, ktoré sme do nej implementovali. Pribudli hlavne metódy na prácu s balíčkami uloženými v databáze. Taktiež pribudli metódy na prácu s publikovanými balíčkami a napokon metódy pre kategórie balíčkov.

IsPackagePage() patrí medzi balíčkové funkcie a slúži na získanie informácie, či stránka, ktorú práve zobrazuje wiki je stránka s informáciami o balíčku. Takéto stránky majú v databáze nastavený stĺpec **ispackagepage** na ‘Y’.

```
730     function IsPackagePage($tag)
731     {
732         $result = $this->LoadSingle("SELECT ispackagepage FROM ".$this->config["table_prefix"].
733             "pages WHERE tag='".mysql_real_escape_string($tag)."'");
734         return $result['ispackagepage'] == "Y";
735     }
```

IsPagePublished() je metóda pre publikované balíčky a slúži pre zistenie informácie, či stránka, ktorá nesie informácie o balíčku bola používateľom publikovaná alebo nie. Táto informácia je dôležitá najmä pri zobrazovaní publikovaných balíčkov a pre vytvorenie menu pre publikovanie balíčka.

```
736     function IsPagePublished($tag)
737     {
738         $result = $this->LoadSingle("SELECT published FROM ".$this->config["table_prefix"].
739             "packages WHERE page_tag='".mysql_real_escape_string($tag)."'");
740         return $result['published'];
741     }
```

LoadAllPackagePageOwners() je metóda, ktorá vráti všetkých používateľov, ktorý vlastnia stránky s informáciami o balíčkoch. Potrebná je najmä pre filtrovanie údajov v rôznych zobrazeniach a prehľadoch o balíčkoch.

```
743     function LoadAllPackagePageOwners()
744     {
745         $query = "SELECT ".$this->config["table_prefix"]."packages.added AS added, ".
746             $this->config["table_prefix"]."packages.page_tag AS tag FROM ".
747             $this->config["table_prefix"]."pages LEFT JOIN ".
748             $this->config["table_prefix"]."packages ON ".
749             $this->config["table_prefix"]."pages.tag = ".
750             $this->config["table_prefix"]."packages.page_tag WHERE ".
751             $this->config["table_prefix"]."pages.ispackagepage = 'Y' AND ".
752             $this->config["table_prefix"]."pages.latest = 'Y' ".
753             "ORDER BY ".$this->config["table_prefix"]."packages.added";
754 
755         $result = $this->LoadAll($query);
756         $users = array();
757         $i = 0;
758         foreach ($result as $res)
759         {
760             if ($this->HasAccess('read', $res['tag']))
761             {
762                 $users[] = $res['added'];
763             }
764         }
765 
766         return array_unique($users);
767     }
```

IsCategoryPage() je metóda patriaca k metódam kategórií balíčkov. Slúži na získanie informácie o tom či zobrazovaná stránka zobrazuje všetky balíčky v danej kategórií. Metóda vytvorí spojenie dvoch tabuliek v databáze a porovnáva údaje v nich. Stránka má v sebe kategórie ak vlastník stránky je označený ako **is_category_admin**.

```
769     function IsCategoryPage($tag)
770     {
771         $result = $this->LoadSingle(
772             "SELECT ".$this->config["table_prefix"]."users.is_category_admin AS is_cat_admin FROM ".
773             $this->config["table_prefix"]."packages LEFT JOIN ".$this->config["table_prefix"]."users ON ".
774             $this->config["table_prefix"]."packages.added = ".$this->config["table_prefix"]."users.name WHERE ".
775             $this->config["table_prefix"]."packages.page_tag = '".mysql_real_escape_string($tag)."'"
776             );
777 
778         return $result['is_cat_admin'] == 'Y';
779     }
```

LoadAllPublishedPackages() slúži na správu všetkých balíčkov, ktoré boli publikované používateľmi. Metóda má dva nepovinné parametre, ktoré slúžia na filtráciu dát a to na zobrazenie publikovaných balíčkov len do určitého dátumu alebo zobrazenie publikovaných balíčkov len pre konkrétneho používateľa.

```
781     function LoadAllPublishedPackages($todate = null, $find_user = "")
782     {
783         if ($todate == null)
784         {
785             $query = "SELECT * FROM ".$this->config["table_prefix"].
786                 "packages WHERE (published_by IS NOT NULL) AND (published IS NOT NULL) ";
787
788             if($find_user != "")
789             {
790                 $query .= "AND (published_by LIKE '%".
791                     mysql_real_escape_string($find_user)."%' )";
792             }
793
794             $query .= "ORDER BY published DESC";
795
796             return $this->LoadAll($query);
797         }
798         else
799         {
800             return $this->LoadAll(
801                 "SELECT * FROM ".$this->config["table_prefix"].
802                 "packages WHERE (published_by IS NOT NULL) AND (published IS NOT NULL) ".
803                 "AND (DATEDIFF(published, '".mysql_real_escape_string($todate).
804                 "') >= 0) ORDER BY published");
805         }
806     }
```

