

# Getting Started with Zend\_Auth

By Rob Allen, [www.akrobat.com](http://www.akrobat.com)

Document Revision 1.0.8

Copyright © 2007

This tutorial is intended to give a very basic introduction to using the Zend\_Auth component with the Zend Framework. It builds on the previous tutorial "Getting Started with the Zend Tutorial" which is available at <http://akrobat.com/zend-framework-tutorial>.

**NOTE:** This tutorial has been tested on versions 1.0.0 and 1.0.1 of the Zend Framework. It stands a very good chance of working with later versions.

## Before We Get Started

The implementation of authentication that we are going to use in this tutorial uses PHP's sessions. Make sure that the directory set in your php.ini file's *session.save\_path* is writable by the web server otherwise you will get a very odd exception thrown!

## Authentication

For the purposes of this tutorial, authentication is the process of logging someone into a web application. We are going to modify the CD listing application created in "Getting Started with the Zend Framework" to require log in before accessing any part of the application.

Loosely, the things we need to do are:

- Create a database table for users (and populate it with a user!).
- Create a login form.
- Create a controller containing actions for logging in and logging out.
- Alter the footer to allow for log out
- Ensure that the user is logged in before allowing access to the application.

## The *users* Table

The first thing we need is a users table. It doesn't need to be complicated, so the schema looks like this:

<i>Fieldname</i>	<i>Type</i>	<i>Null?</i>	<i>Notes</i>
id	Integer	No	Primary key, Autoincrement
username	Varchar(50)	No	Unique key
password	Varchar(50)	No	
real_name	Varchar(100)	No	

When using MySQL, the SQL statement to create the table is:

```
CREATE TABLE users (  
    id int(11) NOT NULL auto_increment,  
    username varchar(50) NOT NULL,  
    password varchar(50) NOT NULL,  
    real_name varchar(100) NOT NULL,  
    PRIMARY KEY (id),  
    UNIQUE KEY username (username)  
);
```

We also need a user that can log in:

```
INSERT INTO users (id, username, password, real_name)  
VALUES (1, 'rob', 'rob', 'Rob Allen');
```

Run these statements in a MySQL client such as phpMyAdmin or the standard MySQL command-line client. (Obviously, you should pick a better username and password!)

## Bootstrap Changes

In order to keep track of the fact that the user is logged in, we are going to use the session. The Zend Framework provides `Zend_Session_Namespace` which provides a nice object oriented interface to the session.

The changes to `index.php` are:

### zf-tutorial/index.php:

```
...
Zend_Loader::loadClass('Zend_Db_Table');
Zend_Loader::loadClass('Zend_Debug');
Zend_Loader::loadClass('Zend_Auth');

// load configuration
...
and
..
// setup database
$db = Zend_Db::factory($config->db->adapter,
    $config->db->config->toArray());
Zend_Db_Table::setDefaultAdapter($db);
Zend_Registry::set('db', $db);

// setup controller
$frontController = Zend_Controller_Front::getInstance();
...
```

All we actually have to do here is ensure we have loaded the `Zend_Auth` class and registered the database adapter with the registry. We store it to the registry as we need it when we do the authentication later.

## The Auth Controller

We need a controller to hold the login and logout actions. It makes sense to call it `AuthController`.

We'll start it with the basics from the `IndexController`:

### zf-tutorial/application/controllers/AuthController.php:

```
<?php
class AuthController extends Zend_Controller_Action
{
    function init()
    {
        $this->initView();
        $this->view->baseUrl = $this->_request->getBaseUrl();
    }

    function indexAction()
    {
        $this->_redirect('/');
    }
}
```

We set up `init()` so that the view is initialised and that `baseUrl` is assigned to it. We also create an `indexAction()` function as required by `Zend_Controller_Action`. We don't need `indexAction()` though as we are going to use `loginAction()` and `logoutAction()`, so we just redirect back to the default if someone does navigate to `auth/index`.

## Logging In

To log into the application we need a form, so the login action is going to work in a manner very similar to the other forms in IndexController. The form template will live in `views/scripts/auth/login.phtml` and the code will be in `AuthController::loginAction()`. The form is very simple, requiring just two fields: username and password:

### zf-tutorial/application/views/scripts/auth/login.phtml:

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>

<?php if(!empty($this->message)) :?>
<div id="message">
<?php echo $this->escape($this->message);?>
</div>
<?php endif; ?>

<form action="<?php echo $this->baseurl ?>/auth/login" method="post">
<div>
    <label for="username">Username</label>
    <input type="text" name="username" value=""/>
</div>
<div>
    <label for="password">Password</label>
    <input type="password" name="password" value=""/>
</div>
<div id="formbutton">
<input type="submit" name="login" value="Login" />
</div>
</form>

<?php echo $this->render('footer.phtml'); ?>
```

The template renders the `header.phtml` and `footer.phtml` at the top and bottom as usual. Note that we display a message only if `$this->message` is not empty. This is used to tell the user that they failed to log in. The rest of the template is the log in form itself.

Now that we have a form, we need to create the controller action that will run it. This is added to `AuthController.php`:

### zf-tutorial/application/controllers/AuthController.php:

```
class AuthController extends Zend_Controller_Action
{
    ...
    function loginAction()
    {
        $this->view->message = '';
        $this->view->title = "Log in";
    }
}
```

Initially, all we need to do is set the title and message and then render the form. If you navigate to `http://zf-tutorial/auth/login` then you should see the login form.

What about processing the form after it has been submitted though? To do that, we utilise the same trick as in the add and edit forms in the IndexController and do the process if the request method is a post. Change the `loginAction()` that we have just created:

### zf-tutorial/application/controllers/AuthController.php:

```
class AuthController extends Zend_Controller_Action
{
    ...
    function loginAction()
    {
        $this->view->message = '';
        if ($this->_request->isPost()) {
            // collect the data from the user
            Zend_Loader::loadClass('Zend_Filter_StripTags');
            $f = new Zend_Filter_StripTags();
            $username = $f->filter($this->_request->getPost('username'));
            $password = $f->filter($this->_request->getPost('password'));

            if (empty($username)) {
                $this->view->message = 'Please provide a username.';
            } else {
                // setup Zend_Auth adapter for a database table
                Zend_Loader::loadClass('Zend_Auth_Adapter_DbTable');
                $db = Zend_Registry::get('db');
                $authAdapter = new Zend_Auth_Adapter_DbTable($db);
                $authAdapter->setTableName('users');
                $authAdapter->setIdentityColumn('username');
                $authAdapter->setCredentialColumn('password');

                // Set the input credential values to authenticate against
                $authAdapter->setIdentity($username);
                $authAdapter->setCredential($password);

                // do the authentication
                $auth = Zend_Auth::getInstance();
                $result = $auth->authenticate($authAdapter);
                if ($result->isValid()) {
                    // success: store database row to auth's storage
                    // system. (Not the password though!)
                    $data = $authAdapter->getResultRowObject(null,
                        'password');
                    $auth->getStorage()->write($data);
                    $this->_redirect('/');
                } else {
                    // failure: clear database row from session
                    $this->view->message = 'Login failed.';
                }
            }
        }
        $this->view->title = "Log in";
    }
}
```

Quite a lot is going on here, so lets go through it:

```
// collect the data from the user
Zend_Loader::loadClass('Zend_Filter_StripTags');
$f = new Zend_Filter_StripTags();
$username = $f->filter($this->_request->getPost('username'));
$password = $f->filter($this->_request->getPost('password'));

if (empty($username)) {
    $this->view->message = 'Please provide a username.';
} else {
```

As usual we set up a filter and then extract the username and password fields from the POST data. Note that we use the request's `getPost()` function as it will handle the `isset()` check for us and return an empty string if the field doesn't exist in the POST array. If the username is empty, then there's not a lot of point in trying to authenticate (and `Zend_Auth` will throw an exception if we try!), so we check for an empty username and inform the user instead.

```

// setup Zend_Auth adapter for a database table
Zend_Loader::loadClass('Zend_Auth_Adapter_DbTable');
$db = Zend_Registry::get('db');
$authAdapter = new Zend_Auth_Adapter_DbTable($db);
$authAdapter->setTableName('users');
$authAdapter->setIdentityColumn('username');
$authAdapter->setCredentialColumn('password');

```

Zend\_Auth uses an adapter system to allow you to use any number of systems to actually do the authentication bit. We want to use a database table and so use Zend\_Auth\_Adapter\_DbTable. To set up an adapter, you tell it the fields to use and pass in a valid connection to the database.

```

// Set the input credential values to authenticate against
$authAdapter->setIdentity($username);
$authAdapter->setCredential($password);

```

We need to tell the adapter exactly the username and password that the user has filled in on the form.

```

// do the authentication
$auth = Zend_Auth::getInstance();
$result = $auth->authenticate($authAdapter);

```

To actually authenticate, we call the authenticate() function of Zend\_Auth. This ensures that the results of the authentication are stored into the session automatically for us.

```

if ($result->isValid()) {
    // success : store database row to auth's storage
    // system. (not the password though!)
    $data = $authAdapter->getResultRowObject(null,
        'password');
    $auth->getStorage()->write($data);
    $this->_redirect('/');
}

```

In the case of success, we store the full database row (except the password!) into the Zend\_Auth singleton. This ensures we can collect the user's name for display in the footer.

```

} else {
    // failure: clear database row from session
    $this->view->message = 'Login failed.';
}
}

```

In the case of authentication failure, we set the message so that the user knows what has happened.

The authentication process for logging in is now complete.

## Logging out

Logging out is much simpler than logging in as all we need to do is tell the Zend\_Auth singleton to clear its data. This is done in a new action logoutAction() within the AuthController so that we can navigate to <http://zftutorial/auth/logout> so that the user is logged out:

### zf-tutorial/application/controllers/AuthController.php:

```

class AuthController extends Zend_Controller_Action
{
    ...
    function logoutAction()
    {
        Zend_Auth::getInstance()->clearIdentity();
    }
}

```

```

        $this->_redirect('/');
    }
}

```

The `logoutAction()` function is so trivial, I can't think of anything to say about it!

We now need to provide the user with a link so that they can log out of the application. This is easiest in the footer. We will also tell the user their name, so that they can be confident that they are logged in correctly. Their name is stored in the `real_name` field of the users table and so that is now available in the `Zend_Auth` instance. First thing to do is to get it to the view, which we do in the `init()` function of `IndexController()`:

**zf-tutorial/application/controllers/IndexController.php:**

```

class IndexController extends Zend_Controller_Action
{
    function init()
    {
        $this->initView();
        Zend_Loader::loadClass('Album');
        $this->view->baseUrl = $this->_request->getBaseUrl();
        $this->view->user = Zend_Auth::getInstance()->getIdentity();
    }
    ...
}

```

It's certainly convenient that `Zend_Auth` is a singleton, otherwise we'd have stored it in the registry, by now!

We now need to add some HTML to `footer.phtml`:

**zf-tutorial/application/views/scripts/footer.phtml:**

```

<?php if($this->user) : ?>
<p id="logged-in">Logged in as <?php
    echo $this->escape($this->user->real_name);?>.
<a href="<?php echo $this->baseUrl ?>/auth/logout">Logout</a></p>
<?php endif; ?>
</div>
</body>
</html>

```

This HTML should look quite familiar as there are no new concepts here. We use `escape()` to ensure that the user's real name is safely displayed correctly and we use `baseUrl` to set the anchor's href to the correct place.

That's all that's required for logout.

## Protecting the Actions

All that's left is to ensure that no other actions are accessible if you are not logged in. To do that, we need to add some code to the `preDispatch()` function of `IndexController`.

**zf-tutorial/application/controllers/IndexController.php:**

```

class IndexController extends Zend_Controller_Action
{
    ...
    function preDispatch()
    {
        $auth = Zend_Auth::getInstance();
        if (!$auth->hasIdentity()) {
            $this->_redirect('auth/login');
        }
    }
    ...
}

```

```
}
```

`preDispatch()` is called before every action in the controller. We collect the `Zend_Auth` instance and then its `hasIdentity()` function tells us if we a user is logged in. If we don't then we redirect to the `auth/login` action.

And that's all there is to it!

## Conclusion

This concludes our brief look at integrating `Zend_Auth` into an MVC application. Clearly there is a lot more you could do with `Zend_Auth` and there are many ways to improve the way the code works, especially if you have multiple controllers to be protected. Note that we haven't looked in detail at authorisation here as that is covered by the `Zend_Acl` components. `Zend_Acl` would be used in conjunction with `Zend_Auth` for providing different levels of access to actions or data, but that is the subject of another tutorial.

I hope that you found it interesting and informative. If you find anything that's wrong, please email me at [rob@akrobat.com](mailto:rob@akrobat.com).