|  |
| --- |
| Hanoi university of science and technology  Document Managing System |
| E-Book Dungeon Report |
| Group:  *Trịnh Huy Hoàng ID: 20101586*  *Tăng Thế Cường ID: 20101226*  *Lê Hoàng Phương ID: 20101999* |
|  |
| **Instructor: Mrs. Nguyễn Thị Thu Trang and Mr. Trịnh Thành Trung** |
|  |
|  |
| **Hanoi, 05/2014** |

Table of Contents

[I. Work Assignment 2](#_Toc388474012)

[1. Trịnh Huy Hoàng: 2](#_Toc388474013)

[2. Tăng Thế Cường: 2](#_Toc388474014)

[3. Lê Hoàng Phương: 2](#_Toc388474015)

[II. Introduction 2](#_Toc388474016)

[1. Requirement 2](#_Toc388474017)

[2. Main functions of website 4](#_Toc388474018)

[2.1. Authentication 4](#_Toc388474019)

[2.2. Document Management 4](#_Toc388474020)

[2.3. Group Management 6](#_Toc388474021)

[2.4. User Management 7](#_Toc388474022)

[3. Difficulties: 8](#_Toc388474023)

[III. Technical solution 8](#_Toc388474024)

[1. Framework 8](#_Toc388474025)

[1.1. Phalcon PHP 8](#_Toc388474026)

[1.2. Application of Phalcon PHP 9](#_Toc388474027)

[2. Advanced: 9](#_Toc388474028)

[2.1. Ajax searching 9](#_Toc388474029)

[2.2. PHP Excel 9](#_Toc388474030)

[IV. Result 9](#_Toc388474031)

[1. Database 9](#_Toc388474032)

[2. Interfaces 10](#_Toc388474033)

[3. Evaluation 12](#_Toc388474034)

[3.1. Advantages 12](#_Toc388474035)

[3.2. Disadvantages 13](#_Toc388474036)

[V. Conclusion 13](#_Toc388474037)

[Reference 13](#_Toc388474038)

# Work Assignment

### Trịnh Huy Hoàng:

* Design database structure
* Implementing user server-side use-cases
* Making slide presentation

### Tăng Thế Cường:

* Implement admin server-side use-cases
* Testing
* Write report

### Lê Hoàng Phương:

* Implement client-side front-end (HTML, CSS, javascript,etc.)
* Resolve technological issues.

# Introduction

## Requirement

Build a website that allow user to:

* Upload document with format doc/docx, xls/xlsx, ppt/pptx, pdf, html, etc. to website.
* Create new document by HTML and be able to export to HTML or docx.
* Edit document attributes: name, description, author, year, etc.
* Edit document visibility: Only Me, Member Only, Public.
* Put document into one or many groups
* Search for document by attribute(s).
* View document directly on browser.
* Download document

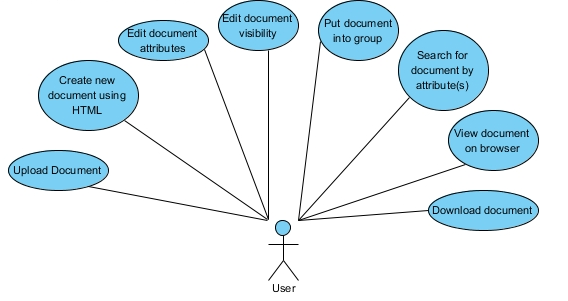


Figure 1: Use-Case Diagram of User according to requirement

Also, website allows administrator to:

* Manage user
* Manage document group. Be able to create child group.
* Download all documents belong to a group.

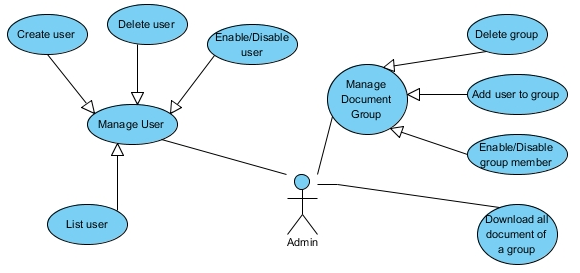


Figure 2: Use-Case Diagram of Administrator according to requirement

## Main functions of website

### Authentication

Authentication process allow user and admin to log in/out of website.

#### Login

Get POST request from view and then use sha1 to encrypt password. After that, check in database for user\_name, password\_hash and status = ‘active’. If true then register in session and user (or admin) can login.

public function startAction() {

**if** ($this->request->isPost()) {

$username = $this->request->getPost('login\_username', 'alphanum');

$password = $this->request->getPost('login\_password');

$password = sha1($password);

$user = **Users::**findFirst("user\_name='$username' AND password\_hash='$password'");

**if** ($user != false && $user->status == 'active') {

$group = **Groups::**findFirst("owner\_id='$user->user\_id'");

**if** ($group == false) {

$this->\_registerSession($user);

**return** $this->redirect("contents/index");

} **else** {

$this->\_registerSession($user, $group->group\_id);

**return** $this->redirect("lobby/index");

}

//**return** $this->forward('contents/index');

} **else** **if** ($user != false && $user->status != 'active')

$this->flash->error('Your account has not been activated!');

**else** **if** ($user == false)

$this->flash->error('Wrong username/password');

}

**return** $this->forward('session/index');

}

#### Logout

Clean the session and return to the index page

public function endAction() {

$this->session->destroy();

**return** $this->redirect("index");

}

#### Register

Receive information from form and do the logic

public function registerAction() {

$validate = TRUE;

$request = $this->request;

**if** ($request->isPost()) {

**if** (isset($\_POST['register\_submit'])) {

$accountType = $request->getPost('accounttype');

$username = $request->getPost('username', 'alphanum');

$email = $request->getPost('email', 'email');

$password = $request->getPost('password');

$firstname = $request->getPost('firstname');

$lastname = $request->getPost('lastname');

$mobilephone = $request->getPost('mobilephone');

$verification\_hash = md5(uniqid(rand(), **1**));

$user = **new** Users();

$user->user\_name = $username;

$user->password\_hash = sha1($password);

$user->verification\_hash = $verification\_hash;

$user->email = $email;

$user->first\_name = $firstname;

$user->last\_name = $lastname;

$user->phone = $mobilephone;

$user->expiration = **new** Phalcon\Db\RawValue('now()');

$user->extra\_time = **0**;

$user->status = 'active';

$user->maximum = **new** Phalcon\Db\RawValue('default');

$user->used = **new** Phalcon\Db\RawValue('default');

;

$user->available = **new** Phalcon\Db\RawValue('default');

//$user->created\_at = **new** Phalcon\Db\RawValue('now()');

try {

**if** ($user->save() == false) {

$validate = false;

$message = $user->getMessages();

$this->flash->error((string) $message[**0**]);

// Logging

**foreach** ($user->getMessages() as $m) {

$this->logger->error($m);

}

} **else** {

**if** ($accountType == "group") {

// Create group

$group = **new** Groups();

$group->owner\_id = $user->user\_id;

$group->group\_name = $request->getPost('groupname');

**if** ($group->save() == false) {

$validate = FALSE;

// Delete user member

**if** ($user->delete() == false) {

**foreach** ($user->getMessages() as $m) {

$this->logger->error('Register error: ' . $m);

}

}

$message = $group->getMessages();

$this->flash->error((string) $message[**0**]);

}

}

// Send mail **if** validation is passed

**if** ($validate == true) {

$this->flash->success('Thanks for sign-up! You can now log in!');

}

}

} catch (\Exception $ex) {

$this->logger->error("Register exception: " . $ex->getMessage());

}

}

}

**return** $this->forward('session/index');

}

### Document Management

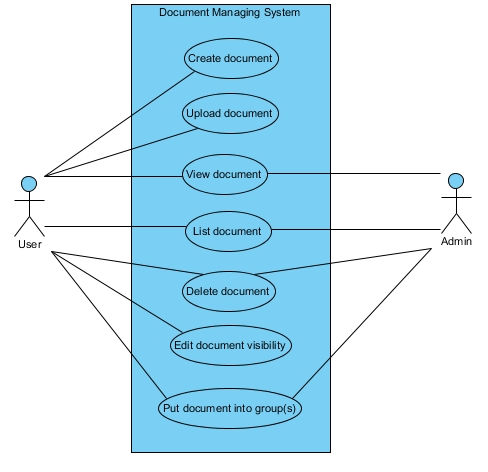


Figure 3: Use-Case Diagram of all document managing activities

#### Create document

Forward user to “create” page, which has a mini editor (enough for regular usage). After an user complete the document, click Save and new document will be saved to database and physical storage. This use TinyMCE library as editor and Htmltodocx library to create docx file

public function createAction()

{

}

public function saveAction()

{

**if** (!$this->request->isPost()) {

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "create"

));

}

$auth = $this->session->get('auth');

$userId = $auth['id'];

$userName = $auth['name'];

$dirPath = $this->config->upload\_dir . $userName . '/';

// Load the files we need:

require\_once 'phpword/PHPWord.php';

require\_once 'simplehtmldom/simple\_html\_dom.php';

require\_once 'htmltodocx\_converter/h2d\_htmlconverter.php';

require\_once 'example\_files/styles.inc';

// Functions to support this example.

require\_once 'documentation/support\_functions.inc';

// HTML fragment we want to parse:

//$html = file\_get\_contents('example\_files/example\_html.html');

$html = $this->request->getPost("content");

// $html = file\_get\_contents('test/table.html');

// New Word Document:

$phpword\_object = **new** PHPWord();

$section = $phpword\_object->createSection();

// HTML Dom object:

$html\_dom = **new** simple\_html\_dom();

$html\_dom->load('<html><body>' . $html . '</body></html>');

// Note, we needed to nest the html in a couple of dummy elements.

// Create the dom array of elements which we are going to work on:

$html\_dom\_array = $html\_dom->find('html', **0**)->children();

// We need this **for** setting base\_root **and** base\_path in the initial\_state array

// (below). We are using a function here (derived from Drupal) to create these

// paths automatically - you may want to **do** something different in your

// implementation. This function is in the included file

// documentation/support\_functions.inc.

$paths = htmltodocx\_paths();

// Provide some initial settings:

$initial\_state = array(

// Required parameters:

'phpword\_object' => &$phpword\_object, // Must be passed by reference.

// 'base\_root' => 'http://test.local', // Required **for** link elements - change it to your domain.

// 'base\_path' => '/htmltodocx/documentation/', // Path from base\_root to whatever url your links are relative to.

'base\_root' => $paths['base\_root'],

'base\_path' => $paths['base\_path'],

// Optional parameters - showing the defaults **if** you don't set anything:

'current\_style' => array('size' => '**11**'), // The PHPWord style on the top element - may be inherited by descendent elements.

'parents' => array(0 => 'body'), // Our parent is body.

'list\_depth' => 0, // This is the current depth of any current list.

'context' => 'section', // Possible values - section, footer or header.

'pseudo\_list' => TRUE, // NOTE: Word lists not yet supported (TRUE is the only option at present).

'pseudo\_list\_indicator\_font\_name' => 'Wingdings', // Bullet indicator font.

'pseudo\_list\_indicator\_font\_size' => '**7**', // Bullet indicator size.

'pseudo\_list\_indicator\_character' => 'l ', // Gives a circle bullet point with wingdings.

'table\_allowed' => TRUE, // Note, if you are adding this html into a PHPWord table you should set this to FALSE: tables cannot be nested in PHPWord.

'treat\_div\_as\_paragraph' => TRUE, // If set to TRUE, each new div will trigger a new line in the Word document.

// Optional - no default:

'style\_sheet' => htmltodocx\_styles\_example(), // This is an array (the "style sheet") - returned by htmltodocx\_styles\_example() here (in styles.inc) - see this function for an example of how to construct this array.

);

// Convert the HTML and put it into the PHPWord object

htmltodocx\_insert\_html($section, $html\_dom\_array[0]->nodes, $initial\_state);

// Clear the HTML dom object:

$html\_dom->clear();

unset($html\_dom);

// Save File

$h2d\_file\_uri = tempnam('', 'htd');

$objWriter = PHPWord\_IOFactory::createWriter($phpword\_object, 'Word2007');

$objWriter->save($h2d\_file\_uri);

// Check user space available

$availSpace = 0;

$user = Users::findFirst("user\_id = $userId");

if ($user) {

$availSpace = intval($user->available);

}

// Create new content in contents table

$content = new Contents();

$content->owner\_id = $userId;

$content->file\_type = 'application/vnd.openxmlformats-officedoc';

$content->uploaded = new Phalcon\Db\RawValue('now()');

$content->created = new Phalcon\Db\RawValue('now()');

$content->content\_name = 'Untitled' . rand(1, 1000);

$content->information = $content->content\_name . '.docx';

$content->path = $dirPath . $content->information;

$content->content\_size = filesize($h2d\_file\_uri);

$content->content\_extension = 'docx';

$content->status = 'private';

$user->used += $content->content\_size;

$user->available -= $content->content\_size;

if ($user->available < 0) {

$this->flash->error("Available space is not enough");

return $this->forward("contents/index");

}

// Create new directory if not exists

if (!is\_dir($dirPath)) {

if (!mkdir($dirPath)) {

$err = error\_get\_last();

$errors[] = $err['message'];

$this->logger->error($err['message']);

$this->flash->error($this->config->message->error->make\_dir);

}

}

if (!$content->save()) {

$messages = $content->getMessages();

// Logging

foreach ($messages as $m) {

$errors[] = (string)$m;

$this->logger->error("Save file upload error: " . (string)$m);

}

}

//var\_dump($dirPath . $contentName); die;

rename($h2d\_file\_uri, ($dirPath . $content->information));

// // Download the file:

// header('Content-Description: File Transfer');

// header('Content-Type: application/octet-stream');

// header('Content-Disposition: attachment; filename=' . $h2d\_file\_uri . '.docx');

// header('Content-Transfer-Encoding: binary');

// header('Expires: **0**');

// header('Cache-Control: must-revalidate, post-check=**0**, pre-check=**0**');

// header('Pragma: public');

// header('Content-Length: ' . filesize($h2d\_file\_uri));

// ob\_clean();

// flush();

// $status = readfile($h2d\_file\_uri);

// unlink($h2d\_file\_uri);

// exit;

$this->flash->success("Your document is created successfully");

**return** $this->forward("contents/index");

}

#### Upload document

Allow users to select multiple documents from their computer and upload to system. The documents info will be saved to database and the physical files will be saved in /upload/username folder

public function uploadAction()

{

$auth = $this->session->get('auth');

$userId = $auth['id'];

$userName = $auth['name'];

$dirPath = $this->config->upload\_dir . $userName;

try {

// Check **if** the user has uploaded files

**if** ($this->request->hasFiles() == true) {

$errors = array();

// Obtain transaction manager

$manager = $this->transactions;

// Request a transaction

$transaction = $manager->get();

// Print the real file names **and** sizes

**foreach** ($this->request->getUploadedFiles() as $file) {

$fileInfo = $file->getName();

$fileSize = intval($file->getSize());

// Check user space available

$availSpace = **0**;

$user = **Users::**findFirst("user\_id = $userId");

**if** ($user) {

$availSpace = intval($user->available);

}

**if** ($fileSize <= $availSpace) {

// Check **if** content has already existed

$content = **Contents::**findFirst("owner\_id = $userId AND information = '$fileInfo'");

**if** (!$content) {

// Create **new** one

$content = **new** Contents();

$content->status = "private";

}

// Assign transaction

$content->setTransaction($transaction);

// Update content info

$content->owner\_id = $userId;

$content->file\_type = $file->getType();

$content->path = $dirPath . '/' . $file->getName();

$content->information = $file->getName();

$content->content\_size = $fileSize;

$info = **new** SplFileInfo($file->getName());

$extension = $info->getExtension();

$contentName = $info->getBasename("." . $extension);

$content->content\_name = $contentName;

$content->content\_extension = $extension;

$content->uploaded = **new** Phalcon\Db\RawValue('now()');

$content->created = **new** Phalcon\Db\RawValue('now()');

// Create **new** directory **if** **not** exists

**if** (!is\_dir($dirPath)) {

**if** (!mkdir($dirPath)) {

$err = error\_get\_last();

$errors[] = $err['message'];

$this->logger->error($err['message']);

$this->flash->error($this->config->message->error->make\_dir);

}

}

//Move the file into the application

**if** (!$file->moveTo($dirPath . '/' . $file->getName())) {

// Logging

$err = error\_get\_last();

$errors[] = $err['message'];

$this->logger->error($err['message']);

} **else** {

// Save to content table

**if** (!$content->save()) {

// Rollback

$messages = $content->getMessages();

$transaction->rollback((string)$messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$errors[] = (string)$m;

$this->logger->error("Save file upload error: " . (string)$m);

}

} **else** {

// Update user account space info

$user->used += $fileSize;

$user->available -= $fileSize;

$user->setTransaction($transaction);

**if** ($user->save() == FALSE) {

$messages = $user->getMessages();

// Rollback

$transaction->rollback((string)$messages[**0**]);

**foreach** ($messages as $m) {

$errors[] = (string)$m;

$this->logger->error("Save account space info error: " . (string)$m);

}

}

// Success transaction

$transaction->commit();

$this->flash->success($this->config->message->success->upload);

}

}

} **else** {

// Not enough space **for** uploading

$errors[] = "Available space is not enough";

}

// Prepare response to client

$content = "<response>";

**if** (!empty($errors)) {

$content .= "<errors>";

**foreach** ($errors as $error) {

$content .= "<error>";

$content .= $error;

$content .= "</error>";

}

$content .= "</errors>";

} **else** {

$content .= "<success>";

$content .= **1**;

$content .= "</success>";

}

$content .= "</response>";

$response = **new** Phalcon\Http\Response();

$response->setHeader('Content-Type', 'text/xml');

$response->setContent($content);

**return** $response;

}

}

} catch (Exception $ex) {

$this->logger->error('Upload exception: ' . $ex->getMessage());

}

}

#### View document

At this moment, our system can only view docx, doc, xls, xlsx, pdf files. With xls and xlsx, we use PHPExcel library. With doc and docx, we use custom script. With pdf, it is “browser support”

public function showAction($content\_id = **0**)

{

$auth = $this->session->get('auth');

$userId = $auth['id'];

$userName = $auth['name'];

try {

$content = **Contents::**findFirst($content\_id);

**if** ($content) {

**if** ($content->status == 'private' && $content->owner\_id != $userId) {

$this->flash->error("You don't have the right to view this document!");

**return** $this->forward("contents/index");

}

**if** ($content->content\_extension == 'docx') {

$filePath = $content->path;

**if** (file\_exists($filePath)) {

$content = $this->read\_file\_docx($filePath);

**if** ($content !== false) {

echo '<div style="width: 90%; background-color: #ffffff">';

echo nl2br($content);

echo '</div>';

} **else** {

echo 'Couldn\'t the file. Please check that file.';

}

}

} elseif ($content->content\_extension == 'doc') {

$DocumentPath = $content->path;

$word = **new** COM("word.application") **or** die("Unable to instantiate application object");

$wordDocument = **new** COM("word.document") **or** die("Unable to instantiate document object");

$word->Visible = **0**;

$wordDocument = $word->Documents->Open($DocumentPath);

$HTMLPath = substr\_replace($DocumentPath, 'html', -**3**, **3**);

$wordDocument->SaveAs($HTMLPath, **3**);

$wordDocument = null;

$word->Quit();

