Создание компонентов

Если у вас есть какой-то код, который выглядит так, будто его можно использовать повторно, но Вы не знаете, является ли это поведением, виджетом или чем-то еще, скорее всего, это компонент. Компонент должен быть унаследован от класса yii\base\component. Позже компонент может быть присоединен к приложению и настроен с помощью раздела components файла конфигурации. Это основное преимущество по сравнению с использованием простого класса PHP. Кроме того, мы получаем поддержку поведения, событий, геттеров и сеттеров.

В нашем примере мы реализуем простой компонент приложения Exchange, который сможет получать курсы валют из http://fixer.io сайт, прикрепите его к приложению и используйте.

Подготовка

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>.

Как это сделать ...

Для получения курсов валют наш компонент должен отправить запрос HTTP GET на URL-адрес службы, например  
<http://api.fixer.io/2016-05-14?base=USD>.

Сервис должен вернуть все Поддерживаемые тарифы в ближайший рабочий день:

{

"base":"USD",

"date":"2016-05-13",

"rates": {

"AUD":1.3728,

"BGN":1.7235,

"ZAR":15.168,

"EUR":0.88121

}

}

Компонент должен извлечь валюту из ответа в формате JSON и вернуть целевой курс:

1. Создайте каталог компонентов в структуре приложения.
2. Создайте пример класса компонента со следующим интерфейсом:

<?php

namespace app\components;

use yii\base\Component;

class Exchange extends Component  
{

public function getRate($source, $destination, $date = null)

{

}

}

3. Реализуйте функционал компонента:

<?php

namespace app\components;

use yii\base\Component;

use yii\base\InvalidConfigException;

use yii\base\InvalidParamException;

use yii\caching\Cache;

use yii\di\Instance;

use yii\helpers\Json;

class Exchange extends Component  
{

/\*\*

* @var string remote host  
  \*/

public $host = 'http://api.fixer.io';

/\*\*

* @var bool cache results or not  
  \*/

public $enableCaching = false;

/\*\*

* @var string|Cache component ID

\*/

public $cache = 'cache';

public function init()

{

if (empty($this->host)) {

throw new InvalidConfigException('Host must be set.');

}

if ($this->enableCaching) {

$this->cache = Instance::ensure($this->cache, Cache::className());

}

parent::init();

}

public function getRate($source, $destination, $date = null)

{

$this->validateCurrency($source);

$this->validateCurrency($destination);

$date = $this->validateDate($date);

$cacheKey = $this->generateCacheKey($source, $destination, $date);  
if (!$this->enableCaching || ($result = $this->cache->get($cacheKey)) ===

false) {

$result = $this->getRemoteRate($source, $destination, $date);  
if ($this->enableCaching) {

$this->cache->set($cacheKey, $result);

}

}

return $result;

}

private function getRemoteRate($source, $destination, $date)

{

$url = $this->host . '/' . $date . '?base=' . $source;

$response = Json::decode(file\_get\_contents($url));  
if (!isset($response['rates'][$destination])) {

throw new \RuntimeException('Rate not found.');

}

return $response['rates'][$destination];

}

private function validateCurrency($source)

{

if (!preg\_match('#A[A-Z]{3}$#s', $source)) {

throw new InvalidParamException('Invalid currency format.');

}

}

private function validateDate($date)

{

if (!empty($date) && !preg\_match('#\d{4}\-\d{2}-\d{2}#s', $date)) {  
throw new InvalidParamException('Invalid date format.');

}

if (empty($date)) {

$date = date('Y-m-d');

}

return $date;

}

private function generateCacheKey($source, $destination, $date)

{

return [ CLASS , $source, $destination, $date];

}

}

1. Подключите компонент к конфигурации config/console.php или config/web.php:

'components' => [

'cache' => [

'class' => 'yii\caching\FileCache',

],

'exchange' => [

'class' => 'app\components\Exchange',

'enableCaching' => true,

],

// ...

db' => $db,

],

1. Прямо сейчас мы можем использовать новый компонент напрямую или с помощью метода get:  
   echo \Yii::$app->exchange->getRate('USD', 'EUR');

echo \Yii::$app->get('exchange')->getRate('USD', 'EUR', '2014-04-12');

1. Создание демонстрационного контроллера консоли:

<?php

namespace app\commands;

use yii\console\Controller;

class ExchangeController extends Controller  
{

public function actionTest($currency, $date = null)

{

echo \Yii::$app->exchange->getRate('USD', $currency, $date) . PHP\_EOL;

}

}

1. Теперь попробуйте выполнить любую команду:

$ ./yii exchange/test EUR

* 0.90196

$ ./yii exchange/test EUR 2015-11-24

* 0.93888

$ ./yii exchange/test OTHER

* Exception 'yii\base\InvalidParamException' with message 'Invalid currency  
  format.'

$ ./yii exchange/test EUR 2015/24/11

Exception 'yii\base\InvalidParamException' with message 'Invalid date format.'

$ ./yii exchange/test ASD

* Exception 'RuntimeException' with message 'Rate not found.'

В результате вы должны увидеть значения курса в успешных случаях или конкретные исключения в ошибках. Помимо создания собственных компонентов, вы можете сделать больше.

Переопределение существующих компонентов приложения

В большинстве случаев нет необходимости создавать собственные компоненты приложения, так как другие типы расширений, такие как виджеты или поведения, охватывают почти все типы многократно используемых кодов. Тем не менее, переопределение основных компонентов платформы является обычной практикой и может использоваться для настройки поведения платформы для ваших конкретных потребностей без взлома ядра.

Например, чтобы иметь возможность форматировать числа с помощью метода Yii::app()->formatter->asNumber($value) вместо нашего метода NumberHelper:: format из рецепта создания помощников, вы можете выполнить следующие шаги:

1. Расширьте компонент Yii\i18n\Formatter следующим образом:

<?php

namespace app\components;

class Formatter extends \yii\i18n\Formatter  
{

public function asNumber($value, $decimal = 2)

{

return number\_format($value, $decimal, '.', ',');

}

}

1. Переопределить в классе встроенный форматер компонент:

'components' => [

// ...

formatter => [

'class' => 'app\components\Formatter,

],

// ...

],

1. Сейчас мы можем использовать этот метод напрямую:

echo Yii::app()->formatter->asNumber(1534635.2, 3);

Кроме того, его можно использовать в качестве нового формата для виджетов GridView и DetailView:

<?= \yii\grid\GridView::widget([

'dataProvider' => $dataProvider,

'columns' => [

'id',

'created\_at:datetime',

'title',

'value:number',

],

]) ?>

4. Кроме того, можно расширить каждый существующий компонент, не перезаписывая его исходный код.

Как это работает...

Чтобы иметь возможность присоединить компонент к приложению,его можно расширить из класса yii\base\component. Прикрепление так же просто, как добавление нового массива в раздел компонентов конфигурации. Там значение класса указывает класс компонента, а все остальные значения устанавливаются для компонента через соответствующие открытые свойства компонента и методы setter.

Сама реализация очень проста; мы завершаем http://api.fixer.io вызывает удобный API с валидаторами и кэшированием. Мы можем получить доступ к нашему классу по его имени компонента с помощью Yii:: $app. В нашем случае это будет Yii:: $app- > exchange.

Смотрите так же

[Для официальной информации о компонентах, см. http://www.yiiframework.com/doc-2.0/guide-concept-](http://www.yiiframework.com/doc-2.0/guide-concept-components.html)components.html.

По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/structure-application-components>

Для источников класса NumberHelper см. рецепт создания помощников.