LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

GRĀMATVĒDĪBAS SISTĒMAS OPTIMIZĀCIJA: MIGRĀCIJA UZ MYSQL/JAVA TEHNOLOĢIJĀM

BAKALAURA DARBS

Autors: **Igors Šakurovs**

Studenta apliecības Nr.: is11208

Darba vadītājs: Dr.dat. Zane Bičevska

RĪGA 2015

Anotācija

Darba mērķis ir grāmatvedības sistēmas optimizācija un uzlabošana ar programmas pilnīgas rekonstrukcijas palīdzību, tajā pašā laikā nemainot biznesa loģiku un sistēmas funkcionalitāti. Autors vēlas ne tikai paātrināt sistēmas darbību, bet arī atvieglot darbu izstrādātajiem. Konkrēti – visas biznesa loģikas pārcelšana datu bāzes līmenī un datu izvade un apstrāde ar Java tehnoloģiju palīdzību. Autors salīdzinās divas sistēmas pēc tādiem parametriem kā: uzbūve, drošība, veiktspēja un testēšana, lai konstatētu minētās metodes pozitīvās un negatīvās puses.

Atslēgvārdi: grāmatvedības sistēma, optimizācija, Java, MySQL

Abstract

Accounting system optimization: migration to Java/MySQL technologies

The goal of this work is to maintain the accounting systems optimization and improvement with the programs complete alteration, without changing basic business logic and systems functionality. The developer of the program desires not only to make the system to work faster, but also to make the work easier for the user, specifically to transform all business logic to data base and data output level and processing with Java technology's aid. The author compares two systems according to these parameters: structure, safety, performance and testing to find out strengths and weaknesses of both sides.

Keywords: accounting system, optimization, Java, MySQL

Satura radītājs

Saīsinājumi un termini

PHP - atklātā pirmkoda skriptu valoda, kura sākotnēji bija paredzēta servera puses lietojumos dinamiska tīmekļa lapu ģenerēšanai [1].

MySQL - ir relāciju datubāzu vadības sistēma [2].

Java - objektorientēta programmēšanas valoda [3].

HTML (Hiperteksta iezīmēšanas valoda) - iezīmēšanas valoda, kas ir izstrādāta tīmekļa lappušu un citas pārlūkprogrammā attēlojamas informācijas glabāšanai [4].

JSP (JavaServer Pages) - servera puses Java tehnoloģija, kas ļauj izveidot dinamiski ģenerētas tīmekļa lapas HTML, XML vai citā formātā [5].

JSTL (JavaServer Pages Standard Tag Library) – standarta tagu bibliotēka JSP [6].

Sāls – gadījuma datu rinda, kas tiek padota uz jaukšanas funkcijas ieeju kopā ar izejas datiem paroles rindas pagarināšanai, kas apgrūtina sākotnējās paroles atjaunošanu [6].

XSS (CrossSiteScripting) — uzbrukuma tips tīkla sistēmām, kas sevī ietver ļaundabīga koda iekļūšanu tīkla sistēmas izsniegtajā lapā (kods tiks izpildīts lietotāja datorā, tam atverot šo lapu) un šī koda mijiedarbību ar ļaunprātīgās personas tīkla serveri [8].

JSON (JavaScript Object Notation) — datu apmaiņas teksta formāts [9].

AJAX (Asynchronous Javascript and XML) — interaktīvā lietotāju web-programmas interfeisa veidošanas pieeja, kas iekļauj sevī datu apmaiņu fonā ar web – serveru. Rezultātā, datu atjaunināšanas brīdī tīmekļa vietne atjaunojās daļēji, tādējādi web-progammas paliek ātrākas un ērtākas. [10].

IDE (Integrated development environment) — sistēma, kas dod iespēju programmētājiem izstrādāt programmatūru. [11].

Ievads

Uz šo brīdi pastāv grāmatvedības sistēma, ko sauc OMA, kura ir uzrakstīta PHP programmēšanas valodā un datu glabāšanai izmanto MySQL datu bāzi. Visa biznesa loģika atrodas PHP kodā, kas nozīmē, ka MySQL datu bāzē nenotiek nekādas darbības ar datiem, nav ne procedūru, ne funkciju, ne trigeru, ne skatījumu. Sistēmai ir daudz trūkumu:

- sistēma ir novecojusi – tiek izmantotas vecas bibliotēku versijas un novecojuši programmēšanas paņēmieni
- sistēmai ir dažādas kļūdas
- tā lēni strādā – dažas darbības aizņem pārāk daudz laika
- slikts programmēšanas stils – grūti lasāms kods, nākas patērēt daudz laika, lai ar to tiktu skaidrībā un notestētu
- sistēma nav elastīga – ir sarežģīti pievienot jaunas funkcijas, kā arī jebkuras izmaiņas notiek tieši kodā nevis, piemēram, konfigurāciju failā.

Lai labotu minēto situāciju, sistēma tiks pilnībā pārveidota. Jaunā sistēma sauksies Scarlett. Visa biznesa loģika tiks pārcelta datu bāzes līmenī, bet datu izvadei un apstrādei tiks izmantotas Java tehnoloģijas, kopā ar JSP un JSTL. Šī programmēšanas valoda tika izraudzīta PHP vietā sekojošu iemeslu dēļ:

- augsta veiktspēja
- tiek izmantota MVC koncepcija
- milzīgs iebūvēto bibliotēku un iespēju daudzums
- tā ir pilnīgi objekt-orientēta valoda
- programmēšanas laikā ir grūti pieļaut kļūdu
- ir viegli testējama
- stingra mainīgo tipizācija

Visas mijiedarbības ar datiem tiks veiktas tikai ar MySQL procedūru palīdzību.

Viens no galvenajiem uzdevumiem ir tas, ka Scarlett ir jāievieš pakāpeniski, šajā laikā izmantojot veco datu bāzi, nesabojājot tajā datus.

Autors salīdzinās OMA un Scarlett pēc tādiem parametriem kā: uzbūve, veiktspēja, drošība, un testēšana, lai konstatētu minētās metodes pozitīvās un negatīvās puses.

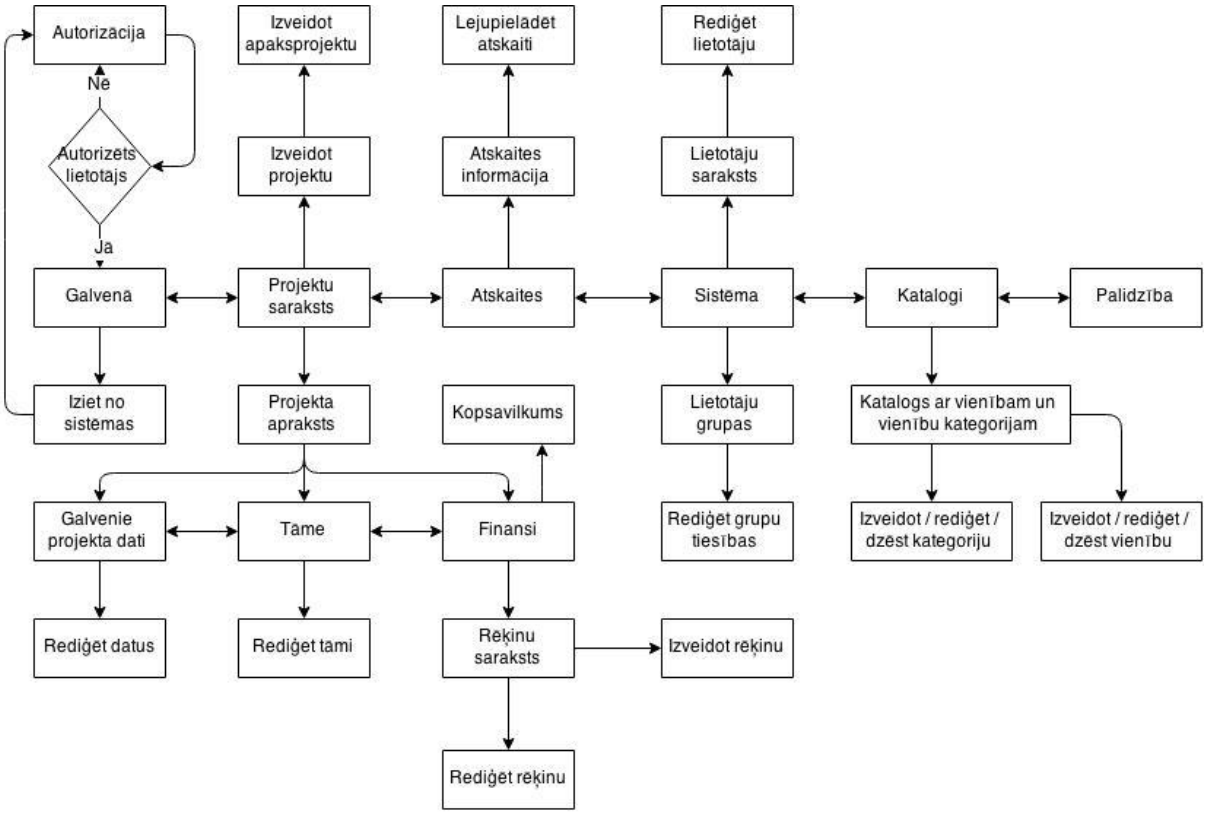
Sadaļā "Uzbūve" ir aprakstītas galvenas abu sistēmu uzbūves priekšrocības un trūkumi, kā arī kādas tehnoloģijas un koncepcijas tiek izmantotas. Sadaļā „Drošība" ir aprakstītas izmantotas metodes, lai nodrošinātu sistēmas datu drošību. Sadaļā „Veiktspēja" ir aprakstītas optimizācijas un veiktspējas palielināšanas metodes, kā arī metožu rezultāti izmērīti. Sadaļā „Testēšana un atkļūdošana" ir aprakstītas galvenas testēšanas un atkļūdošanas procesa priešrocības un trūkumi.

1. Sistēmas apraksts

Sākotnēji minētā grāmatvedības sistēma tika izstrādāta tikai vienas kompānijas vajadzībām, tādēļ tai ir unikālas iespējas un funkcijas. Vēlāk par sistēmu izrādīja interesi citas kompānijas, tagad to ir aptuveni 8.

Sistēmas galvenais mērķis ir atvieglot darbu grāmātvežiem. Tā palīdz nodrošināt pilnīgu uzņēmuma grāmatvedības uzskaiti, nodrošina pilnu grāmatvedības ciklu, sākot ar pirmdokumentu ievadi un beidzot ar gada pārskata sagatavošanu. Programmas sistēmai var pievienot vairākus lietotājus, katram no tiem piešķirot dažādas pieejamības pakāpes datiem.

1.1. Lietotāju saskarnes plūsmas



1.1.1. att. Sistēmas lietotāju saskarnes plūsmas

Ar sistēmu var strādāt tikai autorizēti lietotāji. Pēc autorizācijas lietotājs nokļūst galvenajā lapā, no kuras tas var pāriet uz jebkuru izvēlnes sadaļu. Lapā "Projektu saraksts" lietotājs var

izveidot projektu vai apakš projektu, kā arī atvērt kādu no sarakstā esošajiem projektiem, pēc kā parādīsies vēl viena izvēlne ar trim sadaļām. Lietotājs var aplūkot pamatinformāciju par projektu un to rediģēt, var pariet uz tāmi, kā arī uz sadaļu "Finanses", kurā ir projekta kopsavilkums un rēķinu saraksts, kurus var veidot un rediģēt. Sadaļā "Atskaites" lietotājs var izvēlēties atskaiti, aplūkot tās informāciju un tad to lejuplādēt. Sadaļā "Sistēma" lietotājs (ja viņam ir piešķirtas šādas tiesības) var rediģēt citus lietotājus, kā arī rediģēt lietotāju grupas – rediģēt viņu tiesības. Sadaļā "Katalogi" lietotājs var izvēlēties vienu no katalogiem, piemēram, kompānijas vai personas un veidot un rediģēt šajā katalogā kategorijas vai vienības. Sadaļā "Palīdzība" var izlasīt pēdējos sistēmas atjauninājumus un sistēmas lietošanas instrukciju.

2. Uzbūve

Šajā sadaļā tiks aprakstītas visas abu sistēmu izstrādes un uzbūves priekšrocības, kā arī to trūkumi.

2.1. OMA

Programmēšanas valoda PHP, bez bibliotēku lietošanas, ideāli der tikai mazu sistēmu izstrādei. Tā ir vienkārša un saprotama apmācību laikā, tādēļ mazāk pieredzējušie programmētāji bieži vien izmanto gan HTML gan PHP kopā. PHP valodā nav stingras datu tipizācijas, kas bieži izraisa kļūdas aprēķinos, bet grāmatvedības sistēmā šāda kļūdas nav pieļaujamas. Bieži vien PHP valodas galvenās priekšrocības ir arī tās trūkumi.

Minētajā valodā netiek izmantoti nekādi satvari, kā arī netiek izmantoti nekādi projektēšanas šabloni, tadi kā, piemēram, MVC. Tas nozīmē, ka vienā failā atrodas MySQL pieprasījumi, kā arī PHP un HTML kods, piemēram:

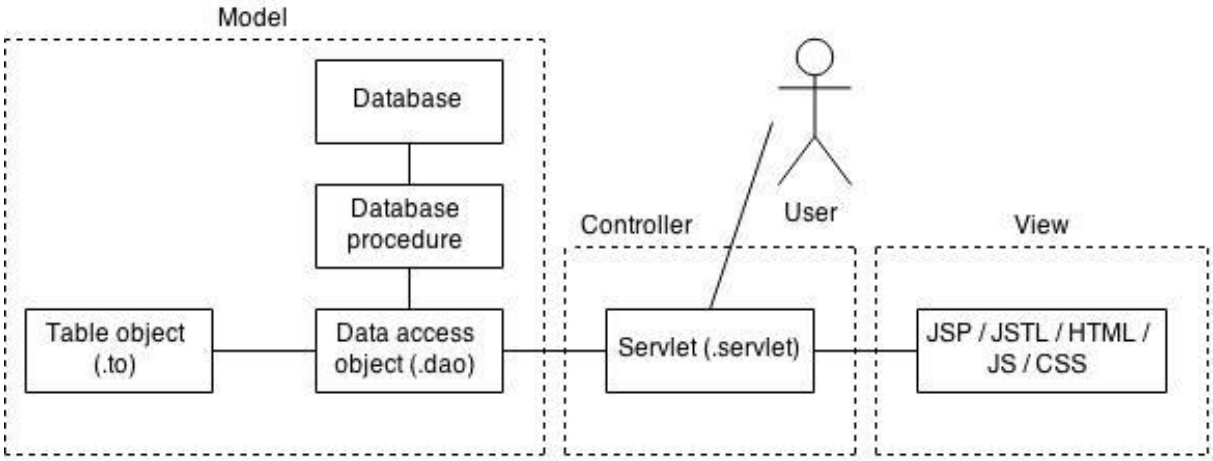
```
<?php
$sql ="SELECT*FROM enumTiers e LEFT JOIN regActivityTier reg ON (reg.tierID = e.id AND reg.activityID =
'$activityID')";
$result = mysql_query($sql) or die(mysql_error().$sql);
echo "<table class=\"tiers\">";
while($row = mysql_fetch_object($result)) {
?>
<tr id='<?=$row->id?>'>
    <td><input type='text' class='val tier' name='val' value='<?=$row->val?>'></td>
    <td><input type='text' class='proc tier' name='proc' value='<?=$row->proc?>'></td>
</tr>
<?
}
echo "</table>";
?>
```

1.2.1. att. OMA sistēmas koda piemērs

Ja jūs vēlēsieties globāli izmainīt dizainu, jums nāksies skart visas programmas daļas, jo „skatījums'' ir sajaukts ar citiem slāņiem.

2.2. Scarlett

Galvenā priekšrocība šīs sistēmas projektēšanā ir MVC [12] koncepcijas izmantošana, kas ļauj viegli mainīt slāņus neatkarīgi vienam no otra. Izstrādei tika izmantota programmēšanas valoda Java 7, izstrādes vidē "Eclipse IDE for Java EE Developers". Tīkla serveris Apache Tomcat 7. Turklāt tiek izmantots neparasts pielikumā arhitektūras veids, konkrēti – visa biznesa loģika atrodas datu bāzē un visas darbības ar datiem notiek ar procedūru palīdzību, t.i. ar sistēmu var pilnībā strādāt, izmantojot tikai MySQL procedūras. Minētais arhitektūras veids ļauj bez īpašām pūlēm pilnībā nomainīt pielikuma grafisko vidi pret citu vai darbam ar sistēmu izmantot citu programmēšanas valodu.



1.3.1. att. MVC koncepcija Scarlett sistēmā

Šajā diagrammā ir redzama programmas struktūra. Objekts .dao izmanto tabulas .to aprakstu un saņem datus no datu bāzes ar procedūras palīdzību, vai arī nosūta datus uz datu bāzi. Lietotājam veicot pieprasījumu, tiek izsaukta serversīklietotne, kas savieno datus ar veidu. Nododot datus programmai, servlets kontrolē datu pareizību.

Katra procedūra pēc izpildes atgriež datu kolekciju, kas sastāv no 4 kolonnām:

- id – izveidotā ieraksta, ja tāds ir bijis, identifikācijas numurs
- ret – sekmīgas izpildes gadījumā dotais mainīgais atgriež "1", pretējā gadījumā atgriež kļūdas numuru
- reload – norāda vai ir nepieciešama pilnīga lapas pārlādēšana
- fraction – minētās sadaļas esošās versijas numurs

Izsaucot procedūru, kura ļauj izmainīt datus, kā viens no parametriem tiek dots sadaļas versijas numurs, kas bija saņemts ielādējot lapu. Pēc operācijas sekmīgas izpildes datu bāzē, kā arī lapā dinamiski atjaunojas versijas numurs, kas, atkārtoti izsaucot funkciju, tiks nodots procedūrai. Neveiksmes gadījumā procedūra atgriež kļūdas kodu, piemēram "-1" nozīmē to, ka versijas numurs nesakrīt ar versijas numuru datu bāzē. Minētais mehānisms ļauj kontrolēt gadījumus, kad vairāki lietotāji vienlaicīgi vēlas izmanīt vienus un tos pašus datus. Parametrs reload ir nepieciešams gadījumiem, kad dati atjaunojas bez lapas pārlādēšanas, t.i. ar AJAX tehnoloģiju palīdzību.

Ar pakāpenisku sistēmas ieviešanu, datu bāzē tika izveidoti funkcijas un skatījumi, kuri neietekmē uz OMA darbu, bet ir nepieciešams izveidot trigerus, kuri izsaucās ja tiek mainīti daži noteikti dati. Lai triggeri izsaucās tikai Scarlett izmantošanas brīdī, tiek izpildīts pieprasījums "SET trigger_off = 1" failā, kur notiek pieslēgšana datubāzei OMA sistēmā, pieprasījums datu bāzē izveidoja mainīgo *trigger_off,* kurš norāda, ka dotajā sistēmā nedrīkst izmantot trigerus. Katrā trigerī tika pārbaudīts dots mainīgais.

2.3. Secinājumi

Scarlett ir daudz ērtāk programmēt komandā, jo sistēma ir sadalīta dažādos slāņos, katrs var programmēt tikai savu slāni, nekrustojoties ar citiem. Scarlett sistēmā bez problēmām var izmainīt dizainu, ko nevarēja izdarīt OMA sistēmā. Visai komandai nav obligāti jāzina sistēmas biznesa loģiku, pietiek, ja to zina tikai datu bāzes programmētājs, jo visa biznesa loģika atrodās datu bāzē. Šī programmas arhitektūra ļauj pilnībā mainīt grafisko vidi vai ļauj izmantot pilnībā citas tehnolģijas datu izvadei un apstrādei. Scarlett sistēmā visas operācijas ar datiem notiek datu bāzē, tas nozīmē, ka nav nepieciešamības saņemt datus no datu bāzes, apstrādāt tos un atgriezt atpakaļ, tas ievērojami palielina programmas veiktspēju.

3. Drošība

Šajā sadaļā autors salīdzinās abu sistēmu drošības risinājumus, jo grāmatvedības sistēmā tas ir viens no galvenajiem rādītājiem, tādēļ, ka sistēmā tiek uzglabāta konfidenciāla informācija un nesankcionēta piekļuve tai nekādā gadījumā nav pieļaujama.

3.1. OMA

Šajā sistēmā netiek izmantoti nekādi gatavie risinājumi, viss tika uzrakstīts patstāvīgi.

3.1.1. Autorizācija

Datu bāzē esošā tabula ar nosaukumu "catSecurityAccounts" satur pamatinformāciju par lietotāju, tādu kā ID, lietotājvārds, parole, pēdējās piekļuves sistēmai laiks, tās grupas ID, kurā ir lietotājs, u.t.t., taču šajā sadaļā mēs pievērsīsim uzmanību tikai lietotājvārdam un parolei.

account	password	groupID
ATOM	d3b6a8cd33c3fc2d96b0bc35cce2f1b3	20
ATOM_AI...	d3b6a8cd33c3fc2d96b0bc35cce2f1b3	20

3.1.1.1. att. catSecurityAccounts tabulas datu piemērs

Lietotāja parole tiek uzglabāta MD5 hešā, kas pat nesatur "sāli". Lietotāja autorizācija notiek sekojoši:

```
SELECT*FROM catSecurityAccounts WHERE account ='".$_POST['user']."'AND password
='".md5($_POST['pass'])."'"
```

3.1.1.2. att. Autorizācijas procesa koda fragments

Ja minētais pieprasījums ir atgriezis rezultātu, tas nozīmē, ka sistēmā eksistē lietotājs ar šādu lietotājvārdu un paroli, pēc kā notiek autorizācija, konkrēti – sesijā tiek izveidots objekts, kas satur informāciju par lietotāju.

```
$obj =new stdClass;
$obj->accountID = $row->accountID;
$obj->groupID = $row->groupID;
$obj->hash = md5($_POST['pass']);
$_SESSION["auth"] = $obj;
```

3.1.1.3. att. Autorizācijas procesa koda fragments

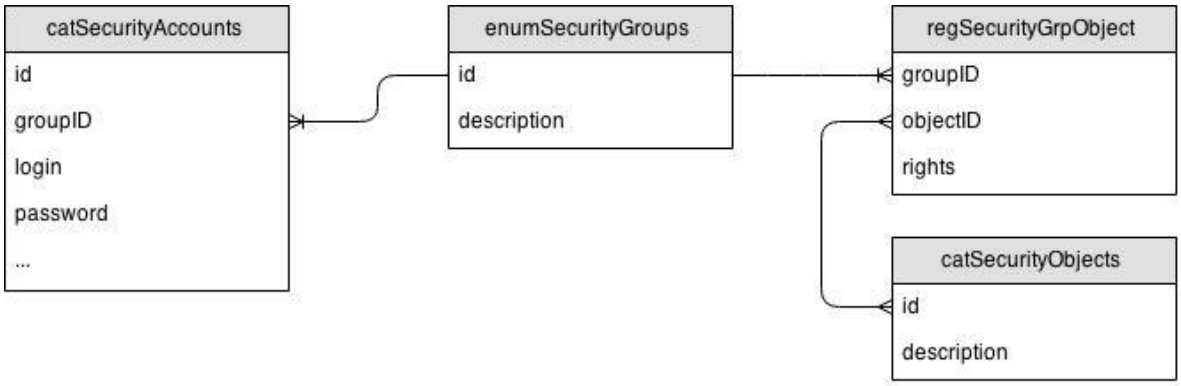
Turpmāk katrā lapā notiek pārbaude, kas ir līdzīga pārbaudei autorizējoties, kas pārbauda, vai lietotājs ir autorizēts.

3.1.2. Piekļuves tiesības

Sistēmā katrs lietotājs tiek attiecināts vienai no vairākām grupām, savukārt katrai grupai ir savas piekļuves tiesības un savas iespējas.

Datu bāzē atrodas tabulas, kuras realizē minēto mehānismu:

- "enumSecurityGroups" - satur lietotāju grupu sarakstu
- "catSecurityObjects" - satur visu to sistēmas sadaļu sarakstu, kurām ir ierobežota piekļuve
- "regSecurityGrpObject" - tabula, kura apvieno abas iepriekšējās tabulas, norādot, kurai grupai ir piekļuve konkrētai sistēmas sadaļai
- "catSecurityAccount" - tabula ar lietotāju datiem, kurā ir norādīts tās grupas ID, kurai ir piederīgs lietotājs



3.1.2.1. att. Entītiju relāciju diagramma drošības mehānismam

Šūna "rights" tabulā "catSecurityObjects" norāda, kādas tieši ir tiesības grupai minētajā sadaļā: 0 – tiesību nav, 1 – tikai skatīties, 2 – skatīties un rediģēt, 3 – pilna piekļuve

id	description
1	Developer
20	SuperUsers
30	Advanced
40	Managers
50	Creative
60	System

id	description
1	Project/Creative
2	Project/Media
12	Project/Main
13	Project/Summary
14	Project/Invoices
17	Catalogs/Companies

groupID	objectID	rights
20	1	3
20	2	3
20	12	3
20	13	3
20	14	3

3.*1.2.2. att.* Datu piemērs drošības mehānismam

Autorizējoties, lietotāja sesijā saglabājas tās grupas ID, kurai tas ir piederīgs. Piekļuves pārbaude notiek ar PHP funkcijas *check($obj, $perm)*, palīdzību, kurā atrodas nesarežģīts SQL pieprasījums.

```
check($obj, $perm){
...
$sql ="SELECT reg.groupID, c.description, reg.rights FROM regSecurityGrpObject reg
INNER JOIN catSecurityObjects c ON (c.id = reg.objectID) WHERE reg.groupID =\"". $auth->groupID ."\" AND
c.description = \"". $obj ."\" AND reg.rights >= ". $perm .""";
...
}
```

3.1.2.3. att. PHP funkcijas koda fragments – tiesību pārbaude

Piemēram, ja mēs gribam pārbaudīt vai lietotājam ir rediģēšanas tiesības sadaļā "Project/Media", ir nepieciešams izsaukt *check('Project/Media', 2).*

3.1.3. SQL injekcijas

Tā kā minētā sistēma neizmanto satvarus, ir nepieciešams rūpīgi kontrolēt un pārbaudīt visus datus, kuri tiek nodoti datu bāzes pieprasījumā. Ja jūs ievērojāt, tad SQL injekciju var izpildīt pat autorizējoties.

```
SELECT*FROM catSecurityAccounts a WHERE account ='".$_POST['user']."'AND password
='".md5($_POST['pass'])."'"
```

3.1.3.1. att. Autorizācijas procesa koda fragments

Masīva *$_POST* šūnā *user* var nodot jebkuru rindu, kura netiek kontrolēta, kas nozīmē to, ka, nododot atsevišķu pēdiņu, minētais pieprasījums jau neizpildīsies tā, ka to bija ieplānojis programmētājs. Katru mainīgo ir jāapstrādā ar iebūvēto funkciju

mysql_real_escape_string(), kura rindā ekranē noteiktus simbolus. Izstrādājot lielu sistēmu, ir diezgan sarežģīti izkontrolēt, lai visi pieprasījumi datu bāzē būtu aizsargāti.

3.1.4. XSS

Lai nepieļautu uzlaušanu ar XSS palīdzību, ir nepieciešams kontrolēt visus datus, kuri tiek doti sistēmas lietotājam, konkrēti, ar iebūvētās funkcijas *htmlspecialchars()* palīdzību nododamajos sistēmai datos ekranēt HTML speciālus simbolus uz HTML-būtībam. Pirmkārt, ir jānolemj, kurā brīdī nododamie dati tiks apstrādāti pret „ļaundarīgo kodu", to var darīt pirms saglabāšanas datu bāzē, vai arī nododot datus lietotājam, tajā pašā laikā glabājot datu oriģinālu datu bāzē. Šajā sistēmā tiek izmantoti abi gadījumi, kas bieži noved pie datu dubultās kodēšanas, vai arī pie "cauruma" sistēmā.

3.2. Scarlett

Šajā sistēmā, tās drošības nodrošināšanai, tiek izmantoti tikai gatavi risinājumi, kas ievērojami atvieglo darbu programmētājam un paaugstina sistēmas drošības līmeni.

3.2.1. Autorizācija

Lietotāju autorizācijai tiek izmantots gatavs risinājums – jdbcRealm [13], kas satur visas nepieciešamās funkcijas, kā arī ir ļoti elastīgs iestatīšanā. Lai aktivētu šo moduli, ir nepieciešams iestatīt tikai divus failus – servera Tomcat konfigurācijas failu *server.xml*, un programmas konfigurācijas failu *web.xml*.

```
<Context docBase="Scarlett" path="/Scarlett" reloadable="true">
<Realm className="org.apache.catalina.realm.JDBCRealm" connectionName="root"
connectionPassword="parole" connectionURL="jdbc:mysql://localhost/datubazes_nosaukums"
driverName="com.mysql.jdbc.Driver" roleNameCol="role_name" userCredCol="user_password"
userNameCol="user_name" userRoleTable="roles" userTable="users"/>
</Context>
```

3.2.1.1. att. server.xml faila koda fragments

Ir nepieciešams norādīt datu bāzi (connectionURL) un datus, lai pieslēgtos tai (connectionName, connectionPassword), kā arī norādīt lietotāju tabulu (userTable) un kolonnas, kuras satur lietotāju lietotājvārdus (userNameCol) un paroles (userCredCol).

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Scarlett</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>user</role-name>
    </auth-constraint>
</security-constraint>
```

3.2.1.2. att. web.xml faila koda fragments

Programmas konfigurācijas failā ir nepieciešams norādīt lapas, kuras būs pieejamas tikai autorizētiem lietotājiem, to var izdarīt ar patterna (url-pattern) palīdzību, minētajā gadījumā ir norādīts, ka lai piekļūtu jebkurai lapai (/*), ir nepieciešama autorizācija.

Šajā sistēmā tiek izmantota arī "sāls", "parole-sāls-lietotājvārds" veidā, pēc tam šī kombinācija tiek kodēta MD5 hešā. Minētā metode ievērojami sarežģī paroles dekodēšanu piekļuves datu bāzei gadījumā.

3.2.2. Piekļuves tiesības

Piekļuves ierobežošanai tiek izmantots tas pats modulis, ko izmanto autorizācijai - jdbcRealm, jo tas realizē visas nepieciešamās funkcijas. Piekļuves ierobežošana tiek realizēta ar lomu palīdzību, konkrēti, piekļuves tiesības noteiktai sadaļai saņem tie lietotāji, kuriem ir noteikta loma. Pielikuma konfigurāciju failā (web.xml) (zīm. 3.2.1.1) bija nepieciešams aizpildīt elementu *auth-constraint – role-name*, kurā ir norādīts, kādai lomai ir jābūt lietotājam, lai veiktu autorizāciju. Šajā gadījumā ir norādīta loma *user*, kas nozīmē, ka lietotājiem bez šīs lomas autorizācija nebūs pieejama. Servera konfigurācijas failā (server.xml) (zīm. 3.2.1.2) arī ir norādīta tabula, kurā ir lomu saraksts (userRoleTable) katram lietotājam un kolonna ar lomu nosaukumiem (roleNameCol).

Lomu tabulas vietā, uz OMA piekļuves tiesību mehānisma pamata (zīm. 3.1.2.1), tika izveidots skatījums (roles). Tika noņemtas tādas tiesības kā „tiesību nav" un „pilna piekļuve'', jo gadījumā, ja lietotājam nav lasīšanas tiesību, viņam nav nekādu tiesību, tajā pašā laikā, ja lietotājam ir rediģēšanas tiesībās, tam ir pilnas tiesības. Ja lietotājam bija pilna piekļuve sadaļai, lomu tabulā ar minēto sadaļu viņam bija tikai divas lomas "view_nosaukums" un

“write_nosaukums”. Piemēram, lietotājam „ATOM” ir piekļuve autorizācijai sistēmā, kā arī pilnas tiesības sadaļai “Catalogs/Activities”, veids atgriezīs šādu rezultātu:

user_name	role_name
ATOM	user
ATOM	view_Catalogs/Activities
ATOM	write_Catalogs/Activities

3.2.2.1. att. Skatījuma “roles” piemērs

Lomas pārbaudei gan Java, gan JSP tiek izmantota iebūvētā funkcija *isUserInRole(),* piemēram:

