

# 2

## User Management

User management is one of the core functions of a CMS—the engine needs to know who is allowed to edit documents, and needs a way to manage those users.

In this chapter, we will discuss the following points:

- Overview of user management
- What "roles" are, and how they work
- Storage of user data in a database
- Creation of a login system
- Using the ReCaptcha tool
- Forgotten-password management
- Create a user-management system

We will cover the basics of role-management, but will not go in-depth into it, as none of the features in the project CMS we are building will require it.

### Types of users

As applications evolve from simple scripts to complex systems, developers tend to add code and ideas as they occur and are needed.

In the beginning, when creating simple CMSs, this means that user access is confined to administrator logins, as user logins are not usually necessary for simple systems like news reporting, or image galleries.

So, the developer creates a table of administrators.

Later on, as the system evolves, it becomes necessary to create front-end users, so that people can log in and contribute comments or content, or purchase items with a user-based discount.

Again, because the system is slowly evolving, the developer now adds a table of front-end users.

But things then get complex – what if we want administrators to correspond with commenters, or someone who uses the system as a normal user but is also an admin?

One solution to this is to have one table of users, and a flag which states whether the user is a normal user or an admin.

But then, we have another problem – what if you want some users to be admins, but you want them to have access only to certain parts of the backend area? For example, let's say the user is in charge of uploading news stories – that user needs access to the admin area, but should not have access to, say, the user management areas.

## Roles

The solution is not to use flags, but to use "roles" (also called "groups").

A role is a group of permissions which you can assign to a user. I will use the words "role" and "group" interchangeably in the book – they essentially mean the same thing when speaking of user rights.

For example, you might have a role such as "page editor", which includes the following permissions:

- Can create pages
- Can delete pages
- Can edit pages

You might have a user who is allowed to edit pages and also to edit online store products, in which case you need to either have a single group which covers all those permissions, or two groups ("page editor" and "online store editor"), and the user is a member of both.

The latter case, multiple groups, is much easier to manage, and is in fact necessary; as the number of possible combinations of permissions grows exponentially, more roles are created.

Another important question is, where do these role names come from? Does an administrator create them?

It's an interesting question, because the answer is both "yes" and "no".

If in order to create roles, you need to be a member of the "administrator" role, then who creates the "administrator" role? What if the role is deleted?

So we have a case where a role should not be created by an administrator.

On the other hand, we might have an online store, and want to assign a 5% discount to all users who are members of the role "favored customers". Who creates that role? It makes sense that the administrator should be allowed to create as many custom roles as is needed. And it is impossible for a sensible application to be created which predicts all the roles that will be required by a user-defined system.

So, we have a case where a role should be created by an administrator.

In these cases, it is okay if the admin deletes the "favored customers" role, but not if the "administrator" role is deleted.

How do we get around this?

One solution, which we'll use in this book, is to prefix system-generated role names with '\_', and to disallow administrators from editing or creating role names that use that scheme.

We will define two starter roles:

- `_administrators`: This role gives a user permission to enter the admin part of a system
- `_superadministrators`: This role is a special one, which gives a user total access

We will not build a role management system in this book, because none of the other chapters will require it. We are discussing it here because it is better to prepare for a future need than to stumble across the need and have to rewrite a lot of hardcoded behavior.

## Database tables

To record the users in the database, we need to create the `user_accounts` table, and the `groups` table to record the roles (groups).

First, here is the `user_accounts` table. Enter it using phpMyAdmin, or the console:

```
CREATE TABLE `user_accounts` (  
  `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT ,  
  `email` text,  
  `password` char(32) DEFAULT NULL,  
  `active` tinyint DEFAULT '0',  
  `groups` text,  
  `activation_key` varchar(32) DEFAULT NULL,  
  `extras` text,  
  PRIMARY KEY (`id`)  
) DEFAULT CHARSET=utf8;
```

---

Name	Description
<code>id</code>	This is the primary key of the table. It's used when a reference to the user needs to be recorded.
<code>email</code>	You can never tell how large an e-mail address should be, so this is recorded as a text field.
<code>password</code>	This will always be 32 characters long, because it is recorded as an MD5 string.
<code>active</code>	This should be a Boolean (true/false), but there is no Boolean field in MySQL. This field says whether the user is active or disabled. If disabled, then the user cannot log in.
<code>groups</code>	This is a text field, again, because we cannot tell how long it should be. This will contain a JSON-encoded list of group names that the user belongs to.
<code>activation_key</code>	If the user forgets his/her password, or is registering for the first time, then an activation key will be sent out to the user's e-mail address. This is a random string which we generate using MD5.
<code>extras</code>	When registering a user, it is frequently desired that a list of extra custom fields such as name, address, phone number (and so on) also be recorded. This field will record all of those using JSON. If you prefer, you could call this "usermeta", and adjust your copy of the code accordingly.

---

Note the usage of JSON for the `groups` field (or "column", if you prefer that word). Deciding whether to fully normalize a database, or whether to combine some values for the sake of speed, is a decision that often needs to be made.

In this table, I've decided to combine the groups into one field for the sake of speed, as the alternative is to use three table (the `user_accounts` table, the `groups` table, and a linking table), which would be slower than what we have here.

If in the future, it becomes necessary to separate this out into a fully normalized database, a simple upgrade script can be used to do this.

For now, populate the table with one entry for yourself, so we can test a login. Remember that the password needs to be MD5-encoded.

Note that MD5, SHA1, and other hashing functions are all vulnerable to collision-testing. If a hacker was to somehow get a copy of your database, it would be possible to eventually find working passwords for each MD5 or SHA1 hash. Of course, for this to happen, the hacker must first break into your database, in which case you have a bigger problem.

Whether you use SHA1, MD5, `bcrypt`, `scrypt`, or any of the other hashing functions is a compromise between your need for security (`bcrypt` being more secure), or speed (MD5 and SHA1 being fast).

Here's an example insert line:

```
insert into user_accounts
  (email,password,active,groups)
values (
  'kae@verens.com',
  md5('kae@verens.com|my password'),
  1,
  '["_superadministrators"]'
);
```

Notice that the groups field uses JSON.

If we used a comma-delimited text field, then that would make it impossible to have a group name with a comma in it. The same is true of other character delimiters.

Also, if we used integer primary key references (to the groups table) then it would require a table join, which takes time.

By putting the actual name of the group in the field instead of a reference to an external table row, we are saving time and resources.

The password field is also very important to take note of.

We encrypt the password in the database using MD5. This is so that no one knows any user's password, even the database administrator.

However, simply encrypting the password with MD5 is not enough. For example, the MD5 of the word `password` is `5f4dcc3b5aa765d61d8327deb882cf99`. This may look secure, but when I run a search for that MD5 string in a search engine, I get 28,300 results!

This is because there are vast databases online with the MD5s of all the common passwords.

So, we "salt" the MD5 by adding the user's e-mail address to it, which causes the passwords to be encrypted differently for each user, even if they all use the same password.

This gets around the problem of users using simple passwords that are easily cracked by looking up the MD5. It will not stop a determined hacker who is willing to devote vast resources to the effort, but as I said earlier, if someone has managed to get at the database in the first place, you probably have bigger problems.

Now, let's put this table to use by creating the login form and the login mechanism.

## Admin area login page

In *Chapter 1, CMS Core Design*, we discussed a number of different systems used by CMSs to allow administrators to log in. Some have the administrator log in using the same form as a normal user would log in with, some have totally separate domains dedicated to administration, and some even have dedicated desktop programs.

We will use a defined directory within the CMS structure, `/ww.admin`. This is how CMSs such as Joomla! or WordPress manage administration. In Joomla!, administrators log into `/administrator`, and in WordPress, administrators log into `/wp-admin`.

