the user is assigned with a role that contains that permission.

Both roles and permissions can be organized in a hierarchy. In particular, a role may consist of other roles or permissions, and a permission may consist of other permissions. Yii implements a partial-order hierarchy, which includes the more special tree hierarchy. While a role can contain a permission, it is not true vice versa.

For testing permissions, we have created two actions. The first action, test, contains checkers for created permissions and roles. The second action is delete, which is limited through the access filter. The rule for the access filter contains the following code:

[

'allow' => true,

'actions' => ['delete'],

'roles' => ['deletePost'],

],

This means that we are allowing all users who have the deletePost permission to run the deletePost action. Yii starts checking with the deletePost permission. Besides the fact that the access rule element is named as roles, you can specify an RBAC hierarchy node be it a role, rule, or permission. Checking for updatePost is complex:

Yii::$app->user->can('updatePost', ['post' => $post]);

We use a second parameter to pass a post (in our case, we have simulated it with stdclass). If a user is logged in as demo, then to get access we need to go from updatePost to author. If you’re lucky, you only have to go through updatePost, updateOwnPost, and author.

As updateOwnPost has a rule defined, it will be run with a parameter passed to checkAccess. If the result is true, then access will be granted. As Yii does not know what the shortest way is, it tries to check all possibilities until it is successful, or no alternatives are left.

There’s more...

There are some useful tricks that will help you to use RBAC efficiently, which are discussed in the following subsections.

**Keeping hierarchy simple and efficient**

Follow these recommendations where possible to maximize the performance and reduce hierarchy complexity:

• Avoid attaching multiple roles to a single user

• Don’t connect nodes of the same type; so, for example, avoid connecting one task to another

**Naming RBAC nodes**

A complex hierarchy becomes difficult to understand without using some kind of naming convention. One possible convention that helps to limit confusion is as follows: