

Tutorial

Zugriffssteuerung im Zend Framework mit Zend_Auth und Zend_Acl

Zend Framework liefert mit Zend_Acl und Zend_Auth leistungsstarke Klassen, mit denen sich eine Zugriffssteuerung sehr schnell und einfach umsetzen lässt. Ich möchte an einem praktischen Beispiel zeigen, wie eine solche Implementierung aussehen kann. Die Benutzer sind dabei in einer (MySQL-)Tabelle gespeichert.



Bettina Ramm

www.die-web-architektin.de

Stand: 17.08.2009

Inhaltsverzeichnis

Voraussetzungen zum Verständnis.....	3
Zend_Acl – Wer darf was?.....	3
Ressourcen.....	3
Rollen.....	4
Regeln.....	4
Der Zend_Auth Adapter – wo sind denn hier die User?.....	4
Access Control – wer bist du und was willst du hier?.....	5
Einbindung des Plugins in der bootstrap-Datei.....	7
Login-Aktion – wer bist du?.....	7
Last but not least – das Logout: Und Tschüss.....	8

Fragen, Anmerkungen, Unklarheiten, Fehler gefunden?

Ich freue mich auf Ihr Feedback an b.ramm@die-web-architektin.de

Voraussetzungen zum Verständnis

Damit Sie dieses Tutorial verstehen, sollten Sie Grundkenntnisse im Umgang mit dem Zend Framework haben, mit den Action-Controllern des ZF vertraut sein, sowie PHP Kenntnisse mit Objektorientierung haben (was ja ohnehin schon Voraussetzung für die Arbeit mit dem Zend Framework ist).

Die Zugriffssteuerung im Zend Framework wird mit den Komponenten `Zend_Auth` und `Zend_Acl` realisiert. `Zend_Auth` übernimmt dabei die Authentifizierung, also die Sicherung der Identität des Benutzers, `Zend_Acl` definiert die Zugriffsrechte, also auf welche Bereiche der Webapplikation der angemeldete Benutzer welchen Zugriff hat.

Zend_Acl – Wer darf was?

ACL steht für Access Control List, das ist also eine Liste, die den Zugriff auf bestimmte Ressourcen (z. B. Bereiche oder Funktionen der Webapplikation) festlegt.

`Zend_Acl` liefert bereits alle nötigen Funktionen, so dass die Festlegung der Regeln mit wenigen Zeilen erledigt werden kann. Dazu erstellen wir eine neue Klasse, die von `Zend_Acl` abgeleitet wird.

```
class Plugin_Auth_Acl extends Zend_Acl
{
    public function __construct()
    {
        // RESSOURCES
        $this->add(new Zend_Acl_Resource('admin'));
        $this->add(new Zend_Acl_Resource('redaktion'));

        // ROLES
        $this->addRole(new Zend_Acl_Role('guest'));
        $this->addRole(new Zend_Acl_Role('redakteur'), 'guest');
        $this->addRole(new Zend_Acl_Role('admin'), 'redakteur');

        // RULES
        $this->allow(null, null);
        $this->deny('guest', 'redaktion');
        $this->deny('guest', 'admin');
        $this->allow('redakteur', 'redaktion');
        $this->allow('admin', 'admin');
    }
}
```

Der Name der Klasse ist natürlich beliebig. Ich lege alle meine Klassen für die Zugriffssteuerung im Verzeichnis `Plugin/Auth/` ab, woraus der Klassenname resultiert.

Ressourcen

Im Konstrktor werden Ressourcen, Rollen und Regeln definiert, wobei die Ressource wohl meist am schwersten greifbar ist. Während jeder Benutzer eine eindeutige Rolle hat, und sich daraus die Regeln ableiten, sind Ressourcen die Bereiche, für die die Regeln gelten. Diese Bereiche können je nach Webanwendung unterschiedlich definiert werden. Am einfachsten ist es, einfach den Controller oder das Modul als Ressource festzulegen. Dann kann für jeden Controller bzw. für jedes Modul eine Zugriffsregel erstellt werden. Für die meisten Anwendungen ist dies ausreichend.

In unserem Beispiel gibt es ein Modul `admin`, auf das nur der Administrator zugreifen darf, und ein

Modul redaktion, auf das Redakteure und Administratoren Zugriff haben.

Wir definieren also zwei Ressourcen: admin und redaktion (die Zuordnung zu den Modulen machen wir später) und drei Rollen: admin, redakteur – und guest für den Rest der Welt. Jeder User, der nichts anderes nachweist, wird automatisch als guest eingestuft – aber auch dazu später mehr.

Rollen

Rollen können hierarchisch aufgebaut sein, d. h. sie können Rechte voneinander erben. Daher macht es Sinn, die Rollen mit den wenigsten Rechten (in unserem Fall guest) zuerst zu definieren, und andere davon abzuleiten wie im obigen Beispiel. redakteur darf alles, was guest darf und admin darf alles, was redakteur darf (und mehr).

Regeln

Anschließend werden die Rechte oder sogenannte Regeln definiert. Dazu kennt Zend_Acl zwei Methoden: allow() und deny(). Beide Methoden haben zwei Parameter – der erste definiert was und der zweite wer. null steht hier nicht für nichts, sondern für alles bzw. alle.

In unserem obigen Beispiel wird also erstmal allen alles erlaubt. Dann wird Gästen der Zugriff auf redaktion und admin entzogen. Da Redakteure von Gästen erben, und Administratoren von Redakteuren, haben beide Gruppen ebenfalls keinen Zugriff mehr auf diese Ressourcen. Sie müssen also explizit wieder Zugriff erhalten. Das tun wir mit den letzten beiden Zeilen.

Das war's – unsere Zugriffsregeln sind gesetzt.

Der Zend_Auth Adapter – wo sind denn hier die User?

Für den Einsatz von Zend_Auth benötigen wir zunächst einen Adapter, der festlegt, wo die Daten der User herkommen. So stellt Zend Framework z. B. Adapter für LDAP oder Open ID Authentifizierung zur Verfügung. In unserem Fall pflegen wir aber unsere eigene kleine Userliste in einer MySQL-Datenbank.

Wir leiten unsere Auth-Adapter Klasse also von Zend_Auth_Adapter_DbTable ab.

```
class Plugin_Auth_AuthAdapter extends Zend_Auth_Adapter_DbTable
{
    public function __construct()
    {
        $registry = Zend_Registry::getInstance();
        parent::__construct($registry->dbAdapter);

        $this->setTableName('user');
        $this->setIdentityColumn('username');
        $this->setCredentialColumn('password');
        $this->setCredentialTreatment('PASSWORD(?));
    }
}
```

Im Konstruktor wird zunächst der DB-Adapter übergeben, den Sie in der Regel in der Registry abgelegt haben. Dann müssen für den Datenbank-Zugriff verschiedene Parameter gesetzt werden (siehe oben). Die Methode setTableName() setzt dabei den Tabellennamen, in dem die User gelistet sind. setIdentityColumn() gibt den Namen der Spalte an, über die der User identifiziert wird. Das kann der Benutzername sein, oder die E-Mail Adresse – je nachdem, wie Sie das handhaben möchten. Schließlich setzt setCredentialColumn() die Passwort-Spalte und setCredentialTreatment() die Behandlung des Passwortes (falls es verschlüsselt ist). Es handelt sich hierbei um die Datenbank-

Funktion für die Verschlüsselung, wobei das Fragezeichen als Platzhalter für die Daten steht. Im obigen Fall wird also geprüft, ob der Wert der Spalte password dem übermittelten und durch die PASSWORD-Funktion der Datenbank verschlüsselten Passwort entspricht.

Später wird übrigens der gesamte Datensatz des Users über Zend_Auth verfügbar sein, wenn der Abgleich der Login-Daten mit der Datenbank geklappt hat (meine Tabelle user muss mindestens zwei Spalten haben: username und password – genauer gesagt, drei Spalten, denn in der dritten ist die Rolle festgelegt – dazu kommen wir gleich).

Access Control – wer bist du und was willst du hier?

Schließlich benötigen wir eine Klasse, die die Authentifizierung und Autorisierung übernimmt.

Diese Klasse nennen wir Plugin_Auth_AccessControl und leiten sie von Zend_Controller_Plugin_Abstract ab. Wenn Sie sich bereits mit Plugins auskennen, wissen Sie, dass diese ganz einfach in die Applikation eingebunden und ihre Methoden vom Front Controller bei jedem Aufruf der Website in ganz bestimmter Reihenfolge abgearbeitet werden.

Ich möchte hier die Grundlagen nicht behandeln, Sie finden im [Manual des Zend Framework](#) eine ausführliche Abhandlung dazu.

Der Vorteil der AccessControl als Plugin liegt auf der Hand: Die Prüfung von Authentizität und Autorisierung erfolgt bei jedem Website-Aufruf und Sie müssen nichts weiter tun, als Ihre Rollen, Ressourcen und Regeln zu pflegen. Den Rest übernimmt unser Plugin.

Die Klasse ist etwas umfangreicher, genauer gesagt besteht sie aus zwei Methoden plus Konstruktor, weshalb ich sie in Schritten erklären möchte.

Der Konstruktor erhält eine Zend_Acl Instanz und eine Zend_Auth Instanz und speichert diese in klasseneigenen Variablen:

```
public function __construct(Zend_Auth $auth, Zend_Acl $acl)
{
    $this->_auth = $auth;
    $this->_acl = $acl;
}
```

Die Methode routeStartup() wird vor Starten des Routers aufgerufen und prüft, ob mit dem Aufruf Login-Daten übermittelt wurden.

```
public function routeStartup(Zend_Controller_Request_Abstract $request)
{
    if (!$this->_auth->hasIdentity() &&
        null !== $request->getPost('login_user') &&
        null !== $request->getPost('login_password')) {

        // POST-Daten bereinigen
        $filter = new Zend_Filter_StripTags();
        $username = $filter->filter($request->getPost('login_user'));
        $password = $filter->filter($request->getPost('login_password'));

        if (empty($username)) {
            $message = 'Bitte Benutzernamen angeben.';
        } elseif (empty($password)) {
            $message = 'Bitte Passwort angeben.';
        } else {
            $authAdapter = new Plugin_Auth_AuthAdapter();
            $authAdapter->setIdentity($username);
        }
    }
}
```

```

    $authAdapter->setCredential($password);

    $result = $this->_auth->authenticate($authAdapter);

    if (!$result->isValid()) {
        $messages = $result->getMessages();
        $message = $messages[0];
        echo $message; exit;
    } else {
        $storage = $this->_auth->getStorage();
        // die gesamte Tabellenzeile in der Session speichern,
        // wobei das Passwort unterdrückt wird
        $storage->write($authAdapter->getResultRowObject(null, 'password'));
    }
}

$registry = Zend_Registry::getInstance();
$view = $registry->view;
if (isset($message)) {
    $view->message = $message;
}
}
}

```

Die Funktion wird nur durchlaufen, wenn noch keine Identität bekannt ist und Post-Daten (login_user und login_password) übermittelt wurden. Diese werden bereinigt und wenn Username und Passwort angegeben wurden, gegen die Usertabelle geprüft. Das übernimmt unser Auth-Adapter für uns. Die eigentliche Zugriffssteuerung erfolgt hier noch nicht. Es wird lediglich geprüft, ob das Login korrekt war, und wenn ja, merkt sich Zend_Auth den angemeldeten User und seine Rolle. Es erfolgt also die Authentifizierung.

Spannender wird es in der zweiten Methode preDispatch(), die durchlaufen wird, bevor der Dispatcher eine Aktion verarbeitet. Hier wird geprüft, ob der User zu der Aktion überhaupt berechtigt ist, diese Aktion auszuführen. Das ist dann also die Autorisierung.

```

public function preDispatch(Zend_Controller_Request_Abstract $request)
{
    if ($this->_auth->hasIdentity() && is_object($this->_auth->getIdentity())) {
        $role = $this->_auth->getIdentity()->role;
    } else {
        $role = 'guest';
    }

    $module = $request->getModuleName();

    // Ressourcen = Modul -> kann hier geändert werden!
    $resource = $module;

    if (!$this->_acl->has($resource)) {
        $resource = null;
    }

    if (!$this->_acl->isAllowed($role, $resource, $action)) {
        if ($this->_auth->hasIdentity()) {

```

```

    // angemeldet, aber keine Rechte -> Fehler!
    $request->setModuleName('default');
    $request->setControllerName('error');
    $request->setActionName('noAccess');
  } else {
    // nicht angemeldet -> Login
    $request->setModuleName('default');
    $request->setControllerName('user');
    $request->setActionName('login');
  }
}
}
}

```

Zuerst wird geprüft, ob Zend_Auth den User kennt (siehe oben, Authentifizierung). Wenn nicht, erhält er die Standardrolle guest zugewiesen.

In der Zeile `$role = $this->_auth->getIdentity()->role;` wird übrigens auf den Inhalt der Spalte role in der User-Tabelle zugegriffen. Sie können auf diese Weise jede beliebige Spalte des User-Datensatzes auslesen, auch seinen Namen, seine Adresse, und was Sie sonst noch so von ihm gespeichert haben.