$word = null;

echo '<div style="width: 90%; background-color: #ffffff">';

echo readfile($HTMLPath);

echo '</div>';

unlink($HTMLPath);

} elseif ($content->content\_extension == 'xlsx' || $content->content\_extension == 'xls') {

$filePath = $content->path;

include 'Classes/PHPExcel/IOFactory.php';

$inputFileName = $filePath;

// Read your Excel workbook

try {

$inputFileType = **PHPExcel\_IOFactory::**identify($inputFileName);

$objReader = **PHPExcel\_IOFactory::**createReader($inputFileType);

$objPHPExcel = $objReader->load($inputFileName);

} catch (Exception $e) {

die('Error loading file "' . pathinfo($inputFileName, PATHINFO\_BASENAME)

. '": ' . $e->getMessage());

}

// Get worksheet dimensions

$sheet = $objPHPExcel->getSheet(**0**);

$highestRow = $sheet->getHighestRow();

$highestColumn = $sheet->getHighestColumn();

echo '<div class="table-responsive">';

echo '<table class="table table-blue table-bordered table-hover">';

echo '<thead>';

$row = **1**;

$rowData = $sheet->rangeToArray('A' . $row . ':' . $highestColumn . $row,

NULL, TRUE, FALSE);

echo '<tr>';

**foreach**($rowData[**0**] as $k=>$v) {

?>

<th><?php echo $v ?></th>

<?php

}

echo '</tr>';

echo '</thead>';

echo '<tbody>';

**for** ($row = **2**; $row <= $highestRow; $row++) {

$rowData = $sheet->rangeToArray('A' . $row . ':' . $highestColumn . $row,

NULL, TRUE, FALSE);

echo '<tr>';

**foreach**($rowData[**0**] as $k=>$v) {

?>

<td><?php echo $v ?></td>

<?php

}

echo '</tr>';

}

echo '</tbody>';

echo '</table>';

echo '</div>';

} **else** {

$filePath = $content->path;

**if** (file\_exists($filePath)) {

header('Content-Description: File Transfer');

header('Content-Type: ' . $content->file\_type);

header('Content-Disposition: inline; filename="' . basename($content->information) . '"');

header('Content-Transfer-Encoding: binary');

header('Expires: 0');

header('Cache-Control: must-revalidate, post-check=0, pre-check=0');

header('Pragma: public');

header('Content-Length: ' . filesize($filePath));

ob\_clean();

flush();

readfile($filePath);

exit();

}

}

} **else** {

$this->flash->error("Content was not found");

**return** $this->forward('contents/index');

}

} catch (Exception $ex) {

$this->logger->error('Show content exception: ' . $ex->getMessage());

**return** $this->forward('contents/index');

}

}

#### List document

In this function, we select all properly documents from database, and then print all in a table

public function indexAction()

{

/\*

\* Assets resource

\*/

// Adding style sheet

$this->assets

->addCss('upload\_plugin/css/jquery.fileupload.css')

->addCss('upload\_plugin/css/jquery.fileupload-ui.css');

// Adding javascript

$this->assets

->addJs('upload\_plugin/js/vendor/jquery.ui.widget.js')

->addJs('upload\_plugin/js/tmpl.min.js')

->addJs('upload\_plugin/js/load-image.min.js')

->addJs('upload\_plugin/js/canvas-to-blob.min.js')

->addJs('upload\_plugin/js/jquery.blueimp-gallery.min.js')

->addJs('upload\_plugin/js/jquery.iframe-transport.js')

->addJs('upload\_plugin/js/jquery.fileupload.js')

->addJs('upload\_plugin/js/jquery.fileupload-process.js')

->addJs('upload\_plugin/js/jquery.fileupload-image.js')

->addJs('upload\_plugin/js/jquery.fileupload-audio.js')

->addJs('upload\_plugin/js/jquery.fileupload-video.js')

->addJs('upload\_plugin/js/jquery.fileupload-validate.js')

->addJs('upload\_plugin/js/jquery.fileupload-ui.js')

->addJs('upload\_plugin/js/main.js');

$auth = $this->session->get('auth');

$userId = $auth['id'];

$userName = $auth['name'];

// Account space

$maxSpace = $usedSpace = $availableSpace = $percent = **0**;

$user = **Users::**findFirst("user\_id = $userId");

**if** ($user) {

$maxSpace = intval($user->maximum);

$usedSpace = intval($user->used);

$availableSpace = intval($user->available);

$percent = ($maxSpace != **0**) ? round((($usedSpace / $maxSpace) \* **100**), **2**) : **0**;

}

$this->view->setVar("maxSpace", $maxSpace);

$this->view->setVar("usedSpace", $usedSpace);

$this->view->setVar("availableSpace", $availableSpace);

$this->view->setVar("percent", $percent);

$dirPath = $this->config->upload\_dir . $userName;

try {

$numberPage = **1**;

$query = **Criteria::**fromInput($this->di, "Contents", array("owner\_id" => $userId));

$this->persistent->parameters = $query->getParams();

**if** (!$this->request->isPost()) {

$numberPage = $this->request->getQuery("page", "int");

**if** ($numberPage <= **0**) {

$numberPage = **1**;

}

}

$parameters = $this->persistent->parameters;

**if** (!$parameters) {

$parameters = array();

}

$parameters["order"] = "content\_id";

$contents = **Contents::**find($parameters);

$paginator = **new** Paginator(array(

"data" => $contents,

"limit" => **10**,

"page" => $numberPage

));

$this->view->page = $paginator->getPaginate();

} catch (Exception $ex) {

$this->logger->error('Session exception: ' . $ex->getMessage());

}

}

#### Delete document

In this function, we can select one or more document to delete. Documents info will be deleted from database and also on physical storage. Transaction must be used in this function to keep integrity

private function \_deleteContent($content\_id = **0**)

{

$auth = $this->session->get('auth');

$userId = $auth['id'];

$result = TRUE;

try {

$user = **Users::**findFirstByuser\_id($userId);

**if** (!$user) {

$this->logger->error("Delete content of user\_id: $userId does not exits");

**return** $result;

}

$content = **Contents::**findFirstBycontent\_id($content\_id);

**if** (!$content) {

$this->logger->error("Delete content: $content\_id does not exits");

**return** $result;

}

$contentSize = $content->content\_size;

// Increase user account available space

$user->used -= $contentSize;

$user->available = (intval($user->maximum) - intval($user->used));

// Obtain transaction manager

$manager = $this->transactions;

// Request a transaction

$transaction = $manager->get();

// Delete shared contents

$sharedContents = **SharedContents::**find("content\_id=$content->content\_id");

**if** (count($sharedContents) > **0**) {

**foreach** ($sharedContents as $sharedContent) {

$sharedContent->setTransaction($transaction);

**if** ($sharedContent->delete() == false) {

$result = FALSE;

// Rollback the transaction

**foreach** ($sharedContent->getMesasges() as $m) {

$transaction->rollback($m->getMessage());

}

}

}

}

// Delete content

$content->setTransaction($transaction);

**if** (!$content->delete()) {

$result = FALSE;

**foreach** ($content->getMessages() as $message) {

$transaction->rollback($message->getMessage());

}

}

// Delete physical

$filepath = $content->path;

$success = true;

**if** (is\_file($filepath))

$success = unlink($filepath);

**else**

$success = FALSE;

**if** ($success == FALSE) {

$result = FALSE;

$err = error\_get\_last();

$transaction->rollback($err['message']);

}

// Save account space info

$user->setTransaction($transaction);

**if** ($user->save() == FALSE) {

$result = FALSE;

// Rollback

**foreach** ($user->getMessages() as $message) {

$transaction->rollback($message->getMessage());

}

}

// Commit the transaction

$transaction->commit();

} catch (TxFailed $ex) {

$this->logger->error("Delete content exception: " . $ex->getMessage());

}

**return** $result;

}

private function \_deleteAction($contentList)

{

$result = TRUE;

try {

**if** (count($contentList) > **0**) {

**foreach** ($contentList as $content\_id) {

$result = $this->\_deleteContent($content\_id) ? $result : FALSE;

}

}

} catch (Exception $ex) {

$this->logger->error("Delete multiple contents exception: " . $ex->getMessage());

}

**if** ($result == FALSE)

$this->flash->error($this->config->message->error->delete\_content);

**else**

$this->flash->success("Content was deleted successfully");

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index",

));

}

#### Edit document visibility

In this function, if a document is private (“Only Me”), it can be shared to other groups or make public. If a document is public, it can be made back to private

public function editAction($content\_id = **0**)

{

// Get session

$auth = $this->session->get('auth');

$userId = $auth['id'];

$groupId = $auth['group\_id'];

try {

$content = **Contents::**findFirstBycontent\_id($content\_id);

**if** (!$content) {

$this->flash->error("content was not found");

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

$sql = 'SELECT Groups.\*, Users.user\_name FROM Groups '

. 'JOIN Users on Groups.owner\_id = Users.user\_id ';

$query = $this->modelsManager->createQuery($sql);

$groups = $query->execute(array('groupId' => $groupId));

// Getting member list

$memGroup = array();

// Check **if** current account is a group

**if** ($groupId > **0**) {

$memGroup[] = $groupId;

}

$groupMember = **GroupMember::**find("member\_id = $userId AND member\_status = 'active'");

**foreach** ($groupMember as $grp) {

$memGroup[] = $grp->group\_id;

}

$this->view->memList = $memGroup;

$this->view->content = $content;

$this->view->content\_id = $content->content\_id;

$this->view->groups = $groups;

// Getting shared group list

$shared\_list = array();

$sharedGroup = **SharedContents::**find("content\_id = $content\_id AND content\_status='shared'");

**foreach** ($sharedGroup as $group) {

$shared\_list[] = $group->group\_id;

}

$this->view->shared\_list = $shared\_list;

} catch (Exception $ex) {

$this->logger->error('Edit content exception: ' . $ex->getMessage());