How the administration pages will work is that whenever a page is loaded, it checks first to see if you are logged in as an admin, and if not, you are shown a login page.

So, create the directory `ww.admin` in your web root, and let's create a page called `index.php` in that directory:

```
<?php
require 'admin_libs.php';
echo 'you are logged in!';
```

The file `/ww.admin/admin_libs.php` will be included by every page in the admin area. Create that now:

```
<?php
require $_SERVER['DOCUMENT_ROOT'].'ww.incs/basics.php';
function is_admin(){
    if(!isset($_SESSION['userdata']))return false;
    return (
        isset(
            $_SESSION['userdata']['groups']['_administrators']
        ) ||
        isset(
            $_SESSION['userdata']['groups']['_superadministrators']
        )
    );
}
if(!is_admin()){
    require SCRIPTBASE.'ww.admin/login/login.php';
    exit;
}
```

So what happens here is that each time the `admin_libs.php` file is loaded, it checks first that a `userdata` session variable has been created and that it contains either the group `_administrators` or `_superadministrators`. Remember, `_administrators` have access to the admin area, and `_superadministrators` have total access—there is not much of a difference in this book's project, but the difference is important enough that we should "future-proof" the system by using this difference now.

If the function `is_admin()` returns `false`, then the browser is sent a login page, which we'll create next.

Create a directory `/ww.admin/login`, and create the file `login.php` in it:

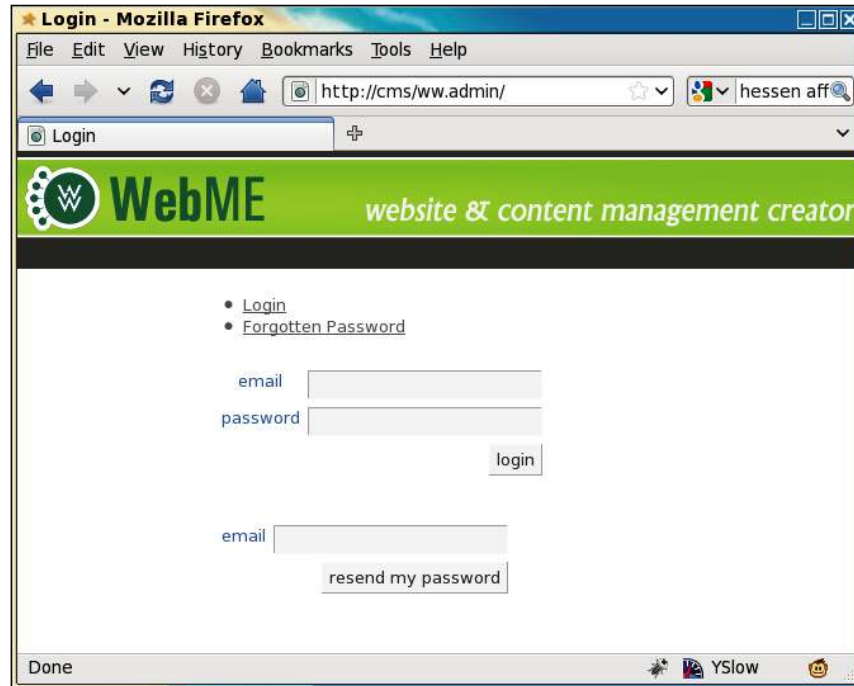
```
<html>
<head>
    <title>Login</title>
    <link rel="stylesheet" type="text/css"
        href="/ww.admin/login/login.css" />
</head>
<body>
    <div id="header"></div>
    <div class="tabs">,
        <ul>
            <li><a href="#tab1">Login</a></li>
            <li><a href="#tab2">Forgotten Password</a></li>
```

```
</ul>
<div id="tab1">
  <form method="post"
    action="/ww.incs/login.php?redirect=<?php
    echo $_SERVER['PHP_SELF'];
    ?>">
    <table>
      <tr><th>email</th><td>
        <input id="email" name="email" type="email" />
      </td></tr>
      <tr><th>password</th><td>
        <input type="password" name="password" />
      </td></tr>
      <tr><th colspan="2" align="right">
        <input name="action" type="submit"
          value="login" class="login" />
      </th></tr>
    </table>
  </form>
</div>
<div id="tab2">
  <form method="post"
    action="/ww.incs/forgotten-password.php?redirect=<?php
    echo $_SERVER['PHP_SELF'];
    ?>">
    <table>
      <tr><th>email</th><td>
        <input id="email" type="text" name="email" />
      </td></tr>
      <tr><th colspan="2" align="right">
        <input name="action" type="submit"
          value="resend my password" class="login" />
      </th></tr>
    </table>
  </form>
</div>
</div>
</body>
</html>
```

A `login.css` file is referenced in that source. The contents of it are not important to what we're doing, so we won't bother repeating it here. The CSS and images are available to download from Packt's website along with all source code from this project.



There are two forms in there; the first is for logging in, and the second is for reminding the user of the password, if the password has been forgotten.



Notice that we ask for the e-mail address of the user, and not a username.

When people choose usernames, if there are a lot of users in the system, it is likely that the username that the person wants in the first place is already taken. For example, I like to log in everywhere as "kae". Unfortunately, in very large systems, that username can be already taken. This would be a bigger problem for people named "James" or "John", and so on.

E-mails, though, are unique. You cannot have two people logged in who have the same e-mail address.

Another reason is that e-mail addresses tend not to be forgotten. People generally have only one or two e-mail addresses that they use. If it's not one, it's the other.

Yet another reason is that if you have forgotten your password, then a reminder service can be used to send a "reset" URL to the registrant's e-mail account.

If you go a very long time without forgetting the password, then it is possible that by the time you need the reminder, you will no longer have access to the e-mail account you used to create the account—you may have changed company, or some other reason.

But, if you're using your e-mail address as the account name, and realize you are about to lose access to it, then the very act of logging in will remind you that you need to change the user account details before you forget the password.

Another thing to note about the HTML is the target of the forms.

We have a single login point, `/ww.incs/login.php`, which can be used by both administrators and normal users. The `redirect` parameter is used to tell the server where the browser should be sent after the login is done.

We're not quite done yet with that file. The screen is a little bit bland. We can use our first piece of jQuery to liven it up a bit using tabs.

Change the header by adding these highlighted lines:

```
<title>Login</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/
jquery.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.0/
jquery-ui.min.js"></script>
<link rel="stylesheet" type="text/css" href="http://ajax.
googleapis.com/ajax/libs/jqueryui/1.8.0/themes/south-street/jquery-ui.
css" />
<script src="/ww.admin/login/login.js"></script>
<link rel="stylesheet" type="text/css"
href="/ww.admin/login/login.css" />
```

The first three highlighted lines load up jQuery and jQuery UI from Google's **Content Delivery Network (CDN)**, and load up a jQuery UI stylesheet as well.

Some people don't like to use Google's CDN, so you may want to download the jQuery and jQuery UI files and link to them on your local server. I've never had a problem using Google's CDN.



Linking to a CDN has some advantages, such as quicker access in cases where the browser is far from the site (the CDN copy may be physically closer to the browser, thus causing less network lag), less bandwidth usage for your own site, and less files to maintain on your own system.

When building a large application, there's a lot of "widget" functionality (tabs, auto completes, sliders, drag/drop, and so on) which may be used in various places. The jQuery UI project provides a lot of these, and is extremely simple to use.

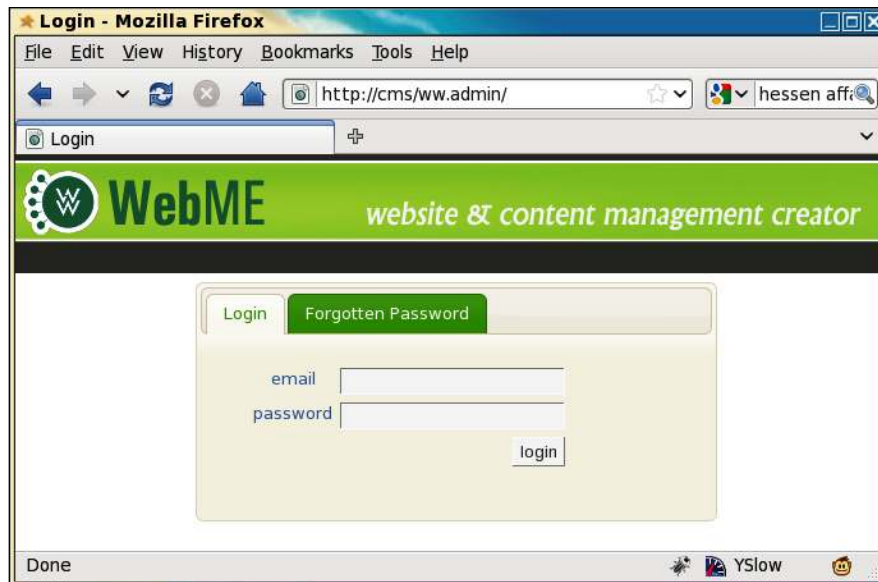
The last line is a link to a local script, which we'll use to set up the tabs. Create the file `/ww.admin/login/login.js`:

```
$(function() {  
    $('.tabs').tabs();  
});
```

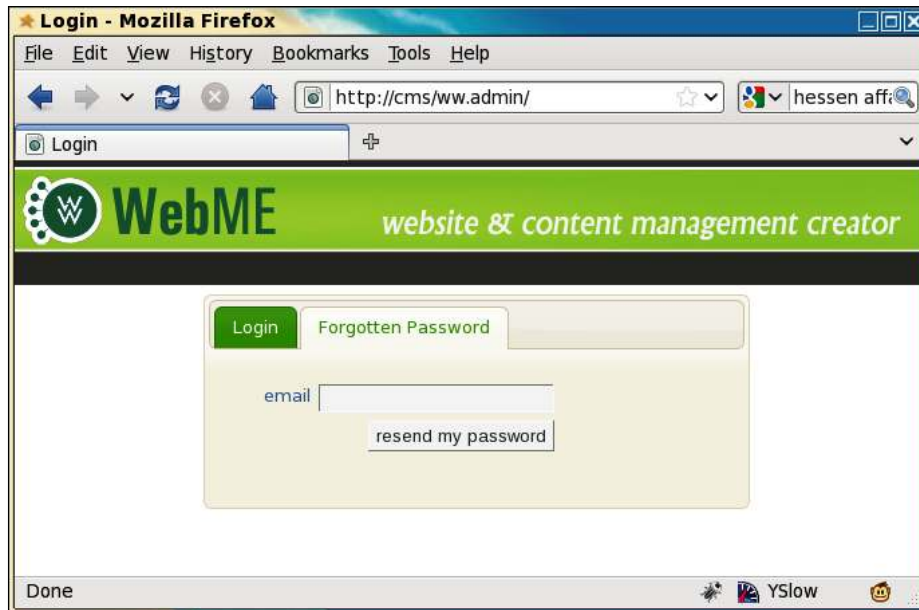
This small piece of code tells jQuery: "When the page is finished loading, run the function `.tabs()` against all elements with the class `tabs`".

 Wrapping a function inside `$( )` is equivalent to running `$( document ). ready ( )` with the function as a parameter. 

After the browser runs that tiny piece of code, the **Login** page now looks like this:



And if the **Forgotten Password** tab is clicked, then it appears as follows:



For a full explanation of how tabs work, see the jQuery UI website – <http://jqueryui.com/demos/tabs/>.

There is one more thing that is needed before the login form is complete.

In order to stop malicious robot programs from trying to log in using brute force to guess the password, and also to stop similar robots from sending out reminder e-mails to you and resetting your password, we will use a "captcha" to verify that whoever is filling in the form is human.

A captcha is a picture of some text. It is obscured slightly by deforming the image or adding static, and so on, so that it is not easy for a robot to decipher it using an optical character recognition program.

Generating captchas is not difficult – there are many scripts online that do it for you. However, if you use a script that you are not constantly tweaking, then it is possible that someone will eventually find a way to decipher the captcha automatically.

A good solution is to use the **reCAPTCHA** library (<http://recaptcha.net/>). This is a well-known captcha program which generates images based on photographs of old books. It also provides alternative audio from old radio shows in case the user cannot see clearly.

Download the latest `recaptcha-php` script from <http://code.google.com/p/recaptcha/> – at the time of writing, this is `recaptcha-php-1.10.zip` – and unzip it in `/ww.incs` so you have a directory called `/ww.incs/recaptcha-php-1.10`. If you found a newer one, replace `-1.10` with whatever is appropriate.

You will also need to get an API key. This is a string of characters which identifies you to the reCAPTCHA engine when it's used. Do this by creating a user account at <http://recaptcha.net/>. If you plan on using your CMS on more than one domain, then make sure to tick the **Enable this key on all domains** check-box while registering.

After registering, you will be given a "public key" and a "private key".

We will record the keys in a file named `/ww.incs/recaptcha.php`:

```
<?php
require SCRIPTBASE
    . 'ww.incs/recaptcha-php-1.10/recaptchalib.php';

define('RECAPTCHA_PRIVATE', ''); // place private key here
define('RECAPTCHA_PUBLIC', ''); // place public key here
```

Replace the second parameters of these lines with your keys. We've placed this file in the `/ww.incs/` directory so that it can be accessed by any code that needs it, whether it's in the admin section or the public section. Also, as the file is named `captcha.php` and doesn't mention the version number of the library, installing a new copy of the reCAPTCHA library involves simply unzipping it in the `/ww.incs` directory and changing the given `require` line to match it.

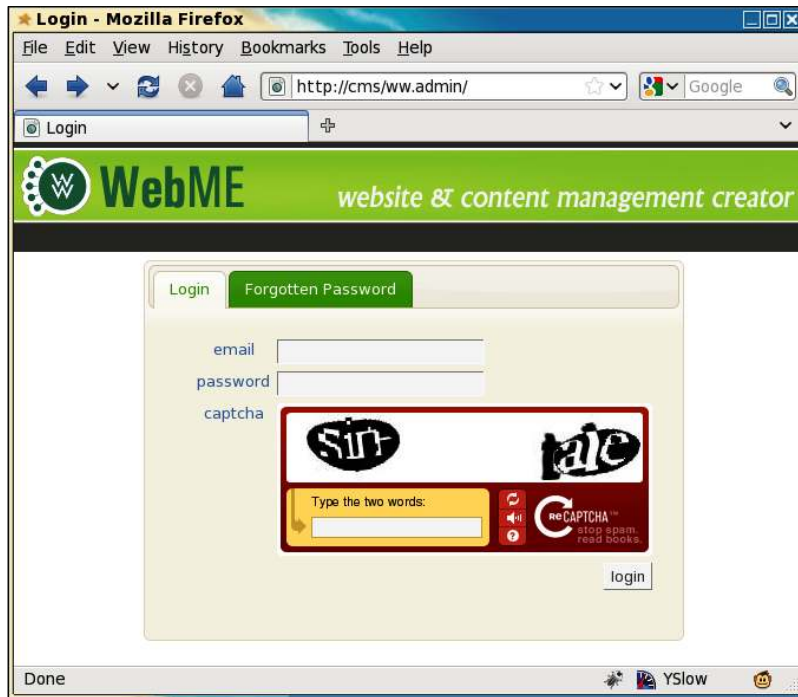
At the top of the `/ww.admin/login/login.php` page, add these highlighted lines:

```
<?php
    require SCRIPTBASE.'ww.incs/recaptcha.php';
    $captcha=recaptcha_get_html(RECAPTCHA_PUBLIC);
?>
<html>
    <head>
```

And in the login form's table, add this just before the submit button's row:

```
<tr id="captcha">
    <th>captcha</th>
    <td><?php echo $captcha; ?></td>
</tr>
```

When the given code is rendered, the captcha writes some HTML which imports an external JavaScript file, and if no JavaScript is available to the browser, then it also shows an `iframe` with alternative HTML in it.



