cakephp-annotation-control-list Documentation

Release 3.0.0

Jose Diaz-Gonzalez

February 05, 2016

Contents

A simple, annotation-based ACL System for CakePHP

CHAPTER 1

Background

For the CakePHP book I wrote, I thought it would make sense to showcase to users how they might come up with an alternative to the ACL system that comes with CakePHP. As annotations are an interesting way of adding attributes to actions - and it's relatively easy to modify during application development - I decided that a method to do so via annotations would be the way to go.

CHAPTER 2

Requirements

- PHP 5.6+
- CakePHP 3.2+

2.1 Installation

The only officialy supported method of installing this plugin is via composer.

2.1.1 Using Composer

View on Packagist, and copy the json snippet for the latest version into your project's `composer.json`. Eg, v. 3.0.0 would look like this:

```
{
    "require": {
        "josegonzalez/cakephp-annotation-control-list": "3.0.0"
    }
}
```

This plugin has the type `cakephp-plugin` set in its own `composer.json`, composer knows to install it inside your `/Plugins` directory, rather than in the usual vendors file. It is recommended that you add `/Plugins/Upload` to your .gitignore file. (Why? read this.)

2.1.2 Enable plugin

You need to enable the plugin your `config/bootstrap.php` file:

```
<?php
Plugin::load('Josegonzalez/AnnotationControlList');
```

If you are already using `Plugin::loadAll();`, then this is not necessary.

2.2 Role-based usage

The `AnnotationControlList` plugin has two modes of usage. The `role` mode requires no more configuration than an `@roles` annotation on your action.

2.2.1 Setup

Setup your `AuthComponent` to use the `AnnotationAuthorize` and `AnnotationFormAuthenticate` classes:

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('Auth', [
        'authenticate' => [
            'Josegonzalez/AnnotationControlList.AnnotationForm' => [
                'passwordHasher' => 'Blowfish',
                'roleField' => 'role',  // `roleField` is `role` by default
            ]
        ],
        'authorize' => [
            'Josegonzalez/AnnotationControlList.Annotation',
            'roleField' => 'role',  // `roleField` is `role` by default
        ],
    ]);
}
```

2.2.2 Requiring roles for a given action

Annotate your methods with the roles you want to allow:

```
/**
 * @roles all
 */
public function index() {}

/**
 * @roles authenticated
 */
public function add() {}

/**
 * this is a list of roles
 * @roles anonymous, some_other_role
 */
public function register() {}

/**
 * this is also a list of roles
 * @roles ["admin", "a_special_role"]
 */
public function admin() {}
```

You can specify one or more roles in any of the above formats. If no role is specified for an action, then no user will be allowed access.

2.2.3 Special Roles

The following roles have a special meaning:

- `all`: All users will have this role

- `anonymous`: Users that have not yet authenticated against your app will have this role
- `authenticated`: Users that have been authenticated fall in this role

2.2.4 Available Classes

The following classes are available for your convenience:

- `AnnotationAuthorize`
- `AnnotationBasicAuthenticate`
- `AnnotationDigestAuthenticate`
- `AnnotationFormAuthenticate`

These extend the core classes and override the following methods:

- `isAuthorized`
- `getActionRoles`
- `getPrefixedAnnotations`
- `getAnnotations`
- `processRoles`
- `authorize`
- `unauthenticated`
- `getController`
- `prefix`

2.2.5 Custom Authenticate Classes

The `AnnotationFormAuthenticate` class extends `FormAuthenticate` to override the `unauthorized()` method, allowing us to use the annotations to define access even if the user has not yet authenticated. You can follow this pattern for any Authenticate class you create/use by adding the following to either your custom authenticate class or a subclass of one of the core classes:

```
use AnnotationParserTrait;
```

2.3 Model-based usage

The `AnnotationControlList` plugin has two modes of usage. The `model` mode requires more configuration than the `role` mode, but also allows you to extend access control to include information from your database records.

2.3.1 Setup

Setup your `AuthComponent` to use the `AnnotationAuthorize` and `AnnotationFormAuthenticate` classes:

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('Auth', [
        'authenticate' => [
            'Josegonzalez/AnnotationControlList.ModelForm' => [
                'passwordHasher' => 'Blowfish',
                'roleField' => 'role',  // `roleField` is `role` by default
            ]
        ],
        'authorize' => [
            'Josegonzalez/AnnotationControlList.Model',
            'roleField' => 'role',  // `roleField` is `role` by default
        ],
    ]);
}
```

2.3.2 Requiring roles for a given action

Annotate your methods with the roles you want to allow:

```
/**
 * @isAuthorized.roles all
 */
public function index() {}

/**
 * @isAuthorized.roles authenticated
 */
public function add() {}

/**
 * this is a list of roles
 * @isAuthorized.roles anonymous, some_other_role
 */
public function register() {}

/**
 * this is also a list of roles
 * @isAuthorized.roles ["admin", "a_special_role"]
 */
public function admin() {}

/**
 * Only allows authenticated users access if the finder returns data
 * @isAuthorized.roles authenticated
 * @isAuthorized.table Post
 * @isAuthorized.find active
 */
public function active_post() {
}

/**
 * Only allows authenticated users access if the Post.check_active()
 * method returns data
 *
 * @isAuthorized.roles authenticated
 * @isAuthorized.table Post
 * @isAuthorized.method check_active
 */
public function active_post() {
}

/**
 * Only allows authenticated users access if the finder returns data
 *
 * If the authenticated user's "group" field is "admin", then they are
 * allowed access without further database checks
 *
 * @isAuthorized.roles authenticated
 * @isAuthorized.always ["group", "admin"]
 * @isAuthorized.table Post
 * @isAuthorized.find active
 */
public function always_if_admin() {
}

/**
 * Only allows authenticated users access if the finder returns data
 *
 * If the authenticated user's "group" field is "admin", then they are
 * allowed access without further database checks
 *
 * If the user's "group" field matches the "Post.group_name", then they are
 * allowed access, otherwise they are denied access. You can have multiple
 * "if" conditions, and if any are true, then access is granted
 * @isAuthorized.roles authenticated
 * @isAuthorized.always ["group", "admin"]
 * @isAuthorized.table Post
 * @isAuthorized.find edit
 * @isAuthorized.conditions.if ["group", "Post.group_name"]
 */
public function edit_post() {
}
```

When a *Model::find()* is called, the current request parameters - as well as the `user_id` - are passed into the find as options. This can be used to further limit the data being retrieved. If an alternative model method is specified, then the current request parameters and `user_id` are passed in as the first argument.

You can specify one or more roles in any of the above formats. If no role is specified for an action, then no user will be allowed access.

2.3.3 Special Roles

The following roles have a special meaning:

- `all`: All users will have this role
- `anonymous`: Users that have not yet authenticated against your app will have this role
- `authenticated`: Users that have been authenticated fall in this role

2.3.4 Available Classes

The following classes are available for your convenience:

- `ModelAuthorize`
- `ModelBasicAuthenticate`
- `ModelDigestAuthenticate`
- `ModelFormAuthenticate`

These extend the core classes and override the following methods:

- `isAuthorized`
- `getActionRoles`
- `getPrefixedAnnotations`
- `getAnnotations`
- `processRoles`
- `authorize`
- `unauthenticated`
- `getController`
- `prefix`
- `performCheck`
- `checkAlwaysRule`
- `checkIfRules`
- `getData`
- `getFinder`
- `missingFinder`
- `ensureList`
- `isAssoc`

2.3.5 Custom Authenticate Classes

The `AnnotationFormAuthenticate` class extends `FormAuthenticate` to override the `unauthorized()` method, allowing us to use the annotations to define access even if the user has not yet authenticated. You can follow this pattern for any Authenticate class you create/use by adding the following to either your custom authenticate class or a subclass of one of the core classes:

```
use ModelParserTrait;
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search