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

}

public function makecontentpublicAction($content\_id = **0**)

{

try {

$content = **Contents::**findFirstBycontent\_id($content\_id);

**if** (!$content) {

$this->flash->error("content was not found");

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

$content->status = 'public';

**if** (!$content->save()) {

$messages = $content->getMessages();

$this->flash->error((string)$messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Make content public error: " . (string)$m);

}

**return** $this->forward("contents/index");

}

$this->flash->success("This content is now public");

**return** $this->forward("contents/index");

} catch (Exception $e) {

$this->logger->error('Make content public exception: ' . $e->getMessage());

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

}

public function makecontentprivateAction($content\_id = **0**)

{

try {

$content = **Contents::**findFirstBycontent\_id($content\_id);

**if** (!$content) {

$this->flash->error("content was not found");

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

$content->status = 'private';

**if** (!$content->save()) {

$messages = $content->getMessages();

$this->flash->error((string)$messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Make content private error: " . (string)$m);

}

**return** $this->forward("contents/index");

}

$this->flash->success("This content is now private");

**return** $this->forward("contents/index");

} catch (Exception $e) {

$this->logger->error('Make content public exception: ' . $e->getMessage());

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

}

#### Put document into group(s)

This function is a part of 2.2.6, which let user select groups to share documents

public function updateAction()

{

**if** (!$this->request->isPost()) {

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "index"

));

}

$auth = $this->session->get('auth');

$userId = $auth['id'];

$groupId = $auth['group\_id'];

$content\_id = $this->request->getPost("content\_id");

$groupList = $this->request->getPost("group");

try {

$isShared = FALSE;

$isError = FALSE;

$content = **Contents::**findFirst("content\_id = $content\_id");

**if** (is\_array($groupList)) {

**foreach** ($groupList as $group\_id) {

$shared = **SharedContents::**findFirst("content\_id = $content\_id AND group\_id = $group\_id");

**if** (!$shared) {

$shared = **new** SharedContents();

$shared->content\_id = $content\_id;

$shared->group\_id = $group\_id;

}

$shared->content\_status = 'shared';

**if** ($shared->save() == FALSE) {

$isError = TRUE;

$messages = $shared->getMessages();

$this->flash->error($messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Share content error: " . (string)$m);

}

} **else** {

$isShared = true;

}

}

}

// Change content status

$content->status = ($isShared == TRUE) ? 'shared' : 'private';

**if** ($content->save() == false) {

$isError = TRUE;

// Logging

**foreach** ($content->getMessages() as $m) {

$this->logger->error("Share content error: " . (string)$m);

}

}

$groupIds = **GroupMember::**find("member\_id = $userId");

/\*

\* Getting array of **not** sharing group

\*/

$ungroupIds = array();

**foreach** ($groupIds as $group) {

$ungroupIds[] = $group->group\_id;

}

// Check **if** current account is a group

**if** ($groupId > **0**) {

$ungroupIds[] = $groupId;

}

// Checking unshared group

**foreach** ($ungroupIds as $ungroup\_id) {

**if** (!is\_array($groupList) || in\_array($ungroup\_id, $groupList) == FALSE) {

$shared = **SharedContents::**findFirst("content\_id = $content\_id AND group\_id = $ungroup\_id");

// Change status to private

**if** ($shared) {

$shared->content\_status = 'private';

**if** ($shared->save() == false) {

$isError = TRUE;

// Logging

**foreach** ($shared->getMessages() as $m) {

$this->logger->error("Private content error: " . (string)$m);

}

}

}

}

}

**if** (!$isError)

$this->flash->success($this->config->message->success->update\_content);

**else**

$this->flash->error($this->config->message->error->update\_content);

} catch (Exception $ex) {

$this->logger->error("Share content exception: " . $ex->getMessage());

}

**return** $this->dispatcher->forward(array(

"controller" => "contents",

"action" => "edit",

"params" => array($content\_id)

));

}

### Group Management

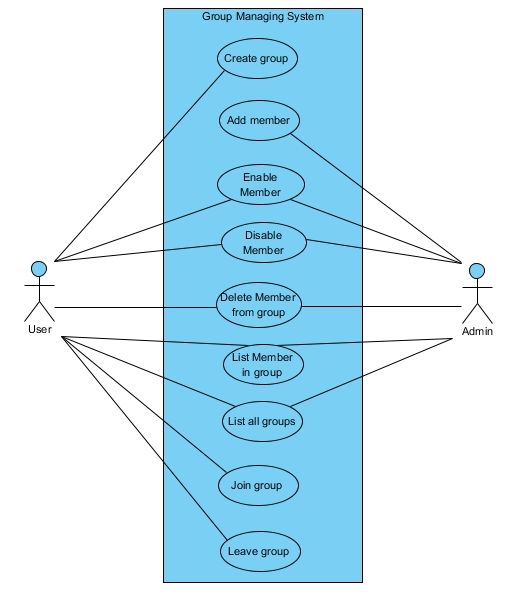


Figure 4: Use-Case Diagram of all group managing activities

#### Create group

When register, if an user select Group in account type, his group will be created at that moment. Otherwise, if user select Individual, then after login, he can create group in account tab

/\*

\* Handle group creating request

\*/ **else** **if** ($request->hasPost('create\_group')) {

$groupName = $request->getPost('group\_name');

$group = **Groups::**findFirst("group\_name = '$groupName'");

// Check **if** group has already existed

**if** ($group) {

$this->flash->error($this->config->message->error->group\_name);

}

// Create **new** group

**else** {

$group = **new** Groups();

$group->owner\_id = $userId;

$group->group\_name = $groupName;

**if** ($group->save() == false) {

$messages = $group->getMessages();

$this->flash->error((string) $messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Update to group admin error: " . (string) $m);

}

} **else** {

// Update session

$this->session->set('auth', array(

'id' => $auth['id'],

'name' => $auth['name'],

'group\_id' => $group->group\_id

));

$this->flash->success($this->config->message->success->upgrade\_group);

}

}

}

#### Add member

Only admin can do this function. Admin can add members to an existing group

public function addmemberAction($group\_id) {

try {

$group = **Groups::**findFirstBygroup\_id($group\_id);

**if** (!$group) {

$this->flash->error("Group was not found");

**return** $this->forward("groups/index");

}

$numberPage = **1**;

**if** (!$this->request->isPost()) {

$numberPage = $this->request->getQuery("page", "int");

**if** ($numberPage <= **0**) {

$numberPage = **1**;

}

}

$members = **Users::**find("status = 'active'");

$array1 = array();

$sql = "SELECT GroupMember.member\_id FROM GroupMember WHERE GroupMember.group\_id = :group\_id:";

$query = $this->modelsManager->createQuery($sql);

$members\_in\_group = $query->execute(array('group\_id' => $group\_id));

**foreach** ($members\_in\_group as $member\_in\_group) {

$array1[] = $member\_in\_group->member\_id;

}

$sql = "SELECT Groups.owner\_id FROM Groups WHERE Groups.group\_id = :group\_id:";

$query = $this->modelsManager->createQuery($sql);

$owner\_of\_group = $query->execute(array('group\_id' => $group\_id));

**foreach** ($owner\_of\_group as $owner) {

$array1[] = $owner->owner\_id;

}

**if** (count($members) == **0**) {

$this->flash->notice("The search did not find any member");

}

$paginator = **new** Phalcon\Paginator\Adapter\Model(array(

"data" => $members,

"limit" => **10**,

"page" => $numberPage

));

$page = $paginator->getPaginate ();

$this->view->setVar ( "group", $group );

$this->view->setVar ( "page", $page );

$this->view->setVar("members", $members);

$this->view->setVar("array1", $array1);

} catch (Exception $e) {

$this->logger->error ( "Add Members Errors : " . $e->getMessage () );

}

}

public function saveusertogroupAction($user\_id, $group\_id) {

$foundMember = **GroupMember::**findFirst("group\_id = $group\_id & member\_id = $user\_id");

**if** ($foundMember) {

$this->flash->error("This user is already in group!");

**return** $this->forward("groups/addmember");

}

$member = **new** GroupMember();

$member->group\_id = $group\_id;

$member->member\_id = $user\_id;

$member->member\_status = "active";

**if** (!$member->save()) {

$this->flash->error("Failed to add user to group");

**return** $this->forward("groups/addmember");

} **else** {

$this->flash->success("Add user to group successfully");

**return** $this->redirect("groups/manage/$group\_id");

}

}

#### Enable member

Group Admin can enable a member (make member status ‘active’) after an user want to join group or if member status is ‘rejected’

public function enableAction($user\_id) {

try {

$user = **Users::**findFirst ( "user\_id = $user\_id" );

**if** (! $user) {

$this->flash->error ( "User was not found" );

**return** $this->forward ( "account/index" );

}

**if** ($user->status == "inactive") {

$user->status = "active";

**if** (! $user->save ()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( ( string ) $message );

}

**return** $this->forward ( "account/index" );

}

$this->flash->success ( "User status was changed to 'active' successfully" );

**return** $this->forward ( "account/index" );

} **else** **if** ($user->status == "active") {

$this->flash->error ( "User is already 'active'" );

**return** $this->forward ( "account/index" );

} **else** {

$this->flash->error ( "User is pending" );

**return** $this->forward ( "account/index" );

}

} catch ( Exception $e ) {

$this->logger->error ( "Enable Errors : " . $e->getMessage () );

}

}

#### Disable member

Group Admin can disable a member (make member status ‘rejected). That member is still in group but can not see any documents from group anymore. If user want to be enabled back, he must send join request again (and of course, Group Admin accept)

public function disableAction($user\_id) {

try {

$user = **Users::**findFirst ( "user\_id = $user\_id" );

**if** (! $user) {

$this->flash->error ( "User was not found" );