Because we have two forms on the page, we should logically want two captchas as well. Unfortunately, you cannot have two captchas on the same page, as each image will be different (they're never cached), and each new captcha invalidates the old one. So, if you had two, only one of them would work.

So, what we will do is add a little bit of jQuery that moves the captcha whenever a tab is clicked.

To do this, rewrite the `/ww.admin/login/login.js` file completely:

```
$(function() {
  // remove the captcha's script element
  $('#captcha script').remove();
  // set up tabs
  $('.tabs').tabs({
    show: function(event, ui) {
      // if the captcha is already here, return
      if($('#captcha', ui.panel).length) return;
      // move the captcha into this panel
    }
  });
});
```

```
        $('table tr:last', ui.panel).before($('#captcha'));
    }
  });
});
```

When the page is loaded, the given script runs.

First, it removes the captcha's `<script>` element. Otherwise, when it is moved, the script will run again, breaking the captcha.

Then, we add some code which tells jQuery UI that whenever a tab panel is shown, we want to check it for the captcha row. If the row doesn't exist, then move it from where it is, to the present panel.

The highlighted line handles the moving.

Okay! We are finally finished with the login forms. Now, let's handle the actual login.

## Logging in

It is tempting to have a separate login script for the admin and normal users, but this can cause problems in the future if you ever change how logins work.

In the form that we created, we set the action to `/ww.incs/login.php`, with an added parameter named "redirect".

What's involved with a login is as follows:

- Verify that the submitted captcha is correct (we don't want robots logging in!)
- Verify there is an entry in `user_accounts` where the submitted e-mail address and password are matched
- If all is well, set a session variable named `userdata` which holds the user's information (saves looking it up in the database all the time)
- Send the browser to wherever the redirect link pointed it, or to the root of the site if none is provided, or if the provided one is invalid
- If anything goes wrong, still send the browser on to the redirect page, but also give an error message as an added parameter

Some of the code for the login will also be needed for other aspects of logins, such as logouts and forgotten passwords, so we'll start this by creating `/ww.incs/login_libs.php`:

```
<?php
require 'basics.php';

$url='/';
$error=0;

function login_redirect($url,$msg='success'){
    if($msg)$url.='?login_msg='.$msg;
    header('Location: '.$url);
    echo '<a href="'.htmlspecialchars($url).'">redirect</a>';
    exit;
}

// set up the redirect
if(isset($_REQUEST['redirect'])){
    $url=preg_replace('/[?&].*/','',$REQUEST['redirect']);
    if($url=='')$url='/';
}
}
```

All of the login functions will require a redirect after the action, so this creates a function for handling the redirect, and does some simple validation on the requested `redirect_url`, such as removing any query string parameters.

If the parameters were not removed, it is possible an admin on your CMS might be fooled into going to a link such as `http://cms/ww.admin/?delete-all-pages`, and after the login, they might be redirected back to that (fake, just an example) URL which would then proceed and delete all pages.

So, we neutralize this problem by removing anything past a `?` or `&`.

Create a file, `/ww.incs/login.php`, containing the following code:

```
<?php
require 'login_libs.php';

login_check_is_email_provided();

// check that the password is provided
if(!isset($_REQUEST['password']) || $_REQUEST['password']==''){
    login_redirect($url,'nopassword');
}

login_check_is_captcha_provided();
login_check_is_captcha_valid();
```



---

```

// check that the email/password combination matches a row in the user
table
$password=md5($_REQUEST['email'].'|'.$_REQUEST['password']);
$r=dbRow('select * from user_accounts where
    email="'.addslashes($_REQUEST['email']).'" and
    password="'. $password.'" and active'
);
if($r==false){
    login_redirect($url,'loginfailed');
}
// success! set the session variable, then redirect
$_SESSION['userdata']=$r;
$groups=json_decode($r['groups']);
$_SESSION['userdata']['groups']=array();
foreach($groups as $g)$_SESSION['userdata']['groups'][$g]=true;
if($r['extras']=='')$r['extras']='[]';
$_SESSION['userdata']['extras']=json_decode($r['extras']);
login_redirect($url);

```

This checks all inputs, sets a session variable if the login is valid, and in all cases does a redirect to send the browser where it was going. The `$_REQUEST` super-global variable is generated by merging the `$_POST` and `$_GET` variables.

There are a number of functions referenced in there that are not defined. We define those in `/ww.incs/login-libs.php` so they can be reused by the other login scripts (add the functions to the end of the file):

```

// check that the email address is provided and valid
function login_check_is_email_provided(){
    if(
        !isset($_REQUEST['email']) || $_REQUEST['email']==''
        || !filter_var($_REQUEST['email'], FILTER_VALIDATE_EMAIL)
    ){
        login_redirect($GLOBALS['url'],'noemail');
    }
}

// check that the captcha is provided
function login_check_is_captcha_provided(){
    if(
        !isset($_REQUEST["recaptcha_challenge_field"]) || $_
REQUEST["recaptcha_challenge_field"]==''
        || !isset($_REQUEST["recaptcha_response_field"]) || $_
REQUEST["recaptcha_response_field"]==''
    ){

```

```
        login_redirect($GLOBALS['url'],'nocaptcha');
    }
}

// check that the captcha is valid
function login_check_is_captcha_valid(){
    require 'recaptcha.php';
    $resp=recaptcha_check_answer(
        RECAPTCHA_PRIVATE,
        $_SERVER["REMOTE_ADDR"],
        $_REQUEST["recaptcha_challenge_field"],
        $_REQUEST["recaptcha_response_field"]
    );
    if(!$resp->is_valid){
        login_redirect($GLOBALS['url'],'invalidcaptcha');
    }
}
```

You'll also have noticed that the `login_redirect()` function has two parameters; the first is the URL to redirect to, and the second is a text code which designates a message to be shown.

Now let's make use of that message code.

First create a file `/ww.incs/login-codes.php`:

```
<?php
$login_msg_codes=array(
    'success'=>'login successful.',
    'noemail'=>'no email address provided, or the email'
        .' address was invalid.',
    'nopassword'=>'no password provided.',
    'nocaptcha'=>'no captcha provided.',
    'invalidcaptcha'=>'captcha invalid.',
    'loginfailed'=>'login incorrect. if you\'ve forgotten'
        .' your password, please use the Forgotten Password form.',
    'permissiondenied'=>'your user account does not have'
        .' permission for this area.'
);
```

These correspond to the `$msg` codes in the login script.

I've added two, for those cases where a person has logged in as a normal user but doesn't have access permission for the admin area (or another area where the user doesn't have the required role).

