Setting up an author automatically

The Blameable behavior allows you to update one or more authors’ fields automatically. This is primarily used to populate data into the created\_by and updated\_by fields. Similar to the Timestamp behavior, you can easily specify some special parameters and essential events for this behavior.

Let us return to the example from the previous section. We also have posts in our blog application. For example, let’s assume that our blog model is called Blogpost. The model has author\_id, the field which points to who created this post, and updater\_id, the field which points to who updated it. We would like to populate these attributes automatically during the create/update model events. Now you can learn how to do it.

Getting ready

1. Create a new application using the Composer package manager, as described in the official guide at

[http://www.viiframework.com/doc-2.0/guide-start-installation.html](http://www.yiiframework.com/doc-2.0/guide-start-installation.html).

1. Set up the database connection and create a table named blog\_post, as follows:

DROP TABLE IF EXISTS 'blog\_post';

CREATE TABLE IF NOT EXISTS 'blog\_post' (

'id' INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,

'author\_id' INT(10) UNSIGNED DEFAULT NULL,

'updater\_id' INT(10) UNSIGNED DEFAULT NULL,

'title' VARCHAR(255) NOT NULL,

'text' TEXT NOT NULL,

PRIMARY KEY ('id')

);

1. Use Gii to create the BlogPost model for the blost\_post table.

How to do it...

1. Add the following behaviors method to models/BlogPost. php: <?php

namespace app\models; use Yii;

use yii\db\BaseActiveRecord;

/\*\*

* This is the model class for table "blog\_post".

\*

* @property integer $id
* @property integer $author\_id
* @property integer $updater\_id
* @property string $title
* @property string $text \*/

class BlogPost extends \yii\db\ActiveRecord {

/\*\*

\* @return array \*/

public function behaviors()

{

return [

[

'class' => 'yii\behaviors\BlameableBehavior',

'attributes' => [

BaseActiveRecord::EVENT\_BEFORE\_INSERT => 'author\_id', BaseActiveRecord::EVENT\_BEFORE\_UPDATE => 'updater\_id'

]

]

];

}

}

2. Create controllers/TestController.php as follows:

<?php

namespace app\controllers;

use app\models\BlogPost; use app\models\User; use Yii;

use yii\helpers\Html; use yii\helpers\VarDumper; use yii\web\Controller;

/\*\*

* Class TestController.
* @package app\controllers \*/

class TestController extends Controller {

public function actionIndex()

{

$users = new User();

$identity = $users->findIdentity(100);

Yii::$app->user->setIdentity($identity);

$blogPost = new BlogPost();

$blogPost->title = 'Very pretty title';

$blogPost->text = 'Success is not final, failure is not fatal...'; $blogPost->save();

return $this->renderContent(Html::tag('pre', VarDumper::dumpAsString( $blogPost->attributes

)));

}

}

3. That is it. Now, run test/index. You will get the following result:



How it works...

By default, the Blameable behavior populates the created\_by and updated\_by attributes, but we will make a change and set up our behavior according to our own fields.

We also specified model events [and fields in the model, so, during the model creation, author\_id will be](http://www.yiiframework.com/doc-2.0/yii-behaviors-blameablebehavior.html) populated. Similarly, during the model update, we will populate updater\_id.

What Blameable does is insert the current user id value into the created\_by and updated\_by fields during the create/update model events. This is a super-convenient way of doing things. Every time a model gets created or updated, we automatically fill out the essential fields.

This works out really well for little projects such as for large systems, where multiple users are admin and you need to keep track of who is doing what. You can also use this for frontend implementations, for example, if you had a blog\_comment table and you wanted to use this method to keep track of the author of a comment. Also, you could set the author’s fields in the controller, but the behavior helps you to avoid writing unnecessary and additional code. This is a very effective and easy way to implement this thing.

There’s more.

Sometimes we need to fill out author\_id and updater\_id by an id other than that of the current user. In such a case, we may detach our behavior as follows:

$model->detachBehavior('blammable ' );

We can detach any behavior we like in this way.

See also

For further information, refer to <http://www.yiiframework.com/doc-2.0/yii-behaviors->blameablebehavior.html