```
if(request.isUserInRole("Catalogs/Activities")) {
// ir tiesības
}
```

3.2.2.2. att. Lomu pārbaudes funkcija

Lietotāja lomas tiek nolasītas no datu bāzes tikai autorizējoties un tiek saglabātas sesijā, kas katrā lapā ievērojami samazina pieprasījumu skaitu datu bāzei.

3.2.3. SQL injekcijas

Minētajā sistēmā SQL injekciju praktiski nav iespējams izpildīt, jo visiem mainīgajiem ir stingrs datu tips un jebkura neatbilstība tipam novedīs pie programmas darbības beigām. Turklāt sistēmā visi pieprasījumi tiek izpildīti ar klases *PreparedStatement* palīdzību, kur visi mainīgie tiek nodoti pieprasījumam un tiek apstrādāti ar funkcijām, kas paredzētas minētā tipa mainīgajiem (3. pielikums). Ar citu metodi nodot mainīgo pieprasījumam nav iespējams. Kā arī, lai datu bāzes lauki tiktu pareizi aizpildītas, visi mainīgie tiek apstrādāti ar klasi *FieldsSetter* (1. pielikums), kurš pārbauda mainīgus,un, gadījumā, ja tie ir tukši, korekti aizpilda ailīti datu bāzē ar vērtību *NULL*.

3.2.4. XSS

Datu ekranēšana notiek skatījumā, tas nozīmē, ka visi dati datu bāzē glabājas oriģinālajā veidā. Šajā gadījumā ir diezgan viegli izkontrolēt, lai visi *String* tipa mainīgie būtu ekranēti, citu tipu mainīgajiem nav nepieciešamības pēc datu kontroles, jo tikai *String* var

saturēt ļaundarīgo kodu. Datu ekranēšanai tiek izmantota funkcija *escapeXml()* no JSTL bibliotēkas.

3.3. Secinājumi

Scarlett sistēmā tiek izmantoti gatavie risinājumi, kurus citi programmētāji jau daudzkārt ir pārbaudījuši, kas garantē stabilu darbību un aizsardzību un kam ir daudz priekšrocību:

- Modulis ir elastīgs un viegli regulējams, kas ļauj mainīt iestatījumus neskarot sistēmas programmas kodu. Piemēram, konfigurāciju failā ir iespēja norādīt tās lapas, kurām nav nepieciešama autorizācija.
- Vienkārša, bet droša piekļuves tiesību pārbaudes metode.
- Lietotāju lomas no datu bāzes tiek nolasītas tikai vienu reizi.
- Ir iespējams liegt autorizāciju atsevišķiem lietotājiem.
- Programmētājam nav jātērē laiks minēto funkciju izstrādei un testēšanai.
- Tiek izmantota "sāls".
- Nav iespējams veikt SQL injekciju.
- Viegli izkontrolēt vietas, kur iespējams pielietot XSS uzbrukumu.

OMA netiek izmantoti gatavie risinājumi, kas ievērojami samazina drošību:

- Ir iespējamas SQL injekcijas, jo ir nepieciešams kontrolēt un apstrādāt katru mainīgu, kas tiek nodots datu bāzes pieprasījumam.
- Ir iespējams XSS uzbrukums.
- Katrā lapā ir nepieciešams veikt lietotāja autorizācijas pārbaudi.
- Piekļuves pārbaudes funkcija katru reizi veic pieprasījumu datu bāzei, kas nozīmē to, ka būs tik daudz pieprasījumu, cik lapā būs piekļuves pārbaužu.
- Nav stingras mainīgo tipizācijas, kas liek veikt kontroli katram mainīgajam, kuru lietotājam ir iespēja izmainīt.
- Netiek izmantota "sāls".

Scarlett aizsardzība ir daudz drošāka, elastīgāka un vieglāk noregulējama nekā OMA aizsardzība.

4. Veiktspēja

Šajā sadaļā autors aprakstīs sistēmas veiktspējas paaugstināšanai izmantotās optimizācijas metodes, kā arī salīdzinās abu sistēmu moduļus pēc izpildes ātruma.

Lai noskaidrotu, ar kādu ātrumu izpildas katrs sistēmas modulis, un neņemot vērā interneta savienojuma ātrumu u.c. faktorus – failā, kurš izpildās pirmais, autors deklarēja globālo mainīgo *startTime*, kurš ietvēr sevī tekošo laiku milisekundēs. Tad, failā, kurš izpildās pēdējais, mainīga *startTime* vērtība tiek salīdzināta ar tekošo laiku, tad starpība būs skripta izpildes laiks. Šiem aprēķiniem PHP kodā tika izmantota funkcija *microtime()*, kura atgriež vērtību mikrosekundēs, bet Java valodā tika izmantota funkcija *nanoTime()* no *System* klases, kura atgriež laiku nanosekundēs.

```php
$startTime = microtime(true); // sakuma laiks
// darbība
$workTime = microtime(true) - $startTime; // darbības laiks
```

4.1. att. PHP koda izpildes laika aprēķināšana

```java
long startTime = System.nanoTime(); // sakuma laiks
// darbība
long workTime = System.nanoTime() - startTime; // darbības laiks
```

4.2. att. Java koda izpildes laika aprēķināšana

Lai mērījuma vērtības būtu precizākas, katr mērījums bija veikts piecas reizes, tikai pēc tam tika ņemta vidēja vērtība.

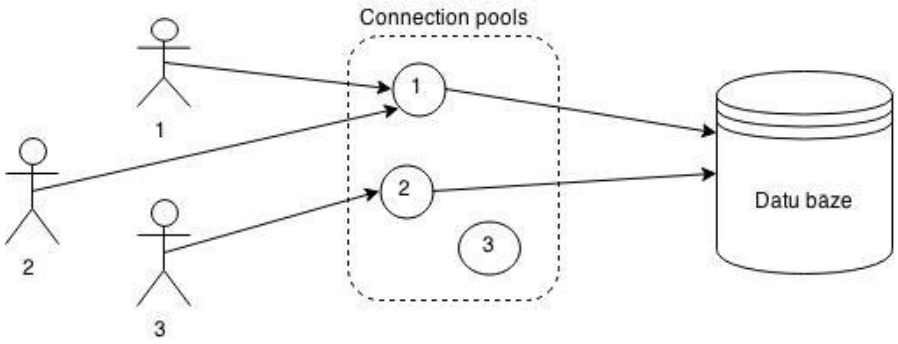
4.1. Pieslēgšanās datu bāzei

Abas sistēmas veic informācijas apmaiņu ar datu bāzi, izmantojot savienojumus. OMA katru reizi veido jaunu savienojumu, kam tiek patērēts daudz laika, jo transakcijas veikšana var aizņemt vairākas milisekundes, tajā pašā laikā savienojuma izveide var prasīt līdz pat vairākām sekundēm.

Var izveidot tikai vienu vienīgo savienojumu un vērsties pie datu bāzes izmantojot to. Taču šāds risinājums pie augstām slodzēm var radīt problēmas: ja daudzi lietotāji vienlaicīgi

centīsies saņemt piekļuvi datu bāzei, izmantojot vienu savienojumu, izveidosies rinda, kas arī atstāj negatīvu iespaidu uz programmas veiktspēju.

Scarlett sistēmā tiek izmantota Database Connection Pool (dbcp) [14] - tā ir augstāk minētās problēmas risinājums. Tas paredz, ka mūsu rīcībā ir zināms daudzums („Pool") savienojumu ar datu bāzi. Kad jauns lietotājs pieprasa piekļuvi DB, viņam no šī daudzuma tiek izsniegts jau atvērts savienojums. Ja visi atvērtie savienojumi ir aizņemti, tiek izveidots jauns savienojums. Līdz ko lietotājs atbrīvo vienu no jau esošajiem savienojumiem, tas kļūst pieejams citiem lietotājiem. Ja savienojums ilgi netiek izmantots, tas tiek slēgts.



4.1.1. att. DBCP izmantošanas piemērs

Attēlā 4.1.1. tika parādīts DBCP darbs – lietotāji "1" un "2" izmanto savienojumu "1", tajā laikā lietotājs "3" izmanto savienojumu "2", tas nozīmē, ka savienojumā "1" ir lietotāju skaita ierobežojums – 2 lietotāji. Savienojums "3" netiek izmantots, un tuvākajā laika tiks slēgts. Šīs piemērs parāda, kā samazinās datu bāzes pieslēgumu skaits, trīs vietā ir tikai divi. Scarlett sistēmā ir lietotāju skaita ierobežojums, uz vienu pieslēgumu 100 lietotāju, tas nozīmē, ka pieslēgumu skaits samazinās 100 reizēs.

Pieslēguma ātruma noteikšanai autors izmantoja tādu pašu metodi, kā tīmekļa lapas izpildes ātruma noteikšanai, tikai skaitītājs deklarējās pirms savienojuma izveidei un uzreiz pēc tā.

4.1.2. tabula

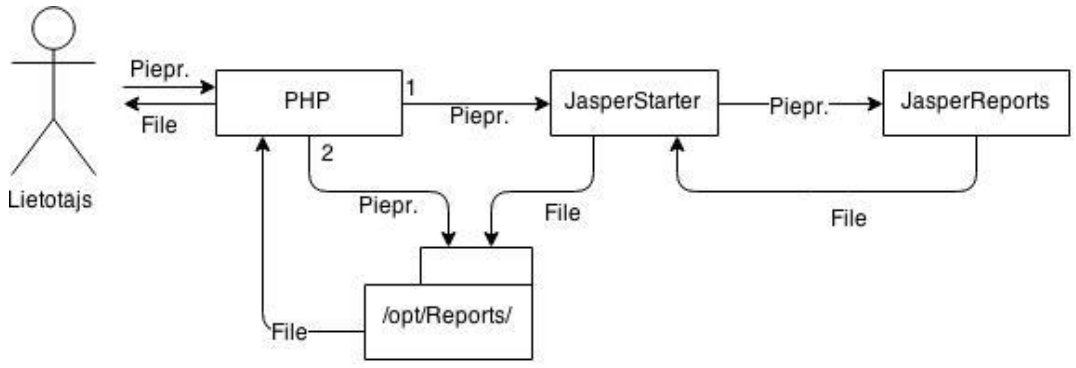
	1	2	3	4	5	Vidējais lielums
OMA	42.419 ms	25.568 ms	13.699 ms	23.260 ms	10.209 ms	23.031 ms
Scarlett	1.175 ms	0.510 ms	0.540 ms	0.572 ms	0.625 ms	0.684 ms

Tabulā var redzēt, ka savienojuma izveides laiks ir samāzīnajies vairākas reizes.

4.2. Atskaites

Viena no sistēmas svarīgākajām funkcijām ir atskaišu veidošana. OMA sistēmā minētā uzdevuma veikšanai tiek izmantota PHP klase/bibliotēka "PEAR::Spreadsheet_Excel_Writer", kura ļauj ģenerēt failus Microsoft Excel programmai. Vienas atskaites programmēšana prasa milzīgu laika patēriņu, jo darbs notiek ar minētās klases funkcijām, kas nozīmē, ka ir nepieciešams pilnībā aprakstīt veidojamo failu no sākuma līdz beigām. Ir nepieciešams norādīt lapas izmēru, kolonnu izmēru, atkāpes, robežas, aizpildīt ar datiem katru nepieciešamo šūnu, vienlaicīgi norādot šūnas precīzu adresi, kā arī daudz ko citu. Jā nākotnē būs nepieciešams globāli izmainīt atskaites veidu, vieglāk ir izveidot jaunu atskaiti, nekā tikt skaidrībā ar esošo.

Aprakstītā metode paredz atskaišu izveidošanu .xls formātā, taču ir parādījusies nepieciešamība ģenerēt atskaites arī citos formātos, piemēram, .pdf. Tika nolemts izmantot programmu JasperReports, kura ļauj ērti un ātri izveidot atskaites, vēlāk tās eksportējot dažādos formātos. Tā kā PHP ir salīdzinoši maz bibliotēku darbam ar citām programmam, tika pielietota diezgan izsmalcināta metode:



4.2.1. att. JasperReport atskaites iveidošana ar JasperStarter palīdzību

Lietotājam pieprasot atskaites izveidi, PHP ar komandas rindas palīdzību (funkcija exec) iedarbināja JasperStarter, kas savukārt vērsās pie JasperReports atskaites ģenerēšanai. Pēc atskaites izveidošanas JasperReports nodeva datus uz JasperStarter, kurš uz datu pamata izveidoja failu un saglabāja to mapē "/opt/Reports/". Pēc minētā procesa izpildes PHP meklēja mapē failu ar noteiktu nosaukumu un nodeva to lietotājam.

Scarlett sistēmā atskaišu ģenerēšanai JasperReports ir nepieciešama bibliotēka "net.sf.jasperreports.engine" un viena funkcija katram atskaites formatam (2. pielikums).

4.2.2. tabula

	1	2	3	4	5	Vidējais lielums
OMA (PEAR)	983.345 ms	824.569 ms	993.469 ms	793.355 ms	834.667 ms	885.881 ms
OMA (JasperStarter)	3.12 s	3.17 s	3.40 s	3.93 s	3.27 s	3.378 s
Scarlett	668.573 ms	201.798 ms	307.074 ms	131.557 ms	254.087 ms	312.618 ms

Tabulā 4.2.2. var redzēt, ka atskaišu izveide Sarlett sistēmā notiek ievērojami ātrāk, salīdzinājumā ar OMA. Atskaišu izveide ar PEAR bibliotēkas palīdzību arī strādā diezgan ātri, bet atbalta tikai vienu formātu – .xls, kas neatbilst nepieciešamai sistēmas funkcionalitātei.

4.3. Katalogi

Visi katalogi sistēmā ir līdzīgi pēc izskata un funkcionalitātes (iekavиņās ir norādīts operācijas izpildes laiks OMA sistēmā) :

- Izveidot kategoriju – jaunas kategorijas izveide (151.340 ms).
- Rediģēt kategoriju – izvēlētās kategorijas rediģēšana (140.493 ms).
- Dzēst kategoriju – izvēlētās kategorijas dzēšana (137.859 ms).
- Kategorijas izvēle – izvēloties vienu no kategorijām, parādās vienības kuras attiecās uz izvēlēto kategoriju (šī funkcija ir vienlīdzīga kataloga atvēršanai) (245.804 ms).
- Izveidot vienību – jaunas vienības izveide (169.833 ms).

- Rediģēt vienību – izvēlētās vienības rediģēšana (161.443 ms).
- Dzēst vienību – izvēlētas vienības dzēšana (155.103 ms).
- Pārvietot vienību – izvēlētās vienības pārvietošana uz citu kategoriju (149.233 ms).
- Meklēšana – vienību meklēšana pēc visām kategorijām (264.903 ms).

OMA sistēmā pēc katras darbības notiek pilna kataloga lapas atjaunošana, tas nozīmē, ka ir nepieciešams atkārtoti saņemt kategoriju un vienību sarakstu, pēc tam izveidot no jauna kataloga lapu, tas nozīmē, ka katra darbība izpildās par 245.804 milisekundēm ilgāk – kataloga ataunošanas (atvēršanas) laiks. Šīs process aizņem gana daudz laika, tāpēc Scarlett sistēmā visas darbības notiek dinamiski ar JavaScript un AJAX tehnoloģiju palīdzību. Tas nozīmē, ka pārsvarā visas darbības notiek klienta pusē. Pēc katras darbības notiek AJAX pieprasījums, nododot nepieciešamus datus servletā, kurš pēc darbības izpildīšanas atgriezīs, aprakstītu sadaļā "Uzbūve", datu kolekciju, kura ietvēr sevī informāciju no izpildītas MySQL procedūras: id, ret, reload, fraction. Ja "ret" vērtība nebūs vienāda ar "1", tiks parādīts paziņojums par kļūdu un darbība netiks izpildīta. Katalogu funkcijas optimizācija notika sekojoši (iekaviņās tika uzrādīts darbības izpildes laiks Scarlett sistēmā kā arī procentuālā attiecība salīdzinājumā ar OMA):

- Kategorijas izveidošana – lai izveidotu jaunu kategoriju ir nepieciešams ierakstīt jaunas kategorijas nosaukumu, tāpēc var dinamiski iespraust jaunu kategoriju saraksta apakšā, jo mēs zinam kaategorijas nosaukumu un ID no saņemtā datu pieprasījuma (99.340 ms, 33%).
- Kategorijas rediģēšana – kategorijas nosaukums tiek dinamiski atjaunots pēc tās rediģēšanas (90.493 ms, 31%).
- Kategorijas dzēšana – izvēlētā kategorija tiek dinamiski dzēsta , pēc tam notiek pāreja uz pirmo kategoriju no saraksta (89.849 ms + 115.349 ms = 205.198 ms, 70%).
- Kategorijas izvēle – vienību saraksts no izvēlētās kategorijas tiek saņemts Json formātā, pēc tam ar JavaScript palīdzību notiek tabulas ģenerēšana (115.349 ms, 29%).
- Vienības izveidošana – jaunas vienības izveidošanas laikā ir nepieciešams norādīt pie kuras kategorijas attiecās dota vienība, tāpēc kad izveidošana tiek pabeigta, notiek dotas kategorijas izvēle, kurā jau būs iekļauta jaunizveidota vienība. Šajā gadījumā nedrīkst dinamiski iespraust vienību tabulā, jo tabula ietvēr sevī dažādus norēķinu

datus un citu informāciju, kuru ir nepieciešams saņemt no datu bāzes (103.343 ms + 115.349 ms = 218.692 ms, 69%).

- Vienības rediģēšana – šajā gadījumā eksistē divi varianti, jo visi dati ir zināmi:
    - Ja kategorija tika rediģēta – vienība tiek dinamiski dzēsta no saraksta, jo šī vienība tagad atrodās citā kategorijā.
    - Ja kategorija netika rediģēta – vienība tabulā tiek atjaunota dinamiski. Nav nepieciešamības atjaunot sarakstu, jo norēķinu dati jau bija saņemti pirms rediģēšanas.

  101.563 ms, 33%
- Vienības dzēšana – dinamiska vienības dzēšana no tabulas (95.845 ms, 31%).
- Vienības pārvietošana – vienība tiek dinamiski izdzēsta no tabulas, jo tagad vienība atrodās citā kategorijā (86.625 ms, 29%).
- Meklēšana – dota darbība vienāda kategorijas izvēlei, šajā gadījumā netiek padots kategorijas ID, bet tiek padoti meklēšanas dati (124.569 ms, 30%).

Kā var redzēt pēc laika un procentuālas attiecības – Scarlett katalogi strādā daudz ātrāk, nekā OMA katalogi, jo gandrīz visi dati tiek ataunoti dinamiski, kas notiek klienta pusē. Tiek padots un saņemts mazs datu apjoms.

4.4. Tāme

Tāme ir galvenais sistēmas modulis, kur notiek lielāka darba daļa, tāpēc šim modulim visvairāk ir nepieciešami uzlabojumi un optimatizācija. Galvenais āmes uzdevums – sastādīt aktivitāšu sarakstu, katrai aktivitātei ir daudz parametru (kolonnas), piemēram: sākuma datums, beigu datums, komentārs, dažādas summas, valūta un citi. Katra tāmes kolonna tiek rediģēta ar savu veidu – dažos tiek ierakstīta teksta vienība, dažos ir nepieciešams izvēlēties vienību no kataloga un citi.

OMA sistēmas tāmei ir sekojoši trūkumi:

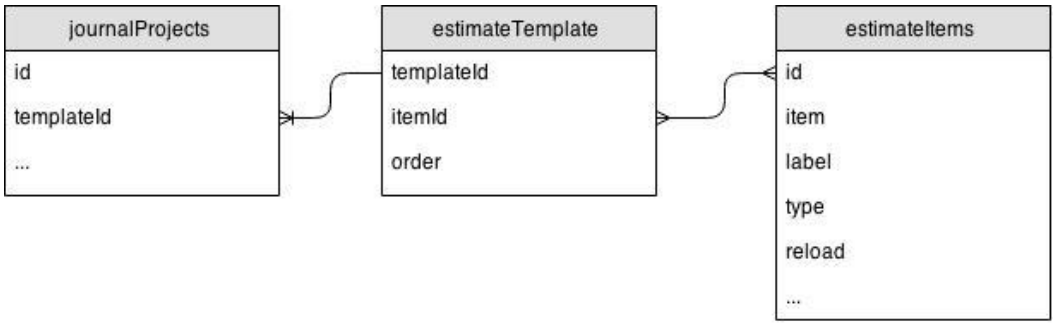
- Visas tabulas kolonnas ir stingri iekodētas PHP kodā, kas ļauj rediģēt kolonnas dažas vietās vai arī pilnībā izdzēst liekus, jo katrā projektā tiek izmantotas ne visas kolonnas, kas tiek dotas, bet tieši tik, cik ir nepieciešams konkrētājā pojektā;
- Pēc katas darbības notiek tabulas atjaunošana, kas palēnina sistēmas darbu.

Scarlett sistēmā visi iepriekšminētie trūkumi tika labotas un novērsti. Lai izveidotu dinamiskas kolonnas, tika izstrādāts šablonu mehānisms. Vispirms ar izveidotas procedūras *get_estimate_template(projectId)* palīdzību ir nepieciešams saņemt katra šablona aprakstu, kurš ietvēr sevī kolonnas nosaukumu, kolonnas tipu, un citu nepieciešamu informāciju. Pēc tam, ar procedūras *get_estimate_data(projectId)* palīdzību ir nepieciešams saņemt datus katrai tabulas kolonnai. Tālāk, skatījumā ar divu ciklu palīdzību notiek iegūto datu pārreizināšana ar šablonu. Katra šablona kolonna satur sevī atslēgu, un atslēga ir kolonnas nosaukums no datu procedūras.