Let's use the codes. Edit the `/ww.admin/login/login.php` file and add the following highlighted lines:

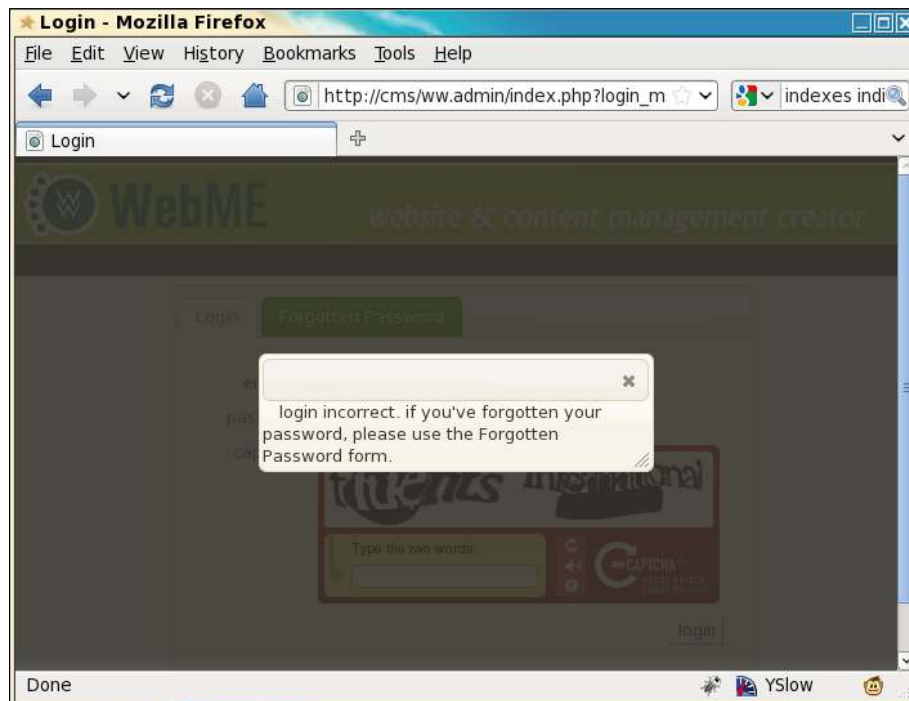
```

        <div id="header"></div>
<?php
if(isset($_REQUEST['login_msg'])){
    require SCRIPTBASE.'ww.incs/login-codes.php';
    $login_msg=(int)$_REQUEST['login_msg'];
    if(isset($login_msg_codes[$login_msg])){
        echo '<script>$(function(){$("<strong>'
            .htmlspecialchars($login_msg_codes[$login_msg])
            .'</strong>").dialog({modal:true});});</script>';
    }
}
?>
<div class="tabs">

```

We first check that a valid message code was sent, then display it as a modal dialog using jQuery UI's `.dialog` plugin.

A visitor could simply change the URL's `login_msg` value to make the various messages appear, but it would be pointless of them to do that as it would not affect their user status.



You can change the dialog content so it has some prettier HTML if you wish.

Now, what if a user is logged in, but doesn't have admin rights?

We'll start testing this one by adding a user to the database with no groups:

```
mysql> insert into user_accounts
(email,password,active,groups)
values('user@verens.com',
      md5('user@verens.com|userpass'),1, ' [] ');
Query OK, 1 row affected (0.03 sec)
```

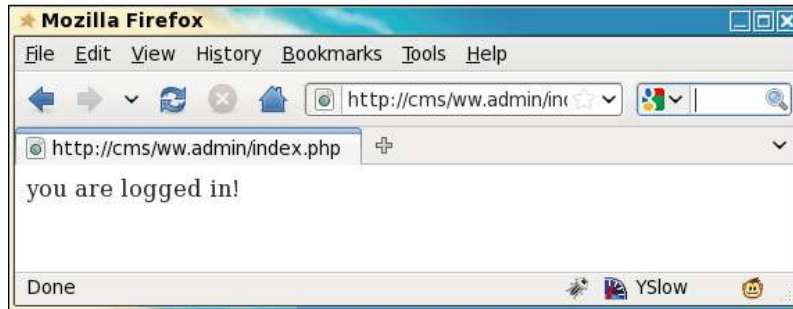
Now, we will change the `/ww.admin/admin_libs.php` file—remember that it has a function in it called `is_admin()`. Change that function to this:

```
function is_admin(){
    if(!isset($_SESSION['userdata']))return false;
    if(
        isset($_SESSION['userdata']['groups']['_administrators']) ||
        isset(
            $_SESSION['userdata']['groups']['_superadministrators'])
    )return true;
    if(!isset($_REQUEST['login_msg'])) $_REQUEST['login_
msg']='permissiondenied';
    return false;
}
```

So in this case, we know that the user is logged in, and doesn't have admin rights, so we set the `$_REQUEST['login_msg']` to 'permissiondenied' if there is not already another message set.

We avoid overwriting an existing message because that existing message has priority. For example, a logged-in user trying to log in as an admin user would not find the "permission denied" message very useful when the actual problem is that they got the password wrong.

Okay – so now do the login and use your proper admin details, filling in the captcha correctly.



As bland as that appears, this little message means you've successfully written a login script, which verifies your e-mail and password, with a captcha, and verifies that you are either an administrator or superadministrator.

We are now in the admin area properly!

So, what do we need next? We have not written the forgotten password reminder, and we also need to provide a method of logging out.

Let's do the logout first.

## Logging out

Logging out is much simpler than logging in. All we need to do is to remove the `userdata` session variable that we created when logging in.

First off, let's edit `/ww.admin/index.php` to add in some design, and the start of the admin menu, including the logout link:

```
<?php
require 'header.php';
echo 'you are logged in!';
require 'footer.php';
```

The footer will simply close off the HTML of the design, so here's `/ww.admin/footer.php`:

```
</div>
</body>
</html>
```

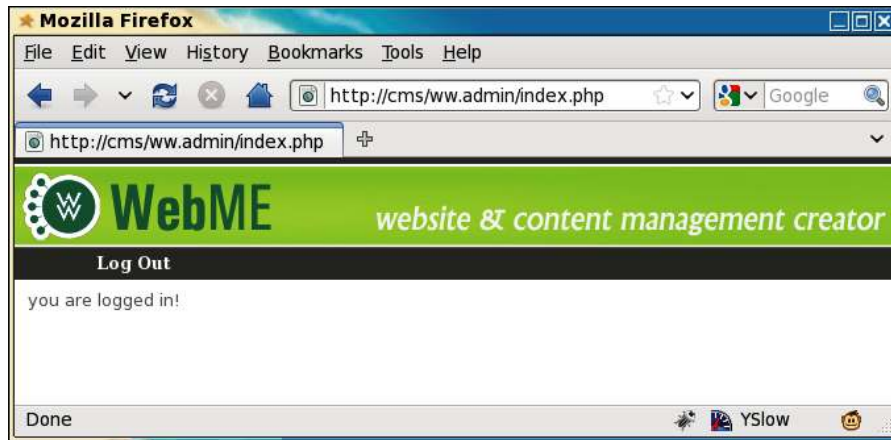
And here's the header — /ww.admin/header.php:

```
<?php
header('Content-type: text/html; Charset=utf-8');
require 'admin_libs.php';
?>
<html>
  <head>
    <script src="http://ajax.googleapis.com/ajax/
      libs/jquery/1.4.2/jquery.min.js"></script>
    <script src="http://ajax.googleapis.com/ajax/
      libs/jqueryui/1.8.0/jquery-ui.min.js"></script>
    <link rel="stylesheet" href="/ww.admin/theme/admin.css"
      type="text/css" />
    <link rel="stylesheet" href="http://ajax.googleapis.com/
      ajax/libs/jqueryui/1.8.0/themes/south-street/
      jquery-ui.css" type="text/css" />
  </head>
  <body>
    <div id="header">
      <div id="menu-top">
        <ul>
          <li><a
            href="/ww.incs/logout.php?redirect=/ww.admin/">
            Log Out</a></li>
        </ul>
      </div>
    </div>
    <div id="wrapper">
```

We will be using jQuery and jQuery UI in all parts of the admin area, so they are included by default.

Again, I've linked to a CSS file which I won't go through in this book. You can download it with the rest of the files from Packt.

Here's the admin index page now with the design and **Log Out** link included:



Now, let's create `/ww.incs/logout.php`:

```
<?php
$url='/';
session_start();

// set up the redirect
if(isset($_REQUEST['redirect'])){
    $url=preg_replace('/[\?\&].*/','',$REQUEST['redirect']);
    if($url=='')$url='/';
}

unset($_SESSION['userdata']);

header('Location: '.$url);
echo '<a href="'.htmlspecialchars($url).'">redirect</a>';
```

In this case, it's not necessary to include any libraries (the `/ww.incs/basics.php` file, for example). All we need to do is `unset($_SESSION['userdata'])` and redirect the browser. And so, we don't need to include `login-libs.php`.

Now, let's work on the forgotten password section.

## Forgotten passwords

There are many ways that CMSs handle missing passwords. In some cases, a new password is sent out through e-mail, in some cases, a security question lets the site verify that the user is who he or she claims to be, and in some cases, a validation e-mail is sent to verify the requester is the owner of the e-mail address.

In this section, I'll mention a few security concerns. I must add that in most cases, it is very unlikely that they will ever happen. But, as a developer of software, you should be aware of those things that can go wrong and do your best to make sure they don't happen in the first place.

If you give the option of resetting the password when a user fills in their e-mail address in the forgotten password form, there are some problems to beware of:

1. You may have just allowed an anonymous person to invalidate someone's account just because they knew the e-mail address of the valid user. If this is done repeatedly, it can really annoy that user, who may have to use a different password every time they log in.
2. E-mail is insecure. Because it is sent through plain text in most cases (yes, PGP e-mail is possible, but it is rarely used by normal users), you are sending passwords that can be potentially read by the e-mail hosters, or anyone that "taps the line".

If you give the option of changing a password by verifying the identity of the user using a "security question", there are also some problems. Some sites make the user pick from a set list of questions, none of which are secure:

1. It is easy to figure out someone's mother's maiden name. There are many genealogy websites online where that information is readily available.
2. Asking who the user's first teacher was is silly. Personally, I barely remember who I met two years ago; let alone 30 years ago!
3. Asking the name of the user's pet assumes that there is a pet in the first place, and that there is only one pet (I have three cats at the moment). Also, all of your friends probably know that pet's name. My cats are Buffy, Thurston, and Tweedo.
4. Car's license number. Again, there is an assumption. I don't drive, and have never owned a car.

Allowing the user to pick a security question is also silly. In most cases, the question will be obvious. As an example, I was chatting with a friend once about this very problem, and demonstrated it by opening up his Hotmail account — as his security question, he'd written something silly like "wibble", and I guessed correctly that if his security question was as much rubbish as that, then his answer would also be rubbish. I entered "wibble" again and was in.

There is possibly no real correct solution to the problem of verifying someone's identity over the Internet, so it's best to choose the "least worst" of the methods.



I mentioned a third possibility – sending out a validation e-mail. E-mail is a very personal form of identification. It is rare these days that you will find anyone online that doesn't have one, and usually, they've had the same e-mail address for years on end – people get attached to their e-mail addresses. The validation e-mail method involves some simple steps:

1. In the validation e-mail method, the user has forgotten their password, and goes to the forgotten password form and enters their e-mail address.
2. An e-mail is sent to the e-mail address with a link embedded in it. This link has a validation code attached which is recorded in the database.
3. When the user clicks on the link, this verifies the person's identity and logs the person into the site.
4. The verification code is then removed from the database. This way the login links only work one single time.

It's not even necessary that the user reset the password as long as they're happy enough to generate a fresh validation link each time – on some sites I use very infrequently, I tend to have "moved on" to a new password and keep forgetting the password I used for those infrequent visits, so for those sites, I'm always using a validation link to log in!

Anyway – enough musing. Let's create the file `/ww.incs/password-reminder.php`:

```
<?php
require 'login-libs.php';
login_check_is_email_provided();
login_check_is_captcha_provided();
login_check_is_captcha_valid();

// check that the email matches a row in the user table
$r=dbRow('select email from user_accounts where
  email="'.addslashes($_REQUEST['email']).'" and active'
);
if($r==false){
  login_redirect($url, 'nosuchemail');
}

// success! generate a validation email, then redirect
$validation_code=md5(time().'|'.$r['email']);
$email_domain=preg_replace('/^www\.\/','',$SERVER['HTTP_HOST']);
dbQuery('update user_accounts set activation_key="'. $validation_
code.'"
  where email="'.addslashes($r['email']).'"');
```

```
$validation_url='http://'.$_SERVER['HTTP_HOST'].'/ww.incs/forgotten-  
password-validate.php?verification_code='.$validation_code.'&email='.$  
r['email'].'&redirect_url='.$url;  
mail(  
    $r['email'],  
    "[$email_domain] forgotten password",  
    "Hello!\n\nThe forgotten password form at http://".$_SERVER['HTTP_  
HOST']."/ was submitted. If you did not do this, you can safely  
discard this email.\n\nTo log into your account, please use the link  
below, and then reset your password.\n\n$validation_url",  
    "From: no-reply@$email_domain\nReply-to: no-reply@$email_domain"  
);  
login_redirect($url, 'validationsent');
```

This script is very similar to the login script, but all references to the password field in the database and `$_REQUEST` have been removed, and we add in the validation link generator.

Notice that we have added two message codes. Amend `/ww.incs/login-codes.php` and add them:

```
'permissiondenied'=>'your user account does not have'  
    .' permission for this area.',  
'nosuchemail'=>'that email address does not exist in the'  
    .' user accounts database',  
'validationsent'=>'a validation message has been sent to'  
    .' your email address. please check your email.'  
);
```

The e-mail's content, when it arrives, will look something like this:

```
Hello!  
  
The forgotten password form at http://cms/ was submitted. If you did  
not do this, you can safely discard this email.  
  
To log into your account, please use the link below, and then reset  
your password.  
  
http://cms/ww.incs/forgotten-password-verification.php?verification_co  
de=97e5daf0d6b96c1945ed450d29c63a42&email=kae@verens.com &redirect_  
url=/ww.admin/index.php
```

You should feel free to amend the validation code generator to write whatever message you want into it.

Now, we need to write the validation script, `/ww.incs/forgotten-password-verification.php`:

```
<?php
require 'login-libs.php';
login_check_is_email_provided();
// check that a verification code was provided
if( !isset($_REQUEST['verification_code'])
    || $_REQUEST['verification_code']==''
){
    login_redirect($url,'novalidation');
}
// check that the email/verification code combination matches a row in
the user table
$password=md5($_REQUEST['email'].'|'.$_REQUEST['password']);
$r=dbRow('select * from user_accounts where
    email="'.addslashes($_REQUEST['email']).'" and
    verification_code="'.$_REQUEST['verification_code'].'" and active'
);
if($r==false){
    login_redirect($url,'validationfailed');
}
// success! set the session variable, clear the code from the
// db, then redirect
dbQuery('update user_accounts set verification_code="" where
    email="'.addslashes($_REQUEST['email']).'"');
$_SESSION['userdata']=$r;
$groups=json_decode($r['groups']);
$_SESSION['userdata']['groups']=array();
foreach($groups as $g)$_SESSION['userdata']['groups'][$g]=true;
if($r['extras']=='' )$r['extras']='[]';
$_SESSION['userdata']['extras']=json_decode($r['extras']);
login_redirect($url,'verified');
```

In this one, we verify the e-mail address and validation code, and if they both are correct, then we do a login, and send a message reminding the user to reset their password.

