Создание моделей поведения

Есть много подобных решений в современных веб-приложениях. Ведущие продукты, такие как Google Gmail, определяют хорошие шаблоны пользовательского интерфейса. Один из них-мягкое удаление. Вместо постоянного удаления с тоннами подтверждений, Gmail позволяет нам сразу пометить сообщения как удаленные, а затем легко отменить его. Такое же поведение можно применить к любому объекту, например к записям в блоге, комментариям и т. д.

Давайте создадим поведение, которое позволит отмечать модели как удаленные, восстанавливать модели, выбирать еще не удаленные модели, удаленные модели и все модели. В этом рецепте мы будем следовать тестовому подходу к разработке для планирования поведения и тестирования, если реализация верна.

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. Создание двух баз данных для работы и тестирования.
3. Настройте Yii для использования первой базы данных в основном приложении в config/db.php. Убедитесь, что тестовое приложение использует вторую базу данных в tests/codeception/config/config.php.
4. Создание новой миграции:

<?php

use yii\db\Migration;

class m160427\_103115\_create\_post\_table extends Migration  
{

public function up()

{

$this->createTable('{{%post}}', [

'id' => $this->primaryKey(),

'title' => $this->string()->notNull(),

'content\_markdown' => $this->text(),

'content\_html' => $this->text(),

]);

}

public function down()

{

$this->dropTable('{{%post}}');

}

}

1. Применить миграции рабочей и тестовой базы данных:

./yii migrate

tests/codeception/bin/yii migrate

1. Создание модели Post :

<?php

namespace app\models;

use app\behaviors\MarkdownBehavior;  
use yii\db\ActiveRecord;

/\*\*

* @property integer $id
* @property string $title
* @property string $content\_markdown
* @property string $content\_html  
  \*/

class Post extends ActiveRecord  
{

public static function tableName()

{

return '{{%post}}';

}

public function rules()

{

return [

[['title'], 'required'],

[['content\_markdown'], 'string'],

[['title'], 'string', 'max' => 255],

];

}

}

Как это сделать...

Давайте сначала подготовим тестовую среду, начиная с определения креплений для модели Post. Создать файл tests/codeception/unit/fixtures/PostFixture.php:

<?php

namespace app\tests\codeception\unit\fixtures;

use yii\test\ActiveFixture;

class PostFixture extends ActiveFixture  
{

public $modelClass = 'app\models\Post';

public $dataFile = '@tests/codeception/unit/fixtures/data/post.php';

}

1. Файл добавить данные фикстуры для tests/codeception/unit/fixtures/data/post.php:

<?php  
return [

[

'id' => 1,

'title' => 'Post 1',

'content\_markdown' => 'Stored \*markdown\* text 1',

'content\_html' => "<p>Stored <em>markdown</em> text 1</p>\n",

],

];

1. Затем, нам нужно создать тест, tests/codeception/unit/MarkdownBehaviorTest.php:  
   <?php

namespace app\tests\codeception\unit;  
use app\models\Post;

use app\tests\codeception\unit\fixtures\PostFixture;  
use yii\codeception\DbTestCase;

class MarkdownBehaviorTest extends DbTestCase  
{

public function testNewModelSave()

{

$post = new Post();

$post->title = 'Title';

$post->content\_markdown = 'New \*markdown\* text';  
$this->assertTrue($post->save());

$this->assertEquals("<p>New <em>markdown</em> text</p>\n", $post->content\_html);

}

public function testExistingModelSave()

{

$post = Post::findOne(1);

$post->content\_markdown = 'Other \*markdown\* text';  
$this->assertTrue($post->save());

$this->assertEquals("<p>Other <em>markdown</em> text</p>\n", $post->content\_html);

}

public function fixtures()

{

return [

'posts' => [

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

]

];

}

}

1. Запустите unit тесты:

codecept run unit MarkdownBehaviorTest  
Ensure that tests has not passed:

Codeception PHP Testing Framework v2.0.9

Powered by PHPUnit 4.8.27 by Sebastian Bergmann and contributors.