```
...
<c:forEach var="row" items="${data}"> // dāti
    <c:forEach var="col" items="${template}"> // templates dāti
        ...
        ${row[col.item]} // col.item – atslēga dātu kolonnai
        ...
    </c:foreach>
</c:foreach>
...
```

4.4.1. att. Šablona un datu apvienošana - koda fragments

Šablonu mehānismu var realizēt ar dažādiem veidiem, šajā gadījumā datu bāzē viņš ir realizēts sekojoši:



4.4.2. att. Entītiju relāciju diagramma šablonu mehānismam

Tabulā estimateItems glabājās visas pieejamas kolonnas. Katrai kolonnai ir savs tips (type), nosaukums (label), atslēga datiem (item) un citi. Tips norāda ar kuru metodi tiks rediģēta vērtība dotā kolonnā. Tabulā estimateTemplate ir informācija, kuras kolonnas pieder

dotam šablonam, kā arī to kārtība (order). Katrs projekts (journalProjects) iekļauj sevī sev piederešo šablona ID (templateId).

Tabulā estimateItems ir kolonna reload , kura nosaka, vai ir nepieciešams pārlādēt tabulu, ja kāda no vērtībām dotajā kolonnā tiks mainīta no lietotāja puses. Dota metode dod iespēju izbēgt no nepārtrauktas tabulas pārlādēšanas, jo tabulu jāatjauno tikai tādos gadījumos, ja no dotas vērtības ir akarīgas kādas citas tāmes vērtības.

Izveidots mehānisms ļauj pilnībā rediģēt tāmes kolonnas – mainīt vietām – mainīt kārtību, dzēst, pievienot jaunas no saraksta, kas atvieglo lietotājam darbu ar sistēmu, jo viņu neapgrūtina nevajadzīgi dati. Kā arī, dots meānisms ļauj pievienot jaunas kolonnas, pievienojot tikai jaunu vērtīnu datu bāzē.

4.5. Secinājumi

Scarlett sistēmai ir vairākums priekšrocības, salīdzinājumā ar OMA sistēmu:

- DBCP izmantošana ievērojami paātrina savienojuma izveidi ar datu bāzi;
- Java bibliotēkas izmantošana JasperReports atskaišu izveidošanā ļauj ātri ģenerēt dažādu formātu atksaites;
- JavaScript un AJAX izmantošana ļauj momentāli atjaunot datus lappaspusē, bez lappaspuses pārlādēšanas;
- Sistēma ir dinamiska, kas ļauj konfigurēt sistēmu bez izejas koda rediģēšanas.

5. Testēšana un atkļūdošana

Šajā sadaļā autors salīdzinās abu sistēmu galvenas priekšrocības un trūkumus. testēšanas un atkļūdošanas procesā.

5.1. OMA

Parasti PHP koda atkļūdošanā tiek izmanotas iebūvētas *echo(), var_dump(), print_r()* un citas funkcijas, kuras parāda mainīgo vērtības lietotājam uz ekrāna, taču šī metode ir neefektīva, jo lai veiktu pilnu analīzi saņemto datu nepietiek, it īpaši lielām sistemām. Eksistē gana daudz līdzekļu lai atkļūdotu PHP kodu, aplūkosim populārākus no tiem:

- **Xdebug Debugger and Profiler Tool** — PHP paplašinājums [15]. Prasa uzstādīšanu uz servera un konfigurāciju. Var atspoguļot: funkciju izsaukumu steks, atmiņas sadalījums. Iespējas: profilēšana, koda pārklājuma analīze, aizsardzība nobezgalīgas rekursijas, interaktīva skriptu atkļūdošana. Xdebug logu vizuālizēšanai ir nepieciešams uzstādīt papildus programmatūru Webgrind – web-interfeisu uzrakstīts PHP valodā Xdebug profilēšanai, vai arī MacGDB - Mac OS X klients.
- **Xhprof** — PHP paplašinājums no Facebook [16]. Prasa uzstādīšanu uz servera un konfigurāciju. Ļauj savākt katras funkcijas izpildes laiku, atmiņas izmantošanu, gaidīšanas laiku, izsaukumu skaitu un daudz ko citu.;
- **ZendDebug** — PHP paplašinājums ietilpst Zend Studio sastāvā (maksas IDE) [17]. Prasa uzstādīšanu uz servera un konfigurāciju. Ļauj darīt praktiski to pašu, ko dara xdebug, kā arī ir maksas grafiskais interfeis IDE Zend Studio vai Zend Server.
- **APD Advanced PHP debugger** — PHP paplašinājums, vājš xdebug konkurents, bet iekļuj sevī memtrack iespējas, kurš izmēra cik daudz laika aizņem katra programmas fragmenta izstrāde [18]. Slikti integrējās ar IDE, taču ir pieejams konsolinterfeis.
- **FirePHP** — klase, uzrakstīta PHP valodā + FireFox paplašinājums [19]. Dod iespēju sūtīt atkļudošanas paziņojumus Fiebug konsolē ar PHP metožu izsaukumu palīdzību. Visa informācija tiek sūtīta caur X-FirePHP-Data virsrakstiem, tādējādi netraucē citam lapu kontentu virsrakstiem.

Visiem iepriekš aprakstītiem līdzekļiem ir viens ievērojams trūkums, jeb šie līdzekļi strādā līdzīgi kā *var_dump()* un citas funkcijas – kuras parāda mainīgas tipu un vienību

lietotājam. Vienīgā atšķirība – šie līdzekļi nodod informāciju pārlūkprogrammas paplašinājumā vai citā sistēmā, kura apstrādā saņemtus datus, tas nozīmē, ka ir nepieciešams mainīt izejošo sistēmas kodu. Piemērām, Xhprof līdzeklim ir nepieciešams pievienot funkcijas palaišanai (*xhprof_enable()*), datu savākšanai (*save_run()*) un profiletāja apstāšanai (*xhprof_disable()*). APD līdzeklim ir nepieciešams norādīt datu saglabāšanas ceļu ar *apd_set_pprof_trace()* funkcijas palīdzību. Lai nosūtītu datus FirePHP līdzeklim ir nepieciešams izsaukt FB klasi. Daži līdzekļi aizvieto tādu funkciju saturu, kā *var_dump()*, lai funkcijas nosūta datus, neviss izvadītu tos.

5.2. Scarlet

Scarltt sistēma tiek izstrādāta Eclipse porammēšanas vidē, kura atbalsta koda atkļūdošanas funkciju. Tam ir nepieciešams ielikt punktu koda apstādināšanai (breakpoint) uz nepieciešamas līnijas, palaist Tomcat serveri atkļūdošanas režīmā, pēc kā atvērt pašu programmu nepieciešamājā vietā. Aizejot līdz atzīmētai līnijai, koda izpilde apstādinās, pēc kā var pilnvērtīgi analizēt kodu. Katra soļa atkļūdošanas atbalsts ļauj pakāpeniski turpināt koda izpildi lai sameklētu kļūdu, kā arī analizēt dažādu darbību izpildes laiku, lai to optimizēt, un daudz kas cits.

Tākā visa sistēmas biznesa loģika atrodās datu bāzē, ir nepieciešams rūpīgi analizēt MySQL procedūras, funkcijas un trigerus. Visizplātītākais veids ir MySQL funkcijas izveide, kura izvada datus programmas konsolē vai saglabā tos speciāli izveidotajā tabulā. Dots veids strādā tāpāt kā PHP atkļūdošana, taču eksistē atkļūdotāji, kuri ļauj pa soļiem izpildīt kodu, nolikt koda apstāšanas punktus, redzēt izsaukuma stekus un daudz ko citu., piemēram, “dbForge Studio for MySQL” vai “Debugger for MySQL”. Ar šo programmatūras palīdzību var ātri un bez problēmām sameklēt kļūdu kodā vai optimizēt to.

5.3. Secinājumi

Viena no priekšrocībām Java un MySQL koda ir tas, ka kompilatori neļauj nokompilēt kļūdaino kodu. Programmēšanas laikā Eclipse norāda uz visām mainīgām un pieslēgtām bibliotēkām, kuras netiek izmantotas, kas ļauj attīrīt kodu no neizmantojamiem objektiem, tādējādi samazinot atmiņu uz dota koda izpildi. Java un MySQL kodu var analizēt IDE sistēmā izveides laikā, tas ļauj ātri un efektīvi sameklēt kļūdas un skripta vājas vietas.

Secinājumi

Ir uztaisīta jauna sistēma, kura ir ievērojami atrāka, elastīgāka un drošāka, nekā iepriekšeja sistēmas versija.

Darba laikā autors izmantoja gatavus risinājumus, kuri ievērojami palīdzēja sava mērķa reālizēšanā. Izmantota programmatūras arhitektūra iespaidīgi atvieglo visiem izstrādātājiem darbu, jo katrs slānis ir praktiski neakarīgs viens no otra, tas ļauj pilnībā mainīt programmatūras dizainu neaiztiekot citus slāņus. Sistēmas biznesa loģika atrodas datu bāzē, tas ļauj pilnībā mainīt programmatūras grafisku vidi vai pāriet uz citām tehnoloģijām, kuri izvedīs un apstrādās datus, jo darbs ar datiem notiek tikai ar MySQL procedūras palīdzību. JavaScript un AJAX tehnoloģiju izmantošana deva iespēju atjaunot datus dinamiski, neatjaunojot visu timekļa lapu, kas ievērojami palielināja programmatūras veiktspēju un samazināja pieprasījumu daudzumu datu bāzē. Gatavie risinājumi, piemēram, Java bibliotēkas, palīdz ātri un ar mazām pūlēm realizēt nepieciešamo sistēmas funkcionālitāti, piemēram, atskaišu veidošanu vai sistēmas un sistēmas datu drošību.

Informācijas avoti

1. https://en.wikipedia.org/wiki/PHP (sk. 03.04.2015)
2. https://en.wikipedia.org/wiki/MySQL (sk. 03.04.2015)
3. https://en.wikipedia.org/wiki/Java_(programming_language) (sk. 03.04.2015)
4. https://en.wikipedia.org/wiki/HTML (sk. 03.04.2015)
5. https://en.wikipedia.org/wiki/JavaServer_Pages (sk. 03.04.2015)
6. https://en.wikipedia.org/wiki/JavaServer_Pages_Standard_Tag_Library (sk. 03.04.2015)
7. https://en.wikipedia.org/wiki/Salt_(cryptography) (sk. 20.04.2015)
8. https://en.wikipedia.org/wiki/Cross-site_scripting (sk. 22.04.2015)
9. https://en.wikipedia.org/wiki/JSON (sk. 25.04.2015)
10. https://en.wikipedia.org/wiki/Ajax_(programming) (sk. 25.04.2015)
11. https://en.wikipedia.org/wiki/Integrated_development_environment (sk. 30.04.2015)
12. https://lv.wikipedia.org/wiki/MVC (sk. 01.05.2015)
13. https://docs.oracle.com/javaee/6/tutorial/doc/glxgo.html (sk. 04.05.2015)
14. https://commons.apache.org/proper/commons-dbcp/ (sk. 06. 05.2015)
15. http://xdebug.org/ (sk. 08.05.2015)
16. http://xhprof.io/ (sk. 08.05.2015)
17. http://framework.zend.com/manual/1.12/en/zend.debug.html (sk. 08.05.2015)
18. http://php.net/manual/en/book.apd.php (sk. 09.05.2015)
19. https://addons.mozilla.org/ru/firefox/addon/firephp/ (sk. 09.05.2015)

Pielikumi

Pielikumos ir atrodāmi Scarlett sistēmas koda fragmeti.

1. pielikums. FieldsSetter klase

```
package atom.scarlett.db;

import java.math.BigDecimal;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class FieldsSetter {

   public static void setSQLDate(PreparedStatement statement, int parameterIndex, Date date) throws
SQLException {
      if(date == null)
         statement.setNull(parameterIndex, java.sql.Types.DATE);
      else
         statement.setDate(parameterIndex, date);
   }

   public static void setSQLString(PreparedStatement statement, int parameterIndex, String value) throws
SQLException {
      if(value == null || "".equals(value))
         statement.setNull(parameterIndex, java.sql.Types.VARCHAR);
      else
         statement.setString(parameterIndex, value.trim());
   }

   public static void setSQLLong(PreparedStatement statement, int parameterIndex, Long value) throws
SQLException {
      if(value == null)
         statement.setNull(parameterIndex, java.sql.Types.BIGINT);
      else
         statement.setLong(parameterIndex, value);
   }

   public static void setSQLInt(PreparedStatement statement, int parameterIndex, Integer value) throws
SQLException {
      if(value == null)
```

```
        statement.setNull(parameterIndex, java.sql.Types.INTEGER);
      else
        statement.setInt(parameterIndex, value);
    }

    public static void setSQLBigDecimal(PreparedStatement statement, int parameterIndex, BigDecimal
value) throws SQLException {
      if(value == null)
        statement.setNull(parameterIndex, java.sql.Types.DECIMAL);
      else
        statement.setBigDecimal(parameterIndex, value);
    }

}
```

2. pielikums. Atskaites ģenerēšanas funkcija

```
public byte[] generateXLSX(String jasperFile, Map<String, Object> parameters) {

    byte[] result = null;
    JasperPrint jasperPrint = null;
    JRXlsxExporter exporter = null;
    ByteArrayOutputStream outputStream = null;

    try {
        connection = getConnection();
        jasperPrint = JasperFillManager.fillReport(jasperFile, parameters, connection);
        exporter = new JRXlsxExporter();
        outputStream = new ByteArrayOutputStream();

        exporter.setExporterInput(new SimpleExporterInput(jasperPrint));
        exporter.setExporterOutput(new SimpleOutputStreamExporterOutput(outputStream));

        SimpleXlsxReportConfiguration configuration = new SimpleXlsxReportConfiguration();
        configuration.setDetectCellType(true);
        configuration.setCollapseRowSpan(false);
        exporter.setConfiguration(configuration);

        exporter.exportReport();
        result = outputStream.toByteArray();

    } catch (JRException | ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    } finally {
        DbUtil.close(connection);
    }
    return result;
}
```

3. pielikums. Datu lasīšana no datu bāzes

```
public List<Fee> feeList(long activity, int type){

    List<Fee> list = new ArrayList<Fee>();
    Fee fee = null;

    try {

      connection = getConnection();

      String query = "CALL get_activity_fees(?, ?)";

      statement = connection.prepareStatement(query);
      FieldsSetter.setSQLLong(statement, 1, activity);
      FieldsSetter.setSQLInt(statement, 2, type);

      ResultSet rs = statement.executeQuery();

      while (rs.next()){

        fee = new Fee();
        fee.setId(FieldsReader.getLong(rs, "feeid"));
        fee.setName(FieldsReader.getString(rs, "description"));
        fee.setChecked(FieldsReader.getInt(rs, "checkbox") == 1);
        list.add(fee);

      }

      DbUtil.close(rs);

    } catch (SQLException e) {
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    } finally {
      DbUtil.close(statement);
      DbUtil.close(connection);
    }

    return list;

  }
```