



Technologia i rozwiązania

Tworzenie aplikacji z Yii

Receptury

Poznaj najlepsze przepisy dla Yii!



Alexander Makarov

[PACKT]
PUBLISHING

Tytuł oryginału: Yii Application Development Cookbook, Second Edition

Tłumaczenie: Joanna Zatorska

ISBN: 978-83-246-8596-7

Copyright © Packt Publishing 2013.

First published in the English language under the title “Yii Application Development Cookbook – Second Edition”

© Helion 2014.

All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/twapyi.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/twapyi>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	7
O recenzentach	8
Wstęp	9
Rozdział 1. Dla zaawansowanych	13
Wstęp	13
Używanie getterów i setterów	13
Używanie zdarzeń Yii	16
Korzystanie z importowania i automatycznego ładowania	23
Używanie wyjątków	26
Konfiguracja komponentów	29
Konfiguracja domyślnych ustawień widżetów	32
Używanie kolekcji platformy Yii	34
Obsługa żądań	37
Rozdział 2. Trasy, kontroler i widoki	41
Wstęp	41
Konfigurowanie reguł URL	42
Generowanie adresów URL według ścieżek	45
Używanie wyrażeń regularnych w regułach URL	49
Tworzenie reguł URL dla stron statycznych	52
Przekazywanie własnych reguł URL w trakcie działania programu	55
Użycie kontrolerów bazowych	59
Użycie zewnętrznych akcji	60
Wyświetlanie stron statycznych przy użyciu metody CViewAction	64

Użycie wiadomości typu flash	66
Użycie kontekstu kontrolera w widoku	67
Ponowne wykorzystanie widoków poprzez widoki częściowe	69
Użycie klipów	71
Użycie dekoratorów	73
Definiowanie kilku układów	74
Dzielenie danych na strony i sortowanie danych	76
Rozdział 3. Ajax i jQuery	79
Wstęp	79
Wczytywanie bloku z wykorzystaniem technologii AJAX	79
Zarządzanie zasobami	85
Dołączanie zasobów do strony	90
Korzystanie z formatu JSON	93
Przekazywanie konfiguracji z PHP do JavaScript	96
Obsługa zróżnicowanej liczby danych wejściowych	98
Renderowanie treści po stronie klienta	104
Rozdział 4. Używanie formularzy	119
Wstęp	119
Pisanie własnych walidatorów	119
Przesyłanie plików na serwer	122
Dodawanie CAPTCHA	126
Dostosowywanie CAPTCHA	131
Tworzenie własnego widżetu do pobierania danych z wykorzystaniem CHtmlWidget	133
Rozdział 5. Testowanie aplikacji	137
Wstęp	137
Przygotowanie środowiska testowego	138
Pisanie i uruchamianie testów jednostkowych	141
Używanie konfiguracji testów	146
Testowanie aplikacji z wykorzystaniem testów funkcjonalnych	152
Generowanie raportów pokrycia	156
Rozdział 6. Baza danych, aktywny rekord i triki związane z modelem	161
Wstęp	161
Pobieranie danych z bazy danych	162
Definiowanie kilku połączeń z bazami danych i korzystanie z nich	167
Używanie podzbiorów do uzyskania modeli dla różnych języków	170
Przetwarzanie pól modelu metodami przypominającymi zdarzenia aktywnego rekordu	173
Stosowanie języka markdown i HTML	175
Podświetlanie kodu przez Yii	178
Automatyzacja znaczników czasu	183
Automatyczne ustawianie autora	185