Dann wird das aktuelle Modul ausgelesen, dessen Name ja gleichzeitig der Name der Ressource ist. Sie können hier auch beliebig Controller-Name (`getControllerName()`) oder / und Action-Name (`getActionName()`) auslesen und weiterverarbeiten.

Über die Zend_Acl Methode `isAllowed()` wird nun geprüft, ob die Rolle und die Ressource zusammenpassen. Wenn dies nicht der Fall ist, gibt es zwei Möglichkeiten: Der User hat keine Rechte (ist aber authentifiziert), dann wird er weitergeleitet an die Aktion noAccess des Error-Controllers. Oder er ist gar nicht erst authentifiziert, dann wird er an die Login-Aktion der Applikation geleitet.

Einbindung des Plugins in der bootstrap-Datei

Das Plugin wird in der bootstrap registriert, und wird somit mit jedem Aufruf der Website automatisch ausgeführt. Das regeln diese vier Zeilen:

```

$auth = Zend_Auth::getInstance();
$acl = new Plugin_Auth_Acl();
$frontController->registerPlugin(new Plugin_Auth_AccessControl($auth, $acl));
$frontController->setParam('auth', $auth);

```

Achtung! Zend_Acl und Zend_Auth implementieren das Singleton Pattern, das heißt, es gibt nur eine einzige Instanz in der gesamten Applikation. Daher kann nicht mit new eine neue Instanz erzeugt werden, sondern `getInstance()` liefert die Instanz zurück.

Login-Aktion – wer bist du?

Die Login-Aktion kann in jeden beliebigen Controller integriert werden. Sie müssen nur darauf achten, in der AccessControl Klasse die Weiterleitung entsprechend einzustellen.

Die Login-Aktion sieht dann wie folgt aus:

```

public function loginAction()
{
    $form = new Zend_Form();
    $form->setAction("")
        ->setMethod('post');
}

```

```

$user = new Zend_Form_Element_Text('login_user',
    array('label' => 'Benutzername', 'required' => true));

$password = new Zend_Form_Element_Password('login_password',
    array('label' => 'Passwort', 'required' => true));

$submit = new Zend_Form_Element_Submit('submit',
    array('label' => 'Anmelden'));

$form->addElements(array($user, $password, $submit));
$this->view->form = $form;
}

```

Das Formular kann dann einfach über die View-Variable `$form` in das Template eingebunden werden.

Die Aktion des Formulars ist übrigens leer – die Prüfung der Login-Daten erfolgt dank unseres Plugins ja ohnehin bei jedem Website-Aufruf. Ist die Aktion des Formulars leer, wird bei Absenden des Formulars die Ursprungs-URL wieder aufgerufen. Der User landet also direkt auf der Seite, auf die er eigentlich wollte, und auf die ihm der Zugriff verwehrt wurde.

Das bedeutet aber auch, dass der User immer wieder im Login-Formular landet, wenn er die URL dazu direkt eingegeben hat, da das Formular ihn nicht automatisch woandershin lotst.

Wir ändern also unsere Action-Methode um ein winziges Detail: Folgende Zeilen fügen wir ganz am Anfang der Methode (also vor `$form = new Zend_Form()`) ein.

```

$auth = Zend_Auth::getInstance();
if ($auth->hasIdentity()) {
    return $this->_forward('index');
}

```

Jetzt sehen Sie, warum es Sinn macht, dass es nur eine einzige `Zend_Auth` Instanz gibt. Wir können von überall aus unserer Applikation heraus darauf zugreifen und feststellen, ob und welcher User angemeldet ist. Wir prüfen hier mit `hasIdentity()`, ob ein User angemeldet ist, und wenn ja, leiten wir zur `indexAction()` des aktuellen Controllers weiter. Die Anzeige des Login-Formulars macht dann ja keinen Sinn.

Es geht übrigens auch kürzer: Wenn Sie nur eine einzige Anfrage an `Zend_Auth` haben, können Sie auch schreiben `Zend_Auth::getInstance()->hasIdentity()`.

Last but not least – das Logout: Und Tschüss

Natürlich brauchen wir auch noch eine Logout-Funktion, mit der sich der Benutzer wieder abmelden kann. Das ist aber ganz unspektakulär:

```

public function logoutAction()
{
    Zend_Auth::getInstance()->clearIdentity();
    $this->view->logout = true;
}

```

Über `clearIdentity()` werden die Userdaten automatisch aus dem Speicher gelöscht. Anschließend müssen wir dem View noch sagen, dass der User ausgeloggt wurde (und eine entsprechende Anzeige im Template erstellen). Fertig.