**return** $this->forward ( "account/index" );

}

**if** ($user->status == "active") {

$user->status = "inactive";

**if** (! $user->save ()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( ( string ) $message );

}

**return** $this->forward ( "account/index" );

}

$this->flash->success ( "User status was changed to 'inactive' successfully" );

**return** $this->forward ( "account/index" );

} **else** **if** ($user->status == "inactive") {

$this->flash->error ( "User is already 'inactive'" );

**return** $this->forward ( "account/index" );

} **else** {

$this->flash->error ( "User is pending" );

**return** $this->forward ( "account/index" );

}

} catch ( Exception $e ) {

$this->logger->error ( "Disable Errors : " . $e->getMessage () );

}

}

#### Delete member from group

Group Admin can delete a member from his group. If user want to be back in group, he must send join request again (and of course, Group Admin accept)

private function \_deleteUser($user\_id = **0**) {

$result = true;

try {

$user = **Users::**findFirstByuser\_id($user\_id);

**if** (!$user) {

$this->logger->error("Delete user: $user\_id does not exits");

**return** $result;

}

// Obtain transaction manager

$manager = $this->transactions;

// Request a transaction

$transaction = $manager->get();

// Delete group that user own

$group = **Groups::**findFirst ( "owner\_id = $user\_id" );

**if** ($group) {

$tmp\_group\_id = $group->group\_id;

$group\_members = **GroupMember::**find ( "group\_id = $tmp\_group\_id" );

**if** ($group\_members) {

**foreach** ( $group\_members as $group\_member ) {

$group\_member->setTransaction($transaction);

**if** (! $group\_member->delete ()) {

$result = false;

**foreach** ( $group\_member->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

$group->setTransaction($transaction);

**if** (! $group->delete ()) {

**foreach** ( $group->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

// Delete contents belong to user

$contents = **Contents::**find ( "owner\_id = $user\_id" );

**if** ($contents) {

**foreach** ( $contents as $content ) {

$shared\_contents = **SharedContents::**find ( "content\_id = $content->content\_id" );

**if** ($shared\_contents) {

**foreach** ( $shared\_contents as $shared\_content ) {

$shared\_content->setTransaction($transaction);

**if** (! $shared\_content->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $shared\_content->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete conversion contents

$conversions = **Conversions::**find("content\_id=$content->content\_id");

**if** ($conversions) {

**foreach** ($conversions as $conversion) {

$conversion->setTransaction($transaction);

**if** (! $conversion->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $conversion->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete physical

$filepath = $content->path;

$success = true;

**if** (is\_file($filepath))

$success = unlink($filepath);

**else**

$success = FALSE;

**if** ($success == FALSE) {

$result = FALSE;

$err = error\_get\_last();

$transaction->rollback($err['message']);

}

$content->setTransaction($transaction);

**if** (! $content->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $content->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete the contents the removed\_contents table

$removedContents = **RemovedContents::**find("owner\_id = $user->user\_id");

**if** ($removedContents) {

**foreach** ($removedContents as $removedContent) {

// Delete physical

$removedContent\_filepath = $removedContent->path;

$success = true;

**if** (is\_file($removedContent\_filepath))

$success = unlink($removedContent\_filepath);

**else**

$success = FALSE;

**if** ($success == FALSE) {

$result = FALSE;

$err = error\_get\_last();

$transaction->rollback($err['message']);

}

$removedContent->setTransaction($transaction);

**if** (! $removedContent->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $conversion->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

$user->setTransaction($transaction);

**if** (! $user->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $user->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

// Commit the transaction

$transaction->commit();

} catch (TxFailed $ex) {

$this->logger->error("Delete user exception: " . $ex->getMessage());

}

**return** $result;

}

public function deleteAction() {

$deleteItems = array ();

**if** (!$this->request->isPost()) {

**return** $this->forward("account/index");

}

try {

// Check **if** request has made with POST

**if** ($this->request->isPost () == true) {

// Access POST data

$deleteItems = $this->request->getPost ( "item" );

$result = TRUE;

**if** (count($deleteItems) > **0**) {

// Delete all of them

**foreach** ( $deleteItems as $user\_id ) {

$result = $this->\_deleteUser($user\_id) ? $result : FALSE;

}

}

}

} catch ( Exception $e ) {

$this->logger->error ( "Delete User Errors : " . $e->getMessage () );

}

**if** ($result == FALSE)

$this->flash->error("Delete contents failed");

**else**

$this->flash->success("User was deleted successfully");

**return** $this->forward("account/index");

}

#### List member in group

Group Admin can list all members of his group to manage (enable, disable or delete)

public function indexAction() {

$auth = $this->session->get('auth');

$groupId = $auth['group\_id'];

// Check valid group account

**if** ($groupId <= **0**) **return** $this->redirect ("index");

// Set group name

$groupName = **Groups::**findFirst(array(

"conditions" => "group\_id = ?0",

"columns" => "group\_name",

"bind" => array(**0** => $groupId)

));

**if** ($groupName) $this->view->setVar ("groupName", $groupName);

$numberPage = **1**;

**if** (!$this->request->isPost()) {

$numberPage = $this->request->getQuery("page", "int");

**if** ($numberPage <= **0**) {

$numberPage = **1**;

}

}

try {

$sql = "SELECT GroupMember.\*, Users.user\_name "

. "FROM GroupMember, Users "

. "WHERE GroupMember.member\_id = Users.user\_id "

. "AND GroupMember.group\_id = :groupId: ";

$query = $this->modelsManager->createQuery($sql);

$contents = $query->execute(array('groupId' => $groupId));

$paginator = **new** Paginator(array(

"data" => $contents,

"limit" => **10**,

"page" => $numberPage

));

$this->view->page = $paginator->getPaginate();

} catch (Exception $exc) {

$this->logger->error("Group admin exception: " . $exc->getMessage());

}

}

#### List all group

Only admin has the ability to list all groups in system

public function indexAction() {

// Get session info

$admin\_auth = $this->session->get ( 'admin\_auth' );

// Query the active user

$admin = **Admin::**findFirst ( $admin\_auth ['id'] );

**if** ($admin == false) {

$this->\_forward ( 'index/index' );

}

$numberPage = **1**;

**if** ($this->request->isPost ()) {

$query = **Criteria::**fromInput ( $this->di, "Groups", $\_POST );

$this->persistent->searchParams = $query->getParams ();

} **else** {

$numberPage = $this->request->getQuery ( "page", "int" );

**if** ($numberPage <= **0**) {

$numberPage = **1**;

}

}

$sql = "SELECT Groups.group\_id, Groups.owner\_id, Groups.group\_name, Users.user\_name "

. "FROM Groups, Users "

. "WHERE Groups.owner\_id = Users.user\_id ";

$query = $this->modelsManager->createQuery($sql);

$groups = $query->execute();

$paginator = **new** Phalcon\Paginator\Adapter\Model ( array (

"data" => $groups,

"limit" => **10**,

"page" => $numberPage

) );

$page = $paginator->getPaginate ();

$this->view->setVar ( "page", $page );

$this->view->setVar ( "groups", $groups );

}

#### Join group

Any user can send join request in order to join a group. This request will be send to Group Admin of that group, and user need to wait for Group Admin to accept

public function joinAction($group\_id) {

$auth = $this->session->get('auth');

$userId = $auth['id'];

$userName = $auth['name'];

$group = **Groups::**findFirstBygroup\_id($group\_id);

$owner = **Users::**findFirst("user\_id = $group->owner\_id");

$token = md5(uniqid(rand(), **1**));

**if** ($group && $owner) {

$member = **GroupMember::**findFirst(array("group\_id = $group\_id AND member\_id = $userId"));

**if** (!$member) {

$groupMeber = **new** GroupMember();

$groupMeber->group\_id = $group->group\_id;

$groupMeber->member\_id = $userId;

$groupMeber->member\_status = 'pending';

$groupMeber->action\_token = $token;

**if** ($groupMeber->save() == false) {

$messages = $groupMeber->getMessages();

$this->flash->error($messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Join group error: " . $m);

}

} **else** {

$this->flash->success("Joining request has been sent. Wait the approval of the admin!");

}

} elseif ($member && $member->member\_status == 'rejected') {

$member->member\_status = 'pending';

**if** ($member->save() == false) {

$messages = $member->getMessages();

$this->flash->error($messages[**0**]);

// Logging

**foreach** ($messages as $m) {

$this->logger->error("Join group error: " . $m);

}

} **else** {

$this->flash->success("Joining request has been sent. Wait the approval of the admin!");

}

}

} **else** {

$this->flash->error("This group doesn't exist!");

// Logging

$this->logger->error("Join group: $group\_id doesn't exist");

}

**return** $this->forward("groups/index");

}

#### Leave group

Member of a group can leave that group at any time

public function leaveAction($group\_id) {

$auth = $this->session->get('auth');

$userId = $auth['id'];

// Delete shared contents

$phql = "SELECT SharedContents.\* FROM SharedContents "

. "JOIN Contents on Contents.content\_id = SharedContents.content\_id "

. "WHERE SharedContents.group\_id = ?0 AND Contents.owner\_id = ?1";

$manager = $this->modelsManager;

$sharedContents = $manager->executeQuery($phql, array(**0** => $group\_id, **1** => $userId));

$member = **GroupMember::**findFirst("group\_id = $group\_id AND member\_id = $userId");

try {

// Create a transaction manager

$manager = **new** txManager();

// Request a transaction

$transaction = $manager->get();

// Deleting content

**foreach** ($sharedContents as $sharedContent) {

$content = **Contents::**findFirst("content\_id = $sharedContent->content\_id");

$sharedContent->setTransaction($transaction);

**if** ($sharedContent->delete() == false) {

// Something went wrong, rollback transaction

**foreach** ($sharedContent->getMessages() as $message) {

$transaction->rollback($message->getMessage());

}

}

// Check to change content status

$content->setTransaction($transaction);

$tmpContent = **SharedContents::**find("content\_id = $sharedContent->content\_id AND content\_status='shared'");

**if** (count($tmpContent) == **1**) {

$content->status = 'private';

**if** ($content->save() == false) {

// Rollback

**foreach** ($content->getMessages() as $message) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete group member

$member->setTransaction($transaction);

**if** ($member->delete() == FALSE) {

**foreach** ($member->getMessages() as $message) {

$transaction->rollback($message->getMessage());

}

}

// Everything goes fine, commit transaction

$transaction->commit();

} catch (Failed $ex) {

$this->logger->error("Leave group exception: " . $ex->getMessage());

}

**return** $this->forward("groups/index");

}

### User Management

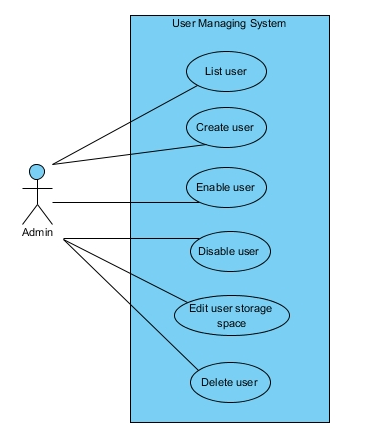


Figure 5: Use-Case Diagram of all user managing activities

#### List user

Admin can list all users in the system to manage (enable, disable, edit user storage and delete user)

public function indexAction() {

// Get session info

$admin\_auth = $this->session->get ( 'admin\_auth' );

// Query the active user

$admin = **Admin::**findFirst ( $admin\_auth ['id'] );

**if** ($admin == false) {

**return** $this->forward ( 'index/index' );

}

$numberPage = **1**;

**if** ($this->request->isPost ()) {

$query = **Criteria::**fromInput ( $this->di, "Users", $\_POST );

$this->persistent->searchParams = $query->getParams ();

} **else** {

$numberPage = $this->request->getQuery ( "page", "int" );

**if** ($numberPage <= **0**) {

$numberPage = **1**;

}

}

$users = **Users::**find ();

$paginator = **new** Phalcon\Paginator\Adapter\Model ( array (

"data" => $users,

"limit" => **10**,

"page" => $numberPage

) );

$page = $paginator->getPaginate ();

$this->view->setVar ( "page", $page );

$this->view->setVar ( "users", $users );

}

#### Create user

Admin can create new user

public function createAction() {

try {

**if** (! $this->request->isPost ()) {

**return** $this->forward ( "account/index" );

}

$password = $this->request->getPost ( "password" );

$password\_confirm = $this->request->getPost ( "password\_confirm" );

**if** ($password != $password\_confirm) {

$this->flash->error ( "Passwords do not match" );

**return** $this->forward ( "account/new" );

} **else** {

$user = **new** Users ();

$user->user\_name = $this->request->getPost ( "user\_name", "striptags" );

$user->password\_hash = sha1 ( $this->request->getPost ( "password" ) );

$user->verification\_hash = $user->password\_hash;

$user->email = $this->request->getPost ( "email", "email" );

$user->first\_name = $this->request->getPost ( "first\_name", "striptags" );

$user->last\_name = $this->request->getPost ( "last\_name", "striptags" );

$user->phone = $this->request->getPost ( "phone", "striptags" );

$user->expiration = "0";

$user->extra\_time = **0**;

$user->status = "active";

**if** (! $user->save ()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( "Create User Error : " . ( string ) $message );

}

**return** $this->forward ( "account/new" );

}

$this->flash->success ( "User was created successfully" );

**return** $this->forward ( "account/index" );

}

} catch ( Exception $e ) {

$this->logger->error ( "Create User Errors : " . $e->getMessage () );

}

}

#### Enable user

Admin can enable user (make user status ‘active’)

public function enableAction($user\_id) {

try {

$user = **Users::**findFirst ( "user\_id = $user\_id" );

**if** (! $user) {

$this->flash->error ( "User was not found" );

**return** $this->forward ( "account/index" );

}

**if** ($user->status == "inactive") {

$user->status = "active";

**if** (! $user->save ()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( ( string ) $message );

}

**return** $this->forward ( "account/index" );

}

$this->flash->success ( "User status was changed to 'active' successfully" );

**return** $this->forward ( "account/index" );

} **else** **if** ($user->status == "active") {

$this->flash->error ( "User is already 'active'" );

**return** $this->forward ( "account/index" );

} **else** {

$this->flash->error ( "User is pending" );

**return** $this->forward ( "account/index" );

}

} catch ( Exception $e ) {

$this->logger->error ( "Enable Errors : " . $e->getMessage () );

}

}

#### Disable user

Admin can disable user (make user status ‘inactive’)

public function disableAction($user\_id) {

try {

$user = **Users::**findFirst ( "user\_id = $user\_id" );

**if** (! $user) {

$this->flash->error ( "User was not found" );

**return** $this->forward ( "account/index" );

}

**if** ($user->status == "active") {

$user->status = "inactive";

**if** (! $user->save ()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( ( string ) $message );

}

**return** $this->forward ( "account/index" );

}

$this->flash->success ( "User status was changed to 'inactive' successfully" );

**return** $this->forward ( "account/index" );

} **else** **if** ($user->status == "inactive") {

$this->flash->error ( "User is already 'inactive'" );

**return** $this->forward ( "account/index" );

} **else** {

$this->flash->error ( "User is pending" );

**return** $this->forward ( "account/index" );

}

} catch ( Exception $e ) {

$this->logger->error ( "Disable Errors : " . $e->getMessage () );

}

}

#### Edit user storage space

Admin can edit maximum user storage

public function changemaximumAction($user\_id) {

try {

$user = **Users::**findFirstByuser\_id($user\_id);

**if** (!$user) {

$this->flash->error ( "User was not found" );

**return** $this->forward ( "account/index" );

}

$this->view->setVar("user", $user);

} catch (Exception $e) {

$this->logger->error("Change Maximum Space error : " . $e->getMessage());

}

}

public function updatemaximumAction() {

**if** (!$this->request->isPost()) {

**return** $this->forward("account/index");

}

try {

$user\_id = $this->request->getPost ("user\_id", "int");

$user = **Users::**findFirstByuser\_id($user\_id);

**if** (!$user) {

$this->flash->error("User was not found");

**return** $this->forward("account/index");

}

$maximum = $this->request->getPost("new\_maximum", "int");

$maximum = $maximum \* **1024** \* **1024**;

**if** ($user->used <= $maximum) {

$user->maximum = $maximum;

$user->available = $maximum - $user->used;

} **else** {

$this->flash->error("User can not used more than maximum space");

**return** $this->redirect("account/changemaximum/$user\_id");

}

**if** (!$user->save()) {

**foreach** ( $user->getMessages () as $message ) {

$this->flash->error ( ( string ) $message );

$this->logger->error ( "Update Maximum Space Errors : " . ( string ) $message );

}

**return** $this->redirect ( "account/changemaximum/$user\_id" );

}

$this->flash->success ( "User maximum space was updated successfully" );

**return** $this->redirect ( "account/changemaximum/$user\_id" );

} catch (Exception $e) {

$this->logger->error ( "Update Maximum Space Errors : " . $e->getMessage () );

}

}

#### Delete user

Admin can delete users

private function \_deleteUser($user\_id = **0**) {

$result = true;

try {

$user = **Users::**findFirstByuser\_id($user\_id);

**if** (!$user) {

$this->logger->error("Delete user: $user\_id does not exits");

**return** $result;

}

// Obtain transaction manager

$manager = $this->transactions;

// Request a transaction

$transaction = $manager->get();

// Delete group that user own

$group = **Groups::**findFirst ( "owner\_id = $user\_id" );

**if** ($group) {

$tmp\_group\_id = $group->group\_id;

$group\_members = **GroupMember::**find ( "group\_id = $tmp\_group\_id" );

**if** ($group\_members) {

**foreach** ( $group\_members as $group\_member ) {

$group\_member->setTransaction($transaction);

**if** (! $group\_member->delete ()) {

$result = false;

**foreach** ( $group\_member->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

$group->setTransaction($transaction);

**if** (! $group->delete ()) {

**foreach** ( $group->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

// Delete contents belong to user

$contents = **Contents::**find ( "owner\_id = $user\_id" );

**if** ($contents) {

**foreach** ( $contents as $content ) {

$shared\_contents = **SharedContents::**find ( "content\_id = $content->content\_id" );

**if** ($shared\_contents) {

**foreach** ( $shared\_contents as $shared\_content ) {

$shared\_content->setTransaction($transaction);

**if** (! $shared\_content->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $shared\_content->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete conversion contents

$conversions = **Conversions::**find("content\_id=$content->content\_id");

**if** ($conversions) {

**foreach** ($conversions as $conversion) {

$conversion->setTransaction($transaction);

**if** (! $conversion->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $conversion->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete physical

$filepath = $content->path;

$success = true;

**if** (is\_file($filepath))

$success = unlink($filepath);

**else**

$success = FALSE;

**if** ($success == FALSE) {

$result = FALSE;

$err = error\_get\_last();

$transaction->rollback($err['message']);

}

$content->setTransaction($transaction);

**if** (! $content->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $content->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

// Delete the contents the removed\_contents table

$removedContents = **RemovedContents::**find("owner\_id = $user->user\_id");

**if** ($removedContents) {

**foreach** ($removedContents as $removedContent) {

// Delete physical

$removedContent\_filepath = $removedContent->path;

$success = true;

**if** (is\_file($removedContent\_filepath))

$success = unlink($removedContent\_filepath);

**else**

$success = FALSE;

**if** ($success == FALSE) {

$result = FALSE;

$err = error\_get\_last();

$transaction->rollback($err['message']);

}

$removedContent->setTransaction($transaction);

**if** (! $removedContent->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $conversion->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

}

}

$user->setTransaction($transaction);

**if** (! $user->delete ()) {

$result = false;

// Rollback the transaction

**foreach** ( $user->getMessages () as $message ) {

$transaction->rollback($message->getMessage());

}

}

// Commit the transaction

$transaction->commit();

} catch (TxFailed $ex) {

$this->logger->error("Delete user exception: " . $ex->getMessage());

}

**return** $result;

}

public function deleteAction() {

$deleteItems = array ();

**if** (!$this->request->isPost()) {

**return** $this->forward("account/index");

}

try {

// Check **if** request has made with POST

**if** ($this->request->isPost () == true) {

// Access POST data

$deleteItems = $this->request->getPost ( "item" );

$result = TRUE;

**if** (count($deleteItems) > **0**) {

// Delete all of them

**foreach** ( $deleteItems as $user\_id ) {

$result = $this->\_deleteUser($user\_id) ? $result : FALSE;

}

}

}

} catch ( Exception $e ) {

$this->logger->error ( "Delete User Errors : " . $e->getMessage () );

}

**if** ($result == FALSE)

$this->flash->error("Delete contents failed");

**else**

$this->flash->success("User was deleted successfully");

**return** $this->forward("account/index");

}

## Difficulties:

The use-case "Download all documents in a group" prove to be difficult, due to the way documents are stored on server.