LoadAllPublishers() patrí medzi používateľské metódy a je potrebná pri filtrovaní publikovaných balíčkov. Táto metóda vracia mená všetkých používateľov, ktorý publikovali nejaký svoj balíček.

```
807     function LoadAllPublishers()
808     {
809         return $this->LoadAll("SELECT published_by FROM ".$this->config["table_prefix"].
810             "packages WHERE (published IS NOT NULL) AND (published_by IS NOT NULL) ".
811             "GROUP BY published_by ORDER BY published_by");
812     }
```

GetAdminCategories() slúži na získanie údajov o všetkých kategóriách do ktorých sú balíčky triedené.

```
814     function GetAdminCategories()
815     {
816         return $this->LoadAll("SELECT * FROM ".$this->config["table_prefix"].
817             "users WHERE is_category_admin = 'Y' ORDER BY name");
818     }
```

IsPackageUnique() je potrebná pri publikovaní balíčka. Pred publikovaním totiž treba skontrolovať, či balíček, ktorý publikujeme náhodou v kategórií už neexistuje.

```
834     function IsPackageUnique($category, $package_name)
835     {
836         $result = $this->LoadSingle("SELECT COUNT(*) as count FROM ".
837             $this->config["table_prefix"]."packages WHERE page_tag = '".
838             mysql_real_escape_string($category."_".$package_name).
839             "' && added = '".mysql_real_escape_string($category)."'");
840
841         return ($result['count']) ? !($result['count'] > 0) : true;
842     }
```

LoadAllPackagePages() je v rebríčku najdôležitejších metód na najvyššom mieste. Využívaná je viacerými akciami, ktoré zobrazujú balíčky podľa rôznych klasifikácií, preto je potrebná jej určitá flexibilita, ktorú predstavujú tri nepovinné parametre. V prípade, že chceme získať z databázy len údaje o stránkach a nepotrebuje údaje o balíčkoch, ktoré akcie zobrazujú, tak nastavíme prvý parameter na hodnotu „false“. Sú akcie, ktoré ale potrebujú oboje údaje, vtedy vo volaní treba nastaviť prvý parameter na „true“. Ďalšie dva parametre slúžia na filtrovanie údajov. Druhý parameter slúži na filtrovanie výsledku podľa mena a tretí parameter indikuje či stačí aby meno autora iba obsahovalo zadaný reťazec alebo aby presne zodpovedalo zadanému reťazcu.

```
855     function LoadAllPackagePages($include_packages_to_result = false, $find_user = "", $strict = false)
856     {
857         if ($include_packages_to_result)
858         {
859             $query = "SELECT * FROM ".$this->config["table_prefix"]."pages LEFT JOIN ".
860                 $this->config["table_prefix"]."packages ON ".
861                 $this->config["table_prefix"]."pages.tag = ".
862                 $this->config["table_prefix"]."packages.page_tag WHERE ".
863                 $this->config["table_prefix"]."pages.ispackagepage = 'Y' AND ".
864                 $this->config["table_prefix"]."pages.latest = 'Y' ";
865
866             if ($find_user != "")
867             {
868                 if (!$strict)
869                 {
870                     $query .= "AND ".$this->config["table_prefix"]."packages.added LIKE '%"
871                         .mysql_real_escape_string($find_user)."%' ";
872                 }
873                 else
874                 {
875                     $query .= "AND ".$this->config["table_prefix"]."packages.added = '"
876                         .mysql_real_escape_string($find_user)."' ";
877                 }
878             }
879             $query .= "ORDER BY ".$this->config["table_prefix"]."packages.page_tag";
880
881             return $this->LoadAll($query);
882         }
883         else
884         {
885             return $this->LoadAll("SELECT * FROM ".$this->config["table_prefix"].
886                 "pages WHERE latest = 'Y' AND ispackagepage = 'Y' ORDER BY tag");
887         }
888     }
```

LoadAllAdminCategoryPages() je metóda, ktorá z databázy získa všetky stránky, ktoré patria do danej kategórie balíčkov.

```
890     function LoadAllAdminCategoryPages($category)
891     {
892         return $this->LoadAll("SELECT * FROM ".$this->config["table_prefix"].
893             "packages WHERE published_by IS NOT NULL AND added = '"
894             .mysql_real_escape_string($category)."' ORDER BY name");
895     }
```

LoadAllPackageInfosFromRepo() slúži na získavanie údajov o balíčkoch nachádzajúcich sa v konkrétnom repozitári, ktorý je zadaný ako vstupný parameter.

```
897     function LoadAllPackageInfosFromRepo($repo)
898     {
899         return $this->LoadAll("select * from ".$this->config["table_prefix"].
900             "packages where repo = '".mysql_real_escape_string($repo)."'");
901     }
```

GetUserRepository() je ďalšia potrebná metóda, ktorá vráti adresu repozitára pre konkrétneho používateľa. Táto metóda je používaná pri vytváraní a pregenerovaní manifestov. Používateľské repozitáre sú vytvárané pomocou „hash-u“ a náhodného čísla, pre to nie je možné získavať názov repozitára kedykoľvek počas behu len znovu vypočítaním cesty.

```
903     function GetUserRepository($username)
904     {
905         $repo = $this->LoadSingle("select repository from ".$this->config["table_prefix"].
906             "users where name = '".mysql_real_escape_string($username)."'");
907         return $repo['repository'];
908     }
```

1.2.4 Akcie

Úlohou tejto sekcie je oboznámiť čitateľa s vytvorenými akciami, ktoré je možné vkladať do wiki stránok. Počas tvorby projektu bolo vytvorených 9 nových akcií, ktoré tvoria hlavnú časť projektu a jeho funkcionalitu, ktorú sme implementovali. Najviac práce počas implementácií bolo strávených práve pre tieto akcie.

1.2.4.1 DistUpload

Prvou akciou, ktorá bude priblížená bližšie bude akcia **DistUpload**. Je tvorená univerzálnym php skriptom. Univerzálny preto, lebo je používaný pre dve navonok rozdielne funkcie a to pre nahrávanie balíčkov na server a ich publikovanie používateľmi pre verejnosť. Tieto dve funkcie majú rovnaký základ práve v tomto skripte.

Nahrávanie balíčkov používa štandardné funkcie jazyka php ako **move_uploaded_file()** a definovanú premennú _FILES. Po úspešnom nahraní príde na rad proces začleňovania balíčka do systému. Prečítajú sa údaje o balíčku uložené v ňom a zapíšu sa do databázy. Nasleduje generovanie manifestu v používateľom repozitári kde bol balíček nahraný. Tento proces je podrobne opísaný

v kapitole **1.2.2**. Nakoniec skript vygeneruje stránku pre balíček. Použije šablónu uloženú v adresári templates a vloží na miesta na to určené údaje o balíčku, ktoré získal. Takto vygenerovanú stránku uloží do databázy a nastaví privátne práva na čítanie, zápis a komentovanie stránky. Proces registrácie zobrazuje nasledujúci zdrojový kód. Vidíme, že každá časť procesu je ošetrovaná proti výnimkám. Metódy **writePackageInfoIntoDb()** a **makeManifest()** boli už spomínané v predchádzajúcej kapitole **1.2.2** metódy generujúce stránku sa nachádzajú nižšie v časti o publikovaní.

```
38  function RegisterPackage($page, $dist_info, $file_info, $is_publishing = false)
39  {
40      if (writePackageInfoToDb($page, $dist_info, $file_info, $is_publishing))
41      {
42          echo ("Info about uploaded package was saved to database successfully.<br />");
43      }
44      else
45      {
46          echo ("<strong>Error: Saving to database!</strong><br />");
47          return false;
48      }
49
50      if (makeManifest($dist_info, $file_info))
51      {
52          echo ("Manifest generated successfully.<br />");
53      }
54      else
55      {
56          echo ("<strong>Error: Make manifest was failed!</strong><br />");
57          return false;
58      }
59
60      if (GeneratePackagePage($page, $dist_info, $file_info, $is_publishing))
61      {
62          echo ("New page for package created successfully.<br />");
63      }
64      else
65      {
66          echo ("<strong>Error: Cannot generate page for package!</strong><br />");
67          return false;
68      }
69
70      return true;
71  }
```

Pri publikovaní balíčka sa využívajú tieto metódy tiež, ale predchádza im iný proces. Publikovanie balíčka totiž nie je nič iné len opätovné nahratie balíčka na server ale do iného adresára, reprezentujúceho kategóriu balíčkov. Preto je možné tento kód znova využiť pri tomto procese. Publikovaniu predchádza proces overenia či balíček, ktorý ideme publikovať do určitej kategórie tam už neexistuje. Ak touto kontrolou prejde, nasleduje presne rovnaký proces ako pri nahrávaní balíčka. Prečítajú sa údaje z neho, uložia sa do databázy, pregeneruje sa manifest v adresári kategórie a vytvorí sa stránka. Tejto stránke sa už ale nenastavujú privátne prístupové práva. Preto je v parametroch metódy **GeneratePackagePage()** údaj na identifikáciu procesu publikovania, ktorý mení toto správanie ako je vidieť v zdrojovom kóde nižšie. Ďalším rozdielom je, že pri publikovaní sa nepoužíva

aktuálny používateľ ale používateľ reprezentujúci kategóriu. Zabráni sa tým vytváraniu duplikátnych údajov o balíčkoch v databáze.

```
74  function GeneratePackagePage($page, $dist_info, $file_info, $is_publishing = false)
75  {
76      //get template
77      $template = GetTemplate("new_dist_page_template");
78
79      if ($template) {
80          //get body of the new page from template
81          $body = GetBody($page, $template, $dist_info, $file_info);
82
83          if ($body) {
84              //create page in db
85              $page->SavePage($file_info["page_name"], $body, "Autogenerated page.", $file_info["user"], 'Y');
86              //remove all permissions, only owner and admins could read/write/comment generated pages by default
87
88              if ($is_publishing)
89              {
90                  $page->SaveACL($file_info["page_name"], 'read', '*');
91                  $page->SaveACL($file_info["page_name"], 'comment', '*');
92              }
93              else
94              {
95                  $page->SaveACL($file_info["page_name"], 'read', '!*');
96                  $page->SaveACL($file_info["page_name"], 'comment', '!*');
97              }
98              $page->SaveACL($file_info["page_name"], 'write', '!*');
99
100             return true;
101         } else {
102             echo "Error: An error occured during creating package page content!<br />";
103             return false;
104         }
105     } else {
106         echo "Error: Cannot retrieve template for new package page!<br />";
107         return false;
108     }
109 }
```

1.2.4.2 DistIndex

DistIndex je akcia, ktorá má za úlohu zobrazovať balíčky viditeľné pre používateľa. Patria sem balíčky publikované, ním vytvorené alebo také, ktoré mu sprístupnili iní používatelia cez ACL.

DistIndex je taktiež php skript, ktorý na úvod vyčíta údaje z http metódy POST, kde sa môžu nachádzať informácie o filtrovaní zobrazovaných balíčkov. Používateľ môže v jeho menu voliť meno používateľa, ktorého balíčky chce vidieť.

Po úvodnom načítaní DistIndex zoberie vyfiltrované stránky a pokúsi sa balíčky vhodne zoskupiť podľa názvov. Môže sa stať, že jeden používateľ môže mať viacero balíčkov v rôznych verziách ale s rovnakým názvom. Vtedy sa tieto balíčky zoskupia a neskôr budú zobrazené ako jeden celok pre lepšiu orientáciu v nich. Zdrojový kód je znázornený nižšie. Metóda **GetGroupedPackages()** sa okrem zoskupovania stará aj o kontrolu či používateľ má práva stránky iných používateľov čítať. V prípade že nie, stránky sa späť do cyklu zobrazovania DistIndexu nevrátia. Taktiež metóda vyčlení zo skupiny balíček, ktorý používateľ označil na zmazanie. Túto možnosť DistIndex používateľom taktiež poskytuje.

```php
function GetGroupedPackages($page, $pages)
{
    if ($pages)
    {
        $result = array();
        //search in array
        foreach ($pages as $packagepage)
        {
            //only if user has read access
            if (!$page->HasAccess('read', $packagepage['page_tag']))
            {
                continue;
            }
            //calcute del_package string
            $del_package = "del_package_".str_replace(".", "_", $packagepage['page_tag']);
            //and check if is not deleted in current web request
            if (!(isset($_GET[$del_package])))
            {
                //compose a unique key for packages
                $key = $packagepage['added']."_".$packagepage['name'];
                //check if exists items for this package
                if ($result[$key])
                {
                    //add item to existing array
                    $packages[count($packages)] = $packagepage;
                    $result[$key] = $packages;
                }
                else
                {
                    //create a new array and set the unique entry
                    $packages = array();
                    $packages[0] = $packagepage;
                    $result[$key] = $packages;
                }
            }
            else
            {
                //a distindex needs to remember that one package is deleted now.
                $result['__package_to_delete__'] = $packagepage;
            }
        }
    }
    return $result;
}
```

Po zoskupení balíčkov sa začne proces zobrazovania. Každý používateľ, ktorý sa v zobrazovaných balíčkoch nachádza má svoju vlastnú tabuľku a do nej sa vkladajú údaje o jeho balíčkoch. Pre každý balíček sa kontroluje, či existujú práva pre používateľa zobrazujúceho DistIndex aby mohol balíček zmazať. To smú len administrátori a vlastníci balíčkov.

