Using controller filters

In many cases, we need to filter the incoming data or perform some actions based on the data. For  
example, with custom filters, we can filter visitors by IP, force users to use HTTPS, or redirect the user to  
an installation page prior to using the application.

In Yii2, filters are essentially a special kind of behavior, so using filters is the same as using behaviors.

Yii has a lot of built-in usable filters, which include:

* Core
* Custom
* Authentication
* Content Negotiator
* HttpCache
* PageCache
* RateLimiter
* Verb
* Cors

In this recipe, we will implement the following:

* Limiting access to the controller action to authorized users only
* Limiting access to the controller action to specified IPs
* Limiting access to specific user roles

Подготовка

1. Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
   <http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
   По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>**.**
2. Create app/components/AccessRule.php:

<?php

namespace app\components;  
use app\models\User;

class AccessRule extends \yii\filters\AccessRule {

/\* \*

\* @inheritdoc  
\*/

protected function matchRole($user)

{

if (empty($this->roles)) {  
return true;

}

$isGuest = $user->getIsGuest();  
foreach ($this->roles as $role) {  
switch($role) {  
case '?':

return ($isGuest) ? true : false;

case User::ROLE\_USER:

return (!$isGuest) ? true : false;  
case $user->identity->role: // Check if the user is logged in, and  
the roles match

return (!$isGuest) ? true : false;  
default:

return false;

}

}

return false;

}

}

3. Create app/controllers/AccessController.php as follows:

<?php

namespace app\controllers;  
use app\models\User;  
use Yii;

use yii\filters\AccessControl;  
use app\components\AccessRule;  
use yii\web\Controller;

class AccessController extends Controller  
{

public function behaviors()

{

return [

'access' => [

'class' => AccessControl::className(),

// We will override the default rule config with the new AccessRule

class

'ruleConfig' => [

'class' => AccessRule::className(),

],

'rules ' => [

[

'allow' => true,

'actions' => ['auth-only'],

'roles' => [User::ROLE\_USER]

],

[

'allow' => true,

'actions' => ['ip'],

'ips' => ['127.0.0.1'],

],

[

'allow' => true,

'actions' => ['user'],

'roles' => [ User::ROLE\_ADMIN],

],

[

'allow' => false,

]

],

]

];

}

public function actionAuthOnly()

{

echo "Looks like you are authorized to run me.";

}

public function actionIp()

{

echo "Your IP is in our list. Lucky you!";

}

public function actionUser()

{

echo "You're the right man. Welcome!";

}

}

4. Modify the User class as follows:

<?php

namespace app\models;

class User extends \yii\base\Object implements \yii\web\IdentityInterface  
{

// add roles contstants  
CONST ROLE\_USER = 200;

CONST ROLE\_ADMIN = 100;

public $id;  
public $username;  
public $password;  
public $authKey;  
public $accessToken;  
public $role;

private static $users = [

'100' => [

'id' => '100',

'username' => 'admin',

'password' => 'admin',

'authKey' => 'test100key',

'accessToken' => '100-token',

'role' => USER::ROLE\_ADMIN // add admin role for admin user

],

'101' => [

'id' => '101',

'username' => 'demo',

'password' => 'demo',

'authKey' => 'test101key',

'accessToken' => '101-token',

'role' => USER::ROLE\_USER // add user role for admin user

],

];

}

How to do it...

1. To use AccessControl, declare it in the behaviors() method of your controller class. We do this as  
follows:

public function behaviors()

{

return [

'access' => [

'class' => AccessControl::className(),

'rules' => [

[

'allow' => true,

'actions' => ['auth-only'],

' roles' => ['@'],

],

[

'allow' => true,

'actions' => ['ip'],

'ips' => ['127.0.0.1'],

],

[

'allow' => true,

'actions' => ['user'],

' roles ' => ['admin'],

],

[

'allow' => true,

'actions' => ['user'],

'matchCallback' => function ($rule, $action) {

return preg\_match('/MSIE 9/',$\_SERVER['HTTP\_USER\_AGENT']) !==

false;

}

],

['allow' => false]

],

]

];

}

2. Now try to run controller actions using Internet Explorer and other browsers by using both the  
admin and demo usernames.

How it works...

We will start with limiting access to the controller action to authorized users only. See the following code in  
the rules array:

[

'allow' => true,

'actions' => ['auth-only'],

'roles' => [User::ROLE\_USER]

],

Each array here is an access rule. You can either use allow=true or allow=false for a deny rule. For  
each rule, there are several parameters.

By default, Yii does not deny everything, so consider adding [' allow' => false] to the end of your rules  
list if you need maximum security.

In our rule, we use two parameters. The first is the actions parameter, which takes an array of actions to  
which the rule will be applied. The second is the roles parameter, which takes an array of user roles to  
determine the users this rule applies to.

Yii2’s built in Access Control supports only two roles by default: guest (not logged in), represented by ?,  
and authenticated, represented by @.

With simple access controls, we can just **lim**it access to specific pages or controller actions based on the

login state. If users are not logged in when they visit these pages, Yii will redirect them to the login page.

Rules are executed one by one, starting from the top, until one matches. If nothing matches, then the  
action is treated as allowed.

The next task is to limit access to specific IPs. In this case, the following two access rules are involved:

[

'allow' => true,

'actions' => ['ip'],

'ips' => ['127.0.0.1'],

],

The first rule allows access to the IP action from a list of IPs specified. In our case, we are using a  
loopback address, which always points to our own computer. Try changing it to 127.0.0.2, for example, to  
see how it works when the address does not match. The second rule denies everything, including all other  
IPs.

Next, we **lim**it access to one specific user role, as follows:

[

'allow' => true,

'actions' => ['user'],

'roles' => [ User::ROLE\_ADMIN],

],

The preceding rule allows a user with a role equal to admin to run the user action. Therefore, if you log in  
as admin, it will let you in, but if you log in as demo, it will not.



We have overridden the standard AccessRule class on our own, which is located in the

components/AccessRule.php file. Inside our AccessRule class, we have overridden the matchRole  
method on our own, where we get and check the current user role and match it with roles from our rules.

Finally, we need to deny access to a specific browser. For this recipe, we are denying only Internet  
Explorer 9. The rule itself is put on top, so it executes first, as follows:

[

'allow' => true,

'actions' => ['user'],

'matchCallback' => function ($rule, $action) {

return preg\_match('/MSIE 9/',$\_SERVER['HTTP\_USER\_AGENT'])!== false;

}

],

The detection technique that we are using is not very reliable, as MSIE is contained in many other user  
agent strings. For a list of possible user agent strings, you can refer to <http://www.useragentstring.com/>.

In the preceding code, we used another filter rule property named ' matchCallback'. This property will  
apply only when functions which are described in this property return true.

Our function checks if the user agent string contains MSIE 9.0 sting. Depending on your requirements,  
you can specify any PHP code.

See also

In order to learn more about access control and filters, refer to the following:

* [http://www.viiframework.com/doc-2.0/guide-structure-filters.html](http://www.yiiframework.com/doc-2.0/guide-structure-filters.html)
* <http://www.yiiframework.com/doc-2.0/yii-filters-accesscontrol.html>
* [http://www.yiiframework.com/doc-2.0/yii-filters-accessrnle.html](http://www.yiiframework.com/doc-2.0/yii-filters-accessrule.html)
* [https://github.com/yiisoft/yii2/hloh/master/docs/guide/structure-filters.md](https://github.com/yiisoft/yii2/blob/master/docs/guide/structure-filters.md)
* [http://www.yiiframework.com/doc-2.0/guide-security-authorization.html#access-control-filter](http://www.yiiframework.com/doc-2.0/guide-security-authorization.html%23access-control-filter)
* The **Using RBAC** recipe