Add these new message codes to `/ww.incs/login-codes.php`:

```
'validation sent'=>'a validation message has been sent to your email
address. please check your email.',
'novalidation'=>'no validation code provided.',
'validation failed'=>'that email and validation code combination does
not exist. maybe it has already been used. please use the Forgotten
Password to resend the validation email.',
'verified'=>'you have verified your email address and we have logged
you in. please remember to reset your password.'
);
```

And that is our login system completed.

In the next section, we will create a user management area in the admin area.

## User management

Okay! We now have the admin area login working, so let's build the first admin page. This will be the user management page, which allows us to create, delete, and edit users.

So first, we need to edit the `/ww.admin/header.php` to add in a link to the user management page. In the next chapter, we will rewrite the menu to make it easier to add items to it. For now, the links will be hardcoded as top-level menu items.

Change the menu list to this:

```
<ul>
  <li><a href="/ww.admin/users.php">Users</a></li>
  <li><a href="/ww.incs/logout.php?redirect=/ww.admin/">
    Log Out</a></li>
</ul>
```

Next, we will create `/ww.admin/users.php`:

```
<?php
require 'header.php';
echo '<h1>User Management</h1>';
echo '<div class="left-menu">';
echo '<a href="/ww.admin/users.php">Users</a>';
echo '</div>';

echo '<div class="has-left-menu">';
echo '<h2>User Management</h2>';
if(isset($_REQUEST['action']))require 'users/actions.php';
if(isset($_REQUEST['id']))require 'users/form.php';
```

```

require 'users/list.php';
echo '</div>';

echo '<script src="/ww.admin/users/users.js"></script>';
require 'footer.php';

```

Because management involves multiple separate functions – displaying lists of items and details of specific items, editing items, deleting and creating, if you do all this in one single file, the file gets huge and unmanageable.

Similarly, if you separate all these functions into separate files and keep all those files in one directory, it makes it difficult for a developer to find the right file to edit (see the root directory of a Mantis BT 1.2.0rc2 installation for an example: 219 files!).

To make it easier to figure out what's going on, I like to place grouped files into their own directories. Hence the login files are in `/ww.admin/login/`, the user management files are in `/ww.admin/users/`, and we'll see more examples as the book goes on.

Anyway... when we click on the **Users** link in the menu, what we want to see is a list of existing users.

Add this to `/ww.incs/basics.php` to give us a `dbAll()` function:

```

function dbAll($query,$key='') {
    $q = dbQuery($query);
    $results=array();
    while($r=$q->fetch(PDO::FETCH_ASSOC))$results[]=$r;
    if(!$key)return $results;
    $arr=array();
    foreach($results as $r)$arr[$r[$key]]=$r;
    return $arr;
}

```

What that does is, given an SQL query, it will build an array of results and return that.



I haven't commented on it yet, but the `db*` functions we are writing here use the PDO library to connect to the database.

One reason for using `dbAll`, `dbQuery`, and so on, instead of accessing the database directly through PDO, `mysql[i]_connect`, or any other method, is that it's easier to port the engine to another database or database library if all DB methods are encapsulated in a small number of wrapper functions.

If given a second parameter (for example, 'id'), then the returned array will be indexed using that parameter's value from each result row.

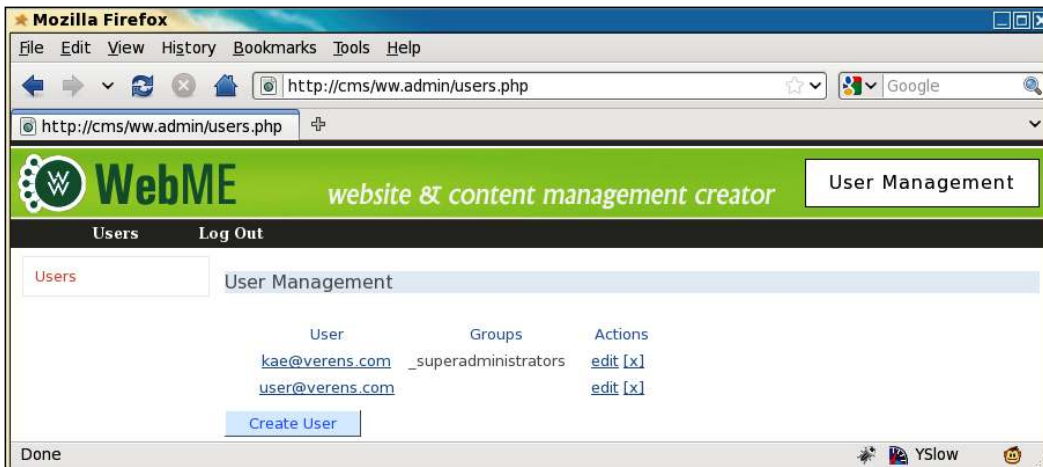
We'll also need a function for returning a single value. Add this to the same file:

```
function dbOne($query, $field='') {
    $r = dbRow($query);
    return $r[$field];
}
function dbLastInsertId() {
    return dbOne('select last_insert_id() as id','id');
}
```


Now that we have that, let's write `/ww.incs/users/list.php`:

```
<?php
$users=dbAll('select id,email,groups from user_accounts
order by email');
echo '<table style="min-width:50%">
<tr><th>User</th><th>Groups</th><th>Actions</th></tr>';
foreach($users as $user){
echo '<tr><th><a href="users.php?id='.$user['id']
.'">'.htmlspecialchars($user['email']).</a></th>';
echo '<td>'.join(', ',json_decode($user['groups'])).</td>';
echo '<td><a href="users.php?id='.$user['id'].'">edit</a>';
echo '&nbsp;<a href="users.php?id='.$user['id']
.'&amp;action=delete" onclick="return confirm(\'are you
sure you want to delete this user?\')">[x]</a></td></tr>';
}
echo '</table>';
echo '<a class="button" href="users.php?id=-1">
Create User</a>';
```

That gives me the following result:



We can now see the existing users, as well as the groups that they belong to.

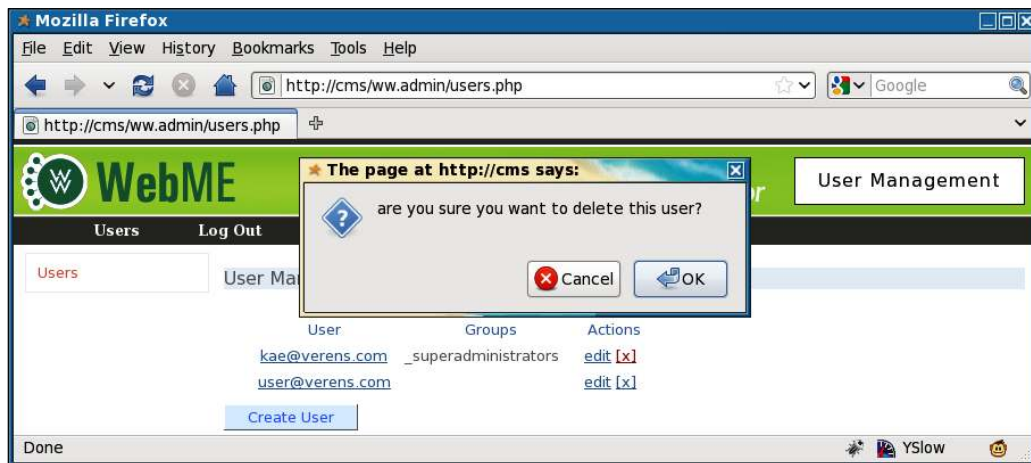
 The code I wrote here generates and echoes HTML directly to the HTTP stream. This is a "down and dirty" method of very quickly generating some code and displaying it. A more appropriate method would be to use a templating engine such as Smarty. Feel free to enhance the code after we've looked at Smarty later in the book.

Before talking about editing and creating, we will look at the delete action.

## Deleting a user

In the previous screenshot, you can see an [x] beside both users. The link is intentionally small and obscure, because we really don't want to accidentally delete a user by clicking the wrong link. So, we make the delete link more difficult to click.

We also add a JavaScript `confirm()` so that if an admin does click it, they are given the chance to say "No, I did not intend to click this".



Now we can write the code to do the deletion.

Create the file `/ww.admin/users/actions.php`:

```
<?php
$id=(int)$_REQUEST['id'];
if($_REQUEST['action']=='delete'){
    dbQuery("delete from user_accounts where id=$id");
    unset($_REQUEST['id']);
}
```

What happens with this is that the delete link is clicked, the user is deleted, then the `/ww.admin/users.php` page displays the users list again.

## Creating or editing a user

Creating and editing can both be done from the same form.

Basically, what happens is that you select to create or edit a user, which sends the user's ID to the server.

The server then uses that ID to get the user's data from the database. If the data doesn't exist, the result will obviously be blank.

The result is then used to fill in the user form.

When submitted, if the user ID is not valid, then the submission is used to create a new user.

For this form, we will need to create the groups database table, and populate it with `_administrator` and `_superadministrator`:

```
CREATE TABLE `groups` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` text,  
  PRIMARY KEY (`id`)  
) DEFAULT CHARSET=utf8;  
insert into groups values(1,"_superadministrators");  
insert into groups values(2,"_administrators");
```

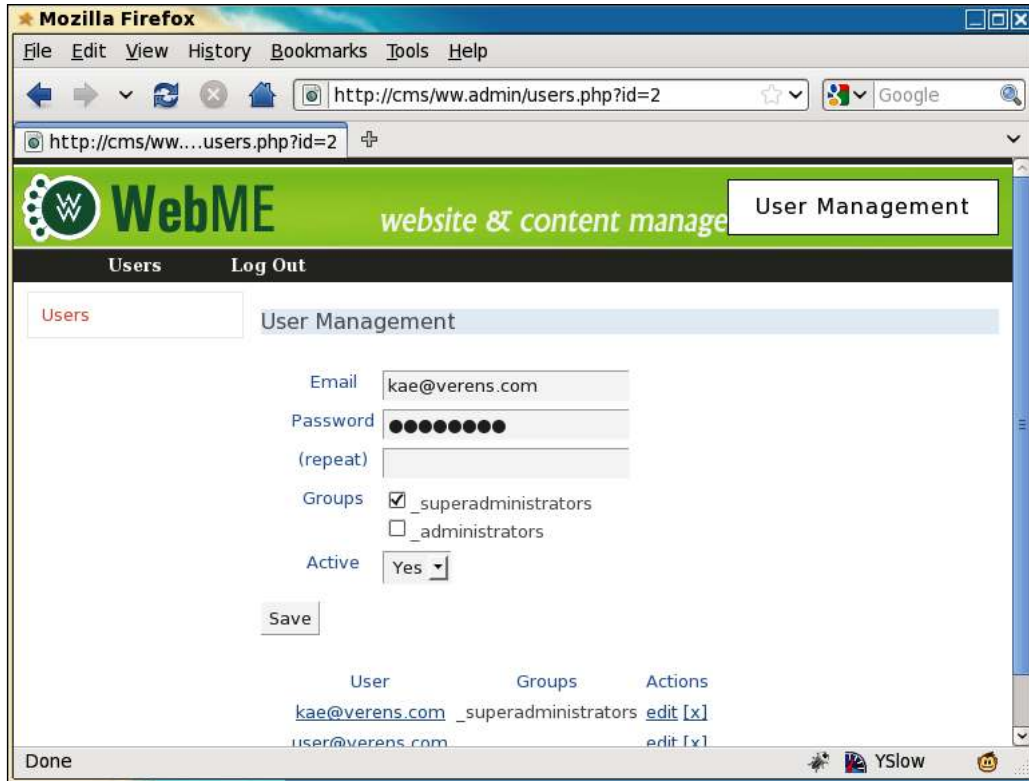
And now, create `/ww.admin/users/form.php`:

```
<?php  
$id=(int)$_REQUEST['id'];  
$groups=array();  
$r=dbRow("select * from user_accounts where id=$id");  
if(!is_array($r) || !count($r)){  
  $r=array('id'=>-1,'email'=>','active'=>0);  
}  
echo '<form action="users.php?id='.$id.'" method="post">  
  .<input type="hidden" name="id" value="'.$id.'" /><table>  
  .<tr><th>Email</th>  
    <td><input name="email" value="'.htmlspecialchars($r['ema  
il']).'" /></td>  
  </tr>  
  .<tr><th>Password</th>  
    <td><input name="password" type="password" /></td>
```



```
</tr>'
    . '<tr><th>(repeat)</th>
      <td><input name="password2" type="password" /></td>
    </tr>'
    . '<tr><th>Groups</th><td class="groups">';
$grs=dbAll('select id,name from groups');
$gms=array();
foreach($grs as $g){    $groups[$g['id']]=$g['name'];
}
$grs=json_decode($r['groups']);
foreach($groups as $k=>$g){
    echo '<input type="checkbox" name="groups['.$k.']"' ;
    if(in_array($g,$grs))echo ' checked="checked"' ;
    echo ' />',htmlspecialchars($g),'<br />';
}
echo '</td></tr>';
// }
echo '<tr><th>Active</th><td><select name="active">
    <option value="0">No</option>
    <option value="1"' . ($r['active']?'
    selected="selected"':'') . '>Yes</option></select></td></tr>';
echo '</table>';
echo '<input type="submit" name="action" value="Save" />';
echo '</form>';
```

After clicking on the **kae@verens.com** link, we get this form:



Next, we just need to save the updated data.

We can do this by adding this code to the end of the `/ww.admin/users/actions.php` file:

```
if($_REQUEST['action']=='Save'){
    $groups=$_REQUEST['groups'];
    if(!count($groups))$groups=array(0);
    $grs=dbAll('select name from groups where id in (
        .addslashes(join(', ',array_keys($groups)))
        .') order by name');
    $groups=array();
    foreach($grs as $r)$groups[]=$r['name'];
    $sql='set email="'.addslashes($_REQUEST['email'])."',
        active="'.(int)$_REQUEST['active']."',
        groups="'.addslashes(json_encode($groups))."'";
    if(
```

```
isset($_REQUEST['password']) &&
$_REQUEST['password']!=''
){
  if($_REQUEST['password']!= $_REQUEST['password2'])
    echo '<em>Password not updated. Must be entered
    the same twice.</em>';
  else $sql.=' ,password=md5("'.addslashes(
    $_REQUEST['email'].'|'.$_REQUEST['password']
    ).'")';
}
if($id==-1){
  dbQuery('insert into user_accounts '.$sql);
  $_REQUEST['id']=dbLastInsertId();
}
else{
  dbQuery('update user_accounts '.$sql.' where id='.$id);
}
echo '<em>users updated</em>';
}
```

That script will handle both the creation and editing of users.

We will discuss the creation and editing of groups later in the book.

## Summary

In this chapter, we created the login system, including captcha management and forgotten password management.

We also created a user management system for creating and editing users.

In the next chapter, we will start building the page management system.