Mazanie balíčkov nastáva počas procesu zobrazovania. Ak používateľ predtým zvolil zmazanie niektorého z balíčkov, je meno tohto balíčku zaznamenané a ak počas zobrazovania naň skript narazí, zmaže ho a nezobrazí. Mazanie reprezentuje kód nižšie.

```
209  //delete page in DB
210  $this->Query("delete from ".$this->config["table_prefix"]."pages where tag = '"
211      .mysql_real_escape_string($page['page_tag'])."'");
212  //get package repo
213  $repo = $this->LoadSingle("select repo from ".$this->config["table_prefix"].
214      "packages where page_tag = '".mysql_real_escape_string($page['page_tag'])."'");
215  $repo = $repo['repo'];
216  //delete package in DB
217  $this->Query("delete from ".$this->config["table_prefix"]."packages where page_tag = '"
218      .mysql_real_escape_string($page['page_tag'])."'");
219
220  //delete childs if exists, applies to all admin category package pages
221  if ($page['published_by'])
222  {
223      $result = $this->Query("UPDATE ".$this->config["table_prefix"].
224          "packages SET published = NULL WHERE name = '".
225          mysql_real_escape_string($page['name'])."' AND version = '".
226          mysql_real_escape_string($page['version'])."' AND added = '".
227          mysql_real_escape_string($page['published_by'])."' AND arch = '".
228          mysql_real_escape_string($page['arch'])."'");
229  }
230
231  //get the name of the file (it has form f.e. KubKub_config-1.0.0-universal-all.dist, we dont want 'KubKub_'
232  unlink($repo ."/". substr($page['page_tag'], strpos($page['page_tag'], "_") + 1));
233  $repo = $this->GetUserRepository($current_username);
234  //create manifest in rpeo
235  createManifestFromDb($this, $repo);
236  //notify user
237  $index_output .= "Package and page ".$page['page_tag']." deleted. <br />";
```

1.2.4.3 RecentlyPublished

RecentlyPublished je posledná z dôležitých akcií, pretože zobrazuje pre používateľov všetky publikované balíčky. Prístup k nej majú aj neregistrovaní používatelia, taktiež sú tu možnosti filtrovať balíčky podľa dátumu publikovania alebo používateľského mena vlastníka balíčku.

1.2.4.4 CreateNewCategory

Akcia **CreateNewCategory** je prístupná len pre administrátorov a slúži pre vytváranie nových kategórií pre balíčky. Z časti využíva funkcionalitu existujúcej akcie **UserSettings**. Kategória je vlastne používateľ s administrátorskými právami, ktorý je ešte na viac označený v databáze ako **category_admin**. Každý takýto používateľ/kategória musí mať svoj repozitár a email. Preto pri vytváraní kategórie sa vlastne zobrazí formulár na tvorbu nového používateľa. Doňho sú vpísané meno, heslo a emailová adresa. Pri ukladaní do databázy sa taktiež vytvorí používateľský repozitár kategórie, pri tomto procese sa ale nepoužíva „hash“ funkcia, názov adresára odpovedá názvu kategórie. V tomto prípade totiž nepotrebuje skrývať adresár kategórie, obsah bude verejný. Posledným krokom je vygenerovanie stránky pre kategóriu a vloženie do tejto stránky akciu **ShowAdminCategories**, ktorá na tejto stránke vždy zobrazí všetky balíčky, ktoré budú do kategórie publikované.

```
default: //valid input, create user
{
    $this->Query("INSERT INTO ".$this->config['table_prefix']."users SET ".
        "signuptime = now(), ".
        "name = '".mysql_real_escape_string($categoryname)."', ".
        "email = '".mysql_real_escape_string(strtolower($email))."', ".
        "repository = '".mysql_real_escape_string($path)."', ".
        "password = md5('".mysql_real_escape_string($password)."'), ".
        "is_category_admin = 'Y'");

    $path = "./repo/".$categoryname;

    if (!is_dir($path))
    {
        if (!mkdir_r($path))
        {
            echo "<strong>Upload failed -> An error has been occured during ".
                "the action mkdir(" . $path . ")!</strong>";
        }
    }

    if (!GenerateAdminCategoryPage($this, "new_admin_category_page_template",
        $categoryname, $email, $path))
    {
        $error = "Cannot generate page for admin category!";
    }

    echo "<p>Category created successfully.</p>";
    echo "<p>Repository: <a href=".$repo.">".$this->htmlspecialchars_ent(substr($path,2)).
        "</a></p>";
    echo "<p>Page: ".$this->Format("[[".$categoryname."]]")."</p>";
}
```