Implementacja odwzorowania dziedziczenia do pojedynczej tabeli	187
Używanie CDbCriteria	191
Rozdział 7. Używanie komponentów Zii	193
Wstęp	193
Używanie dostawców danych	194
Używanie siatek	200
Używanie list	207
Tworzenie niestandardowych kolumn siatek	212
Rozdział 8. Rozszerzanie Yii	219
Wstęp	219
Tworzenie zachowań modeli	219
Tworzenie komponentów	225
Tworzenie akcji kontrolerów do wielokrotnego użycia	229
Tworzenie kontrolerów wielokrotnego użytku	232
Tworzenie widżetów	236
Tworzenie poleceń CLI	238
Tworzenie filtrów	242
Tworzenie modułów	244
Niestandardowe renderowanie widoków	250
Przygotowywanie rozszerzeń do dystrybucji	254
Rozdział 9. Obsługa błędów, debugowanie i logowanie komunikatów	259
Wstęp	259
Używanie różnych tras przekierowania logów	260
Analizowanie stosu wywołań błędów Yii	266
Logowanie komunikatów i używanie informacji kontekstowych	268
Implementowanie własnej inteligentnej procedury obsługi błędu 404	272
Rozdział 10. Bezpieczeństwo	277
Wstęp	277
Używanie filtrów kontrolerów	277
Używanie CHtml i CHtmlPurifier do zapobiegania atakom XSS	282
Zapobieganie atakom typu SQL injection	286
Zapobieganie atakom CSRF	291
Używanie RBAC	294
Opis działania	297
Rozdział 11. Dostrajanie wydajności	303
Wstęp	303
Przestrzeganie najlepszych praktyk	303
Przyspieszanie obsługi sesji	307
Używanie łańcuchów zależności pamięci podręcznej	311

Profilowanie aplikacji z wykorzystaniem Yii	317
Wykorzystywanie buforowania HTTP	327
Rozdział 12. Używanie zewnętrznego kodu	333
Wstęp	333
Korzystanie z platformy Zend Framework w Yii	334
Dostosowywanie mechanizmu automatycznego ładowania w Yii	338
Korzystanie z platformy Kohana w Yii	342
Korzystanie z PEAR w Yii	349
Korzystanie z narzędzia Composer w Yii	351
Rozdział 13. Wdrażanie	357
Wstęp	357
Zmiana układu folderów Yii	357
Przenoszenie aplikacji poza folder główny	360
Udostępnianie folderu platformy	362
Przenoszenie fragmentów konfiguracji do oddzielnych plików	364
Używanie kilku konfiguracji do uproszczenia wdrażania	367
Implementowanie i wykonywanie zadań cron	371
Tryb konserwacji	373
Skorowidz	376

Rozszerzanie Yii

W tym rozdziale zostaną poruszone następujące zagadnienia:

- Tworzenie zachowań modeli.
- Tworzenie komponentów.
- Tworzenie akcji kontrolerów do wielokrotnego użycia.
- Tworzenie kontrolerów do wielokrotnego użycia.
- Tworzenie widżetu.
- Tworzenie poleceń CLI.
- Tworzenie filtrów.
- Tworzenie modułów.
- Niestandardowe renderowanie widoków.
- Przygotowywanie rozszerzeń do dystrybucji.

Wstęp

W tym rozdziale pokażę nie tylko, jak implementować własne rozszerzenie Yii, ale także jak przygotować rozszerzenie do wielokrotnego użycia i do użytku przez społeczność. Ponadto omówię wiele zagadnień, które należy rozważyć, aby nasze rozszerzenie było możliwie wydajne.

Tworzenie zachowań modeli

W dzisiejszych aplikacjach sieci WWW spotykamy wiele podobnych rozwiązań. Wiodące produkty, takie jak Google Gmail, definiują ciekawe wzorce interfejsu użytkownika. Jednym z nich

jest **nietrwale usuwanie**. Zamiast ostatecznego usuwania poprzedzonego koniecznością wielu potwierdzeń Gmail pozwala użytkownikowi na natychmiastowe oznaczenie wiadomości jako usuniętych, a następnie na ich łatwe przywrócenie. To samo zachowanie można zastosować do dowolnego obiektu, takiego jak posty bloga, komentarze itd.

Utwórzmy zachowanie, które pozwