Automating timestamps

For instance, we have a simple blog application. As in any blog, it has posts, comments, and so on. We would like to populate the timestamps during the create/update events for posts. Let us assume that our post model is named Blogpost model.

Getting ready

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

<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,

'title' VARCHAR(255) NOT NULL,

'text' TEXT NOT NULL,

'created\_date' INTEGER,

' modified\_date'INTEGER,

PRIMARY KEY ('id')

);

1. Use Gii to create a model for the blog\_post table.

How to do it...

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

/\*\*

* @return array \*/

public function behaviors()

{

return [

'timestamp'=> [

'class' => 'yii\behaviors\TimestampBehavior', 'createdAtAttribute' => 'creation\_date', 'updatedAtAttribute' => 'modified\_date'

]

];

}

1. Create controllers/TestController.php as follows:

<?php

namespace app\controllers;

use app\models\BlogPost; use yii\helpers\Html; use yii\helpers\VarDumper; use yii\web\Controller;

/\*\*

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

class TestController extends Controller

public function actionIndex()

{

{

$blogPost = new BlogPost();

$blogPost->title = 'Gotcha!';

$blogPost->text = 'We need some laughter to ease the tension of holiday shopping . ' ;

$blogPost->save();

return $this->renderContent(Html::tag('pre',

VarDumper::dumpAsString($blogPost->attributes)

));

}

}

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

How it works...

By default, the Timestamp behavior populates created\_at (the timestamp which points to the time when the model was created) and updated\_at (the time when the model was updated). It’s a standard practice to name these fields, but if we would like to make a change, we can specify fields, which will be updated, and model events.

There’s more.

For instance, our fields are named creation\_date and modified\_date.

Let’s configure our model with behavior according to these fields. In addition, we should add our behavior’s code to our Post model:

<?php

namespace app\models; use Yii;

use yii\db\BaseActiveRecord;

class Post extends \yii\db\ActiveRecord {

// . .

public function behaviors()

{

return [

[

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

'attributes' => [

BaseActiveRecord::EVENT\_BEFORE\_INSERT => 'creation\_date', BaseActiveRecord::EVENT\_BEFORE\_UPDATE => 'modified\_date',

]

]

];

}

// ..

}

In this example, we’ve pointed to the creation\_date and modified\_date attributes before creating and updating our model accordingly by dint of using special ActiveRecord events: event\_before\_insert and

EVENT\_BEFORE\_UPDATE.

In addition...

You may want to save the timestamp for custom scenarios. Let’s say you want to update the last\_login field, for example, for a specific controller action. In this situation, you can trigger the timestamp update for your specific attribute using the following:

$model->touch('last\_login');

Be aware that touch() can’t be used for new models. You will get InvalidCallException in this case:

$model = new Post();

$model->touch('creation\_date ' );

The touch () method calls model saving inside itself so you don’t need to save the model after calling it.

See also

[For further information, refer to http://www.yiiframework.com/doc-2.0/gurde-concept-](http://www.yiiframework.com/doc-2.0/guide-concept-behaviors.html%23using-timestampbehavior) behaviors.html#using-timestampbehav'or.