1.2.4.5 Ostatné

ShowAdminCategories – akcia na zobrazenie všetkých existujúcich kategórií a adresy stránok, kde je možné prezrieť si ich obsah. Funguje na jednoduchom princípe. Zoberie z databázy všetky kategórie a zobrazí odkazy na ich stránky.

```
    echo "<h3>Admin categories:</h3>";

    if ($categories = $this->GetAdminCategories())
    {
        echo "<p><ul>";

        foreach ($categories as $category)
        {
            echo "<li><strong>".$this->Format("[[".$category['name']."]]")."</strong></li>";
        }
        echo "</ul></p>";
    }
    else
    {
        echo "<p>No admin categories found. Try to create one.</p>";
    }
```

AdminCategoryIndex – akcia zobrazujúca všetky balíčky danej kategórie.

Logout – akcia potrebná po zmene spôsobu odhlasovania používateľov. Cieľom vytvorenia tejto akcie bolo oddeliť odhlasovanie používateľov zo systému od stránky používateľských nastavení ako to bolo pôvodne.

DistUploadAuto – akcia vytvorená ako zjednodušená verzia akcie DistUpload. Bola z nej vynechaná funkcionalita na publikovanie balíčkov, pretože táto akcia slúži ako stránka pre externú aplikáciu na automatické nahrávanie balíčkov zo strany klienta. Taktiež nepotrebuje viacero validačných funkcií a potvrdzované výpisy o procese, pretože o validáciu sa stará už samotná externá aplikácia.

MailComments – akcia, ktorá po pridaní do stránky umožní posielanie informácií autorovi stránky prostredníctvom elektronickej pošty o nových komentároch pridaných do stránky inými používateľmi.

```
9   $this->query(
10      'UPDATE '.$this->config['table_prefix'].
11      "pages SET mail_comments = 1 WHERE tag = '"
12      .mysql_real_escape_string($this->GetPageTag()).
13      "'");
```

1.2.5 Šablóny

Úlohou tejto sekcie je oboznámiť s vytvorenými šablónami, ktoré boli pridané do wiki. Patria sem dve šablóny, ktoré sa používajú pre generovanie stránok. Jedná sa o generovanie stránok pri nahrávaní balíčka a vytváraní kategórie pre balíčky.

Pri nahrávaní balíčka sa zistia údaje o ňom a potom sa vytvorí stránka, kde sa tieto údaje zobrazia. Zoberie sa šablóna, ktorá sa stane telom budúcej stránky. Toto telo musí byť formátované ako wiki text. Pravidlá formátovania sa nachádzajú v dokumentácií k WikkaWiki. Nasleduje ukážka šablóny pre nový balíček.

```
1   ======%name%======
2   ----
3   Uploaded:    **%date%** by: **%user%**
4   File size:   **%size% Kb**
5   Stored in:   **""<a href="%repo%">%repo%</a>""**
6   Version:     **%version%**
7   Arch:        **%arch%**
8   Type:    **%type%**
9   Number of files in package: **%filesCount%**
10
11  **Description:**
12  %description%
13  ----
```

V šablóne sú použité rôzne formátovacie prvky wiki ako napríklad štyri pomlčky, ktoré značia vodorovnú čiaru (v html je to tag <hr>) alebo napríklad text oddelený dvojicami hviezdičiek bude

formátovaný ako tučné písmo (v html je to tag). V šablóne sa nachádzajú taktiež rôzne tzv. premenné, ktoré sú ohraničené znakom %. Tieto premenné budú potom nahradené údajmi z balíčka.

Pri generovaní stránky pre novú kategóriu sa využíva druhá šablóna. Jej obsah je strohejší ako v predchádzajúcom prípade, pretože nie je potrebné na týchto stránkach zobrazovať množstvo informácií. Na poslednom riadku šablóny vidíme, že v tele stránky sa bude nachádzať akcia **admincategoryindex**, ktorá pri každom zobrazení stránky pridá do stránky zoznam všetkých balíčkov v kategórií.

```
======%user%======

Admin: **%user%**
Email: **%email%**
Repo: **""<a href="%repo%">%repo%</a>""**

----

{{admincategoryindex}}
```

Ako generovanie prebieha na strane servera nám znázorňuje nasledujúca časť zdrojového kódu. Vidíme ako postupne volané metódy **GetTemplate()** a **GetBody()**, ktoré získajú najprv šablónu zo súboru a potom do šablóny vložia údaje.

```
74  function GeneratePackagePage($page, $dist_info, $file_info, $is_publishing = false)
75  {
76      //get template
77      $template = GetTemplate("new_dist_page_template");
78
79      if ($template) {
80          //get body of the new page from template
81          $body = GetBody($page, $template, $dist_info, $file_info);
82
83          if ($body) {
84              //create page in db
85              $page->SavePage($file_info["page_name"], $body, "Autogenerated page.", $file_info["user"], 'Y');
```

Metóda **GetTemplate()** hľadá danú šablónu v adresári pre šablóny a následne šablónu načíta zo súboru ako text.

Po nej metóda **GetBody()** tento text zoberie a začne mapovať na jednotlivé premenné v šablóne údaje o balíčku. Tie metóda získa v dvoch poliach ako parametre. Polia sú preto dve lebo pri načítavaní informácií o balíčku sa získali údaje o zip archíve v ktorom je balíček zabalený a pri jeho rozbalení sa získali ďalšie údaje zo súboru dist.info priloženého pri balíčku.

```php
function GetBody($page, $template, $dist_info, $file_info)
{
    $template = str_replace("%name%", $dist_info["name"], $template);
    $template = str_replace("%user%", $file_info["user"], $template);
    $template = str_replace("%date%", $file_info["upload_time"], $template);
    $template = str_replace("%size%", $file_info["file_size"], $template);
    $template = str_replace("%repo%", $file_info["path"], $template);
    $template = str_replace("%version%", $dist_info["version"], $template);
    $template = str_replace("%arch%", $dist_info["arch"], $template);
    $template = str_replace("%type%", $dist_info["type"], $template);
    $template = str_replace("%filesCount%", $dist_info["file_count"], $template);
    $template = str_replace("%description%", $dist_info["full"], $template);

    return $template;
}

//Rimmon: give a content of template
function GetTemplate($template_name)
{
    $template_dir = "./templates";
    $template = file_get_contents($template_dir . "/" . $template_name .
        ".disttmpl");
    return $template;
}
```

1.2.6 Hlasovanie

Ďalšou implementovanou časťou, ktorú sme vytvorili je funkcionalita hlasovania používateľov. Hlasovaním pridávajú alebo odoberajú body balíčkom. Informácie o hlasovaní sa nachádzajú v päte každej balíčkovej stránky. Každý používateľ môže hlasovať za jeden balíček len raz, preto bola do päty stránky pridaná kontrola stavu hlasovania pre používateľa. Autenticitu zabezpečuje IP adresa používateľa, ktorá je zaznamenávaná pre každé hlasovanie. Ak používateľ už hlasoval, v päte sú zobrazené len údaje o počte bodov, ak ešte nehlasoval, zobrazia sa mu dve ruky, so vztýčeným palcom resp. palcom ukazujúcim nadol. Po kliknutí na jeden z nich sa stránke pripočítajú resp. odčítajú body a obnoví sa aktuálny počet bodov.

Hlasovanie je implementované pomocou technológie AJAX[9] aby sa po kliknutí nevyvolalo klasické odoslanie formulára a stránka sa celá nanovo nevygenerovala a znova nenačítala u klienta. Aby AJAX fungoval, museli sme do stránok kde sa hlasovania nachádzajú pridať kód v jazyku javascript, ktorý sa stará o asynchrónnu komunikáciu so serverom.

Keďže na túto komunikáciu potrebujeme v prvom rade podporu XML http žiadosti, vytvorili sme metódu, ktorá sa stará o vytvorenie takéhoto objektu. Metóda ošetruje aj rôzne verzie najpoužívanejších prehliadačov.