For admin to be able to download multiple documents at a time, server must first compress those documents into archive. However each document stored in different directory (correspond to its owner), so compress them is difficult. We were unable to solve this problem in time, so this use-case is left unfinished.

Viewing document on browser is much more difficult than expected. Images, plaintext, pdf, etc. can be render by browser by itself, but special format such as doc, xls, ppt,, etc. require helps of additional libraries to read. We only able to find PHP Excel to help read xls/xlsx file, while other format is almost completely unsupported.

Render them is also difficult, because those format have different markup from HTML. Some special effect were hard to reproduce using HTML, so we render those document in plaintext form.

# Technical solution

## Framework

### Phalcon PHP

### Application of Phalcon PHP

## Bootstrap

## Security

## Advanced:

### Ajax searching

### PHP Excel

# Result

## Database

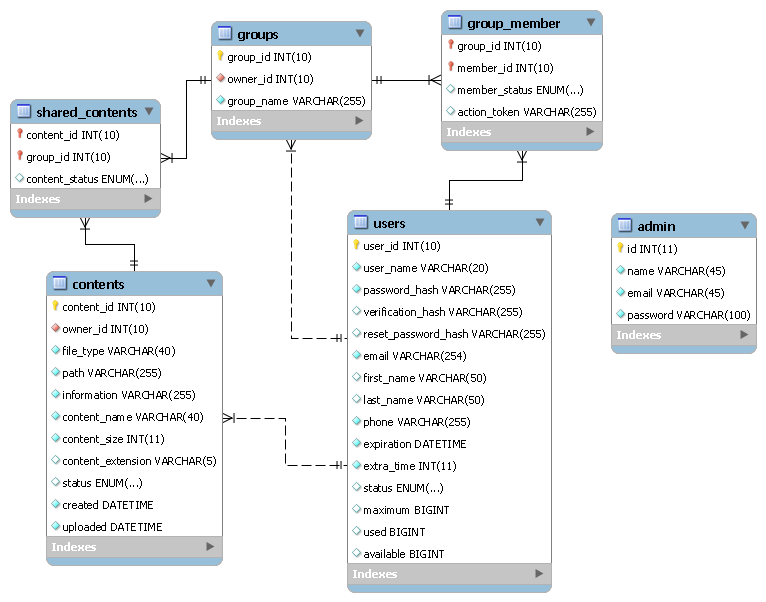
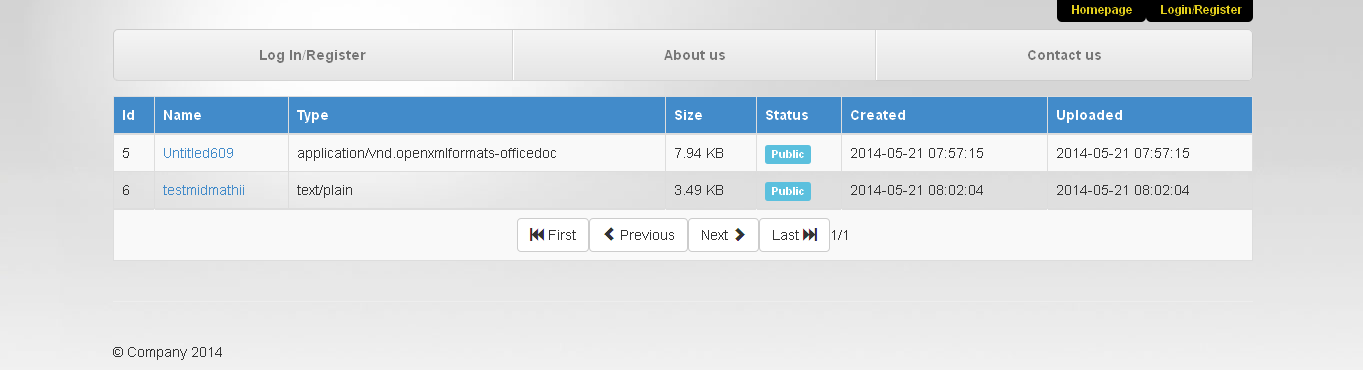
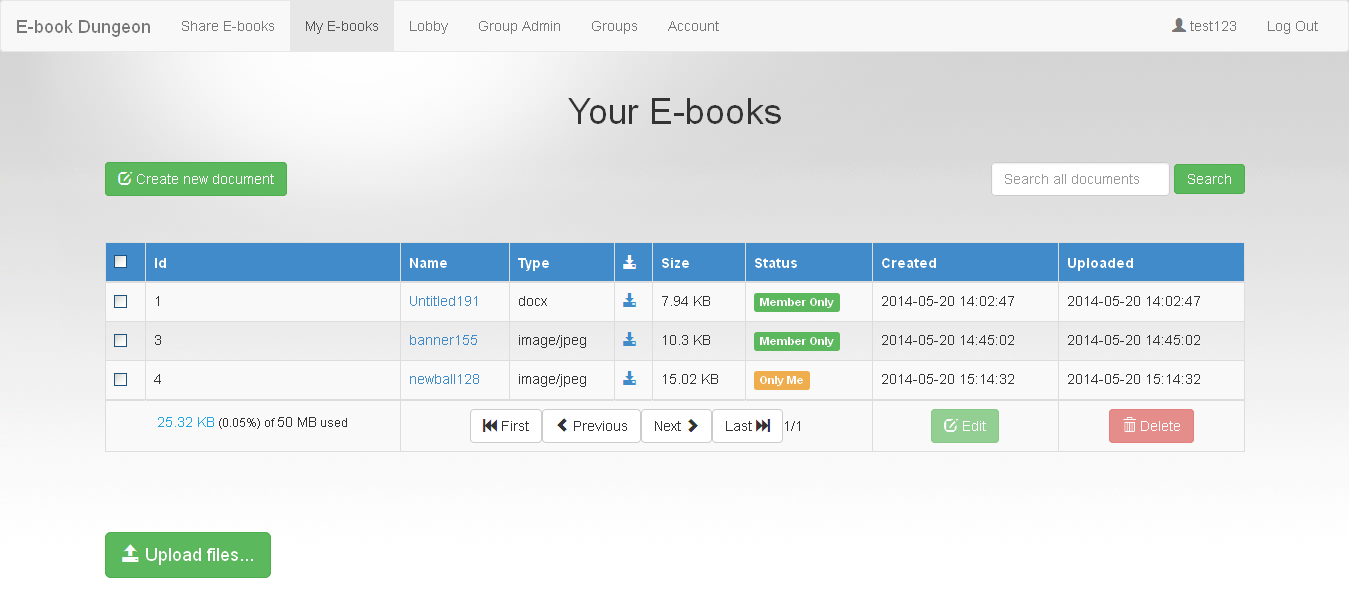


Figure 6 : Database structure

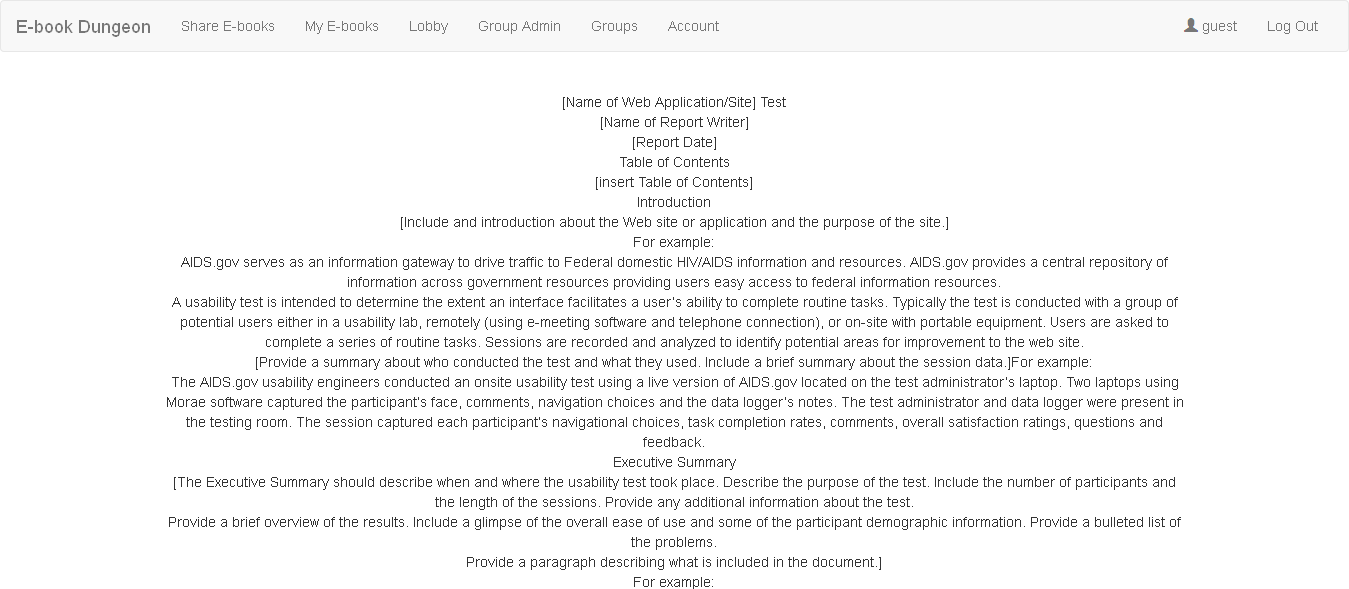
## Interfaces



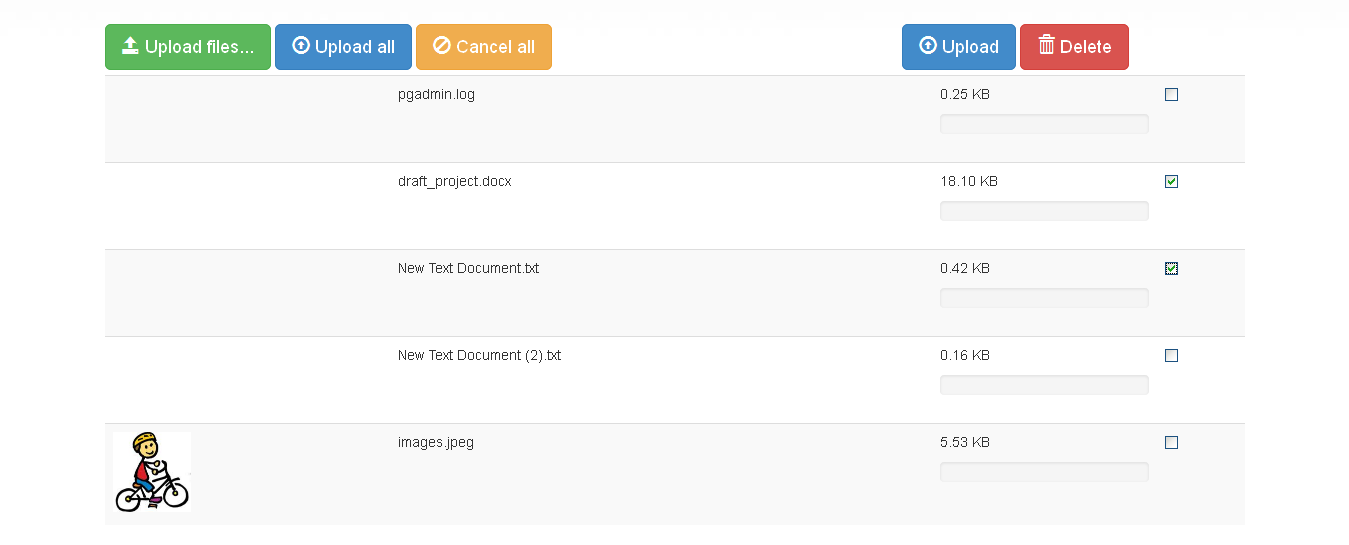
Picture 1: Index page



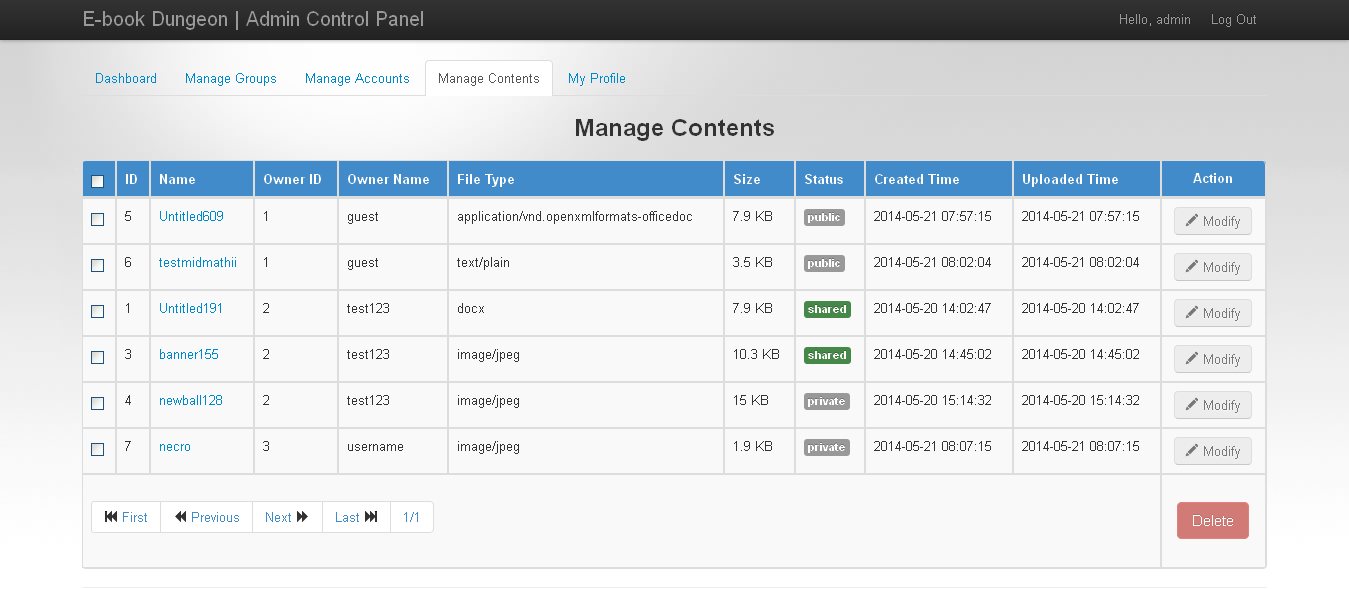
Picture 2: Personal Ebook page



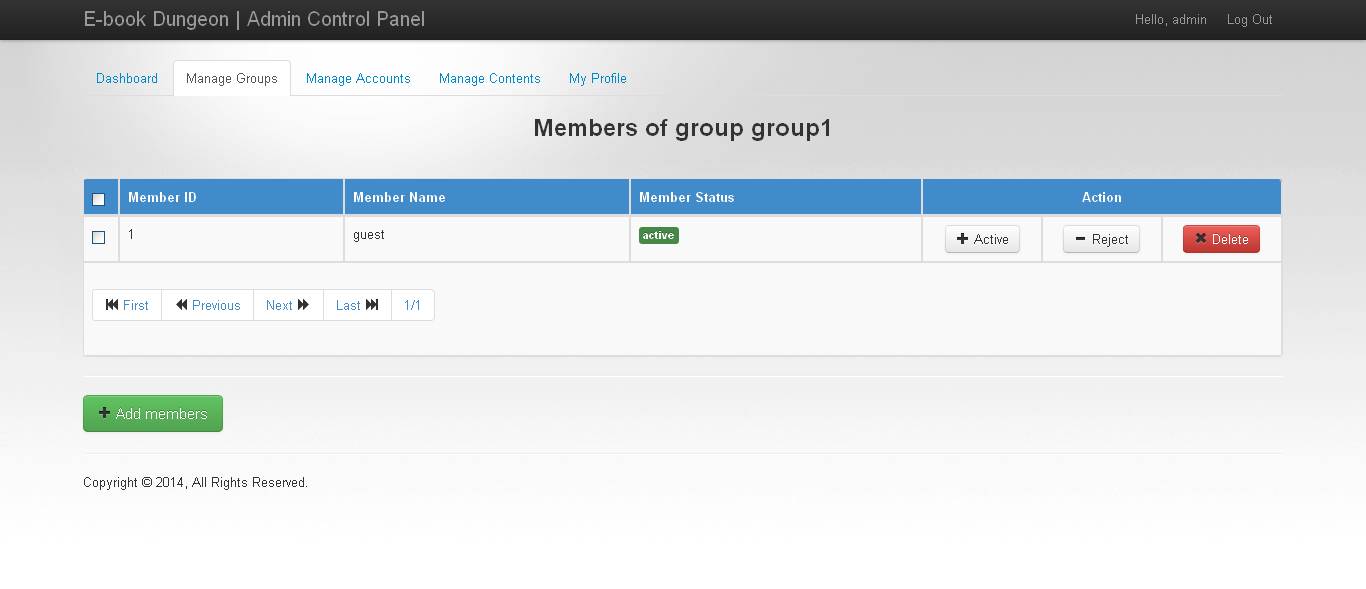
Picture 3: Viewing document by browser



Picture 4: Uploading



Picture 5: Admin's manage contents page



Picture 6: Admin's manage group member page

## Evaluation

### Advantages

The website has no security issue detected, as SHA hashed password, anti SQL injection, anti XSS were implemented.

### Disadvantages

# Conclusion

The website has most of its essential functions working. Some functions of initial requirement were not completed, for example, "Edit document atrributes", or "Download all document from a group", etc. due to either time constraint or technical difficulties. However, the develop team has successfully applied web programming techniques learned through the course into this website.

In the future, this website can be further developed to be actually used as a commercial product. After finishing above uncompleted functions, some new features may be added, such as editing document online, online payment, document renting, etc. Also, security and interface still have lot of room to improve.

# Reference

[1] <http://docs.phalconphp.com/en/latest/>

[2] <https://github.com/PHPOffice/PHPExcel/wiki/User%20Documentation>

[3] <http://stackoverflow.com/>

[4] <http://www.tutorialspoint.com/bootstrap/bootstrap_quick_guide.htm>