Unit Tests (2)

Trying to test... MarkdownBehaviorTest::testNewModelSave Error

Trying to test. MarkdownBehaviorTest::testExistingModelSave Error

Time: 289 ms, Memory: 16.75MB

1. Теперь нам нужно реализовать поведение, присоединить его к модели и убедиться, что тест пройден. Создайте новый каталог, поведение. В этом каталоге создайте класс MarkdownBehavior:

<?php

namespace app\behaviors;

use yii\base\Behavior;  
use yii\base\Event;

use yii\base\InvalidConfigException;  
use yii\db\ActiveRecord;  
use yii\helpers\Markdown;

class MarkdownBehavior extends Behavior  
{

public $sourceAttribute;  
public $targetAttribute;

public function init()

{

if (empty($this->sourceAttribute) || empty($this->targetAttribute)) {  
throw new InvalidConfigException('Source and target must be set.');

}

parent::init();

}

public function events()

{

return [

ActiveRecord::EVENT\_BEFORE\_INSERT => 'onBeforeSave',

ActiveRecord::EVENT\_BEFORE\_UPDATE => 'onBeforeSave',

];

}

public function onBeforeSave(Event $event)

{

if ($this->owner->isAttributeChanged($this->sourceAttribute)) {  
$this->processContent();

}

}

private function processContent()

{

$model = $this->owner;

$source = $model->{$this->sourceAttribute};

$model->{$this->targetAttribute} = Markdown::process($source);

}

}

1. Давайте прикрепим поведение к модели Post:

class Post extends ActiveRecord  
{

public function behaviors()

{

return [

'markdown' => [

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

'sourceAttribute' => 'content\_markdown',

'targetAttribute' => 'content\_html',

],

];

}

}

1. Запустите тест и убедитесь, что он проходит:

Codeception PHP Testing Framework v2.0.9

Powered by PHPUnit 4.8.27 by Sebastian Bergmann and contributors.

Unit Tests (2)

Trying to test... MarkdownBehaviorTest::testNewModelSave Ok

Trying to test. MarkdownBehaviorTest::testExistingModelSave Ok

Time: 329 ms, Memory: 17.00MB

1. Вот и все. Мы создали многоразовое поведение и можем использовать его для всех будущих проектов, просто подключив его к модели.

Как это работает...

Начнем с теста. Поскольку мы хотим использовать набор моделей, мы определяем fixtures. Набор fixtures помещается в “базу данных " при каждом выполнении метода тестирования.

Мы готовим модульные тесты для определения того, как должно работать поведение:

* Во-первых, мы тестируем обработку контента новой модели. Поведение должно преобразовать текст Markdown из исходного атрибута в HTML и сохранить второй в целевой атрибут.
* Во-вторых, мы проводим тестирование для обновления содержания существующей модели. После изменения содержимого Markdown и сохранения модели мы должны получить обновленное содержимое HTML.

Теперь перейдем к интересным деталям реализации. В behavior мы можем добавить собственные методы, которые будут смешаны с моделью, к которой прикреплено поведение. Кроме того, мы можем подписаться на события компонента владельца. Мы используем его, чтобы добавить свой слушатель:

public function events()

{

return [

ActiveRecord::EVENT\_BEFORE\_INSERT => 'onBeforeSave',

ActiveRecord::EVENT\_BEFORE\_UPDATE => 'onBeforeSave',

];

}

Теперь мы можем реализовать этот слушатель:

public function onBeforeSave(Event $event)

{

if ($this->owner->isAttributeChanged($this->sourceAttribute))

{

$this->processContent();

}

}

Во всех методах мы можем использовать свойство owner для получения объекта, к которому прикреплено поведение. В общем, мы можем присоединить любое поведение к нашим моделям, контроллерам, приложениям и другим компонентам, которые расширяют класс yii\base\Component. Кроме того, мы можем многократно присоединять одно поведение к модели для обработки различных атрибутов:

class Post extends ActiveRecord  
{

public function behaviors()

{

return [

[

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

'sourceAttribute' => 'description\_markdown',

'targetAttribute' => 'description\_html',

],

[

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

'sourceAttribute' => 'content\_markdown',

'targetAttribute' => 'content\_html',

],

];

}

}

Кроме того, мы можем расширить класс yii\base\AttributeBehavior как yii\behaviors\TimestampBehavior для обновления заданных атрибутов для любых событий.

Смотрите так же

Дополнительные сведения о поведении и событиях см. на следующих страницах:

* <http://www.yiiframework.com/doc-2.0/guide-concept-behaviors.html>
  1. по русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/concept-behaviors>
* <http://www.yiiframework.com/doc-2.0/guide-concept-events.html>

по русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/concept-events>

Дополнительные сведения о синтаксисе Markdown см. в разделе [http://daringfireball net/proje cts/ma rkdown/](http://daringfireball.net/projects/markdown/).

Кроме того, обратитесь к созданию расширений распределения готовый рецепт этой главы.