---

[9] skratka od „Asynchronous Javascript and XML“

```
48  function GetXmlHttpObject()
49  {
50      var xmlHttp=null;
51      try
52      {
53          // Firefox, Opera 8.0+, Safari
54          xmlHttp=new XMLHttpRequest();
55      }
56      catch (e)
57      {
58          //Internet Explorer
59          try
60          {
61              xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
62          }
63          catch (e)
64          {
65              xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
66          }
67      }
68      return xmlHttp;
69  }
```

Keď bola základná funkcionalita pripravená bola implementovaná funkcia, ktorá je volaná po kliknutí na odkaz. V parametroch tejto funkcie sa nachádzajú údaje potrebné pre server keď bude poslaná XML http žiadosť naň. Funkcia najprv vytvorí XML http objekt pomocou metódy **GetXmlHtppObject()**.

Následne poskladá adresu stránky, ktorá bude asynchrónne volaná prostredníctvom XML http žiadosti. Do parametrov stránky vloží údaje o používateľovi, názov aktuálnej stránky, IP používateľa a akcia hlasovania, ktorá závisí od toho či ide o pozitívny alebo negatívny hlas. Akcia môže byť taktiež aj typu „reset“, ktorú môže vyvolať len administrátor a v tomto prípade sa jedná v vymazania údajov o ohlasovaní pre aktuálnu stránku.

Nasleduje proces, v ktorom funkcia zaregistruje metódu, ktorá bude čakať na odpoveď zo servera[10]. Metóda **stateChanged()** je bližšie opísaná nižšie. Registrácia je nutná kvôli tomu, že volanie na server je asynchrónne, čiže nečaká na odpoveď ale poverí inú metódu čakaním na túto odpoveď.

Po registrácií sa odošle XML http žiadosť. Ako metódu http používa GET.

---

[10] angl. „callback“

```
6  //xml representation
7  var xmlHttp;
8
9  //function called from html
10 function vote(user, tag, action, vote, ip)
11 {
12     xmlHttp=GetXmlHttpObject();
13     if (xmlHttp==null)
14     {
15         alert ("Browser does not support HTTP Request");
16         return;
17     }
18
19     //prepare call to server
20     var url="ajax/vote.php";
21     url=url+"?action=" + action;
22     url=url+"&user=" + user;
23     url=url+"&tag=" + tag;
24     url=url+"&vote=" + vote;
25     url=url+"&ip=" + ip;
26     url=url+"&sid=" + Math.random();
27
28     //register asynchronous call back event
29     xmlHttp.onreadystatechange = stateChanged;
30
31     //call server
32     xmlHttp.open("GET", url, true);
33     xmlHttp.send(null);
34 }
```

Metóda **stateChanged()** sa stará o odpoveď zo servera. Ak nejakú odpoveď dostane, vyberie z html **DOM**-u element s identifikačným menom „votes“ a vpíše doňho text, ktorý mu server vrátil.

```
36 //callback function
37 function stateChanged()
38 {
39     //check if state complete
40     if (xmlHttp.readyState == 4 || xmlHttp.readyState == "complete")
41     {
42         //modify document
43         document.getElementById("votes").innerHTML= xmlHttp.responseText;
44     }
45 }
```

Funkcionalitu, ktorá sa vykonáva na serveri reprezentuje skript „**vote.php**“. Ten sa stará o vykonanie príslušných úprav v databáze na základe akcie, ktorú používateľ pri hlasovaní vykonal. Ak sa jedná o reset, zmaže všetky hlasovania pre stránku. V prípade hlasovania vloží nový údaj do

tabuľky hlasovaní. Nakoniec získa aktuálny stav hlasovania po aplikovaní nových zmien a tento údaj vráti naspäť klientovi.

```
26      if ($action == 'reset')
27      {
28          $wakka->Query(
29              "DELETE FROM ".$prefix."votes WHERE page_tag = '"
30              .mysql_real_escape_string($tag)
31              ."'");
32      }
33      else
34      {
35          //SQL statement for inserting votes into DB
36          $wakka->Query(
37              "INSERT INTO ".$prefix."votes (page_tag, user, rating, ipaddress) VALUES ('"
38              .mysql_real_escape_string($tag)."', '"
39              .mysql_real_escape_string($user)."', '"
40              .mysql_real_escape_string($vote)."', '"
41              .mysql_real_escape_string($ip)
42              ."')");
43      }
44
45      //get current state of voting
46      $result = $wakka->LoadSingle(
47                  "SELECT SUM(rating) AS count FROM ".$prefix."votes WHERE page_tag = '"
48                  .mysql_real_escape_string($tag)
49                  ."'");
50
51      //check count
52      if (!$result['count'])
53          $result['count'] = '0';
54
55      //display count in content
56      echo '<span id="response"> Votes: '.$result['count'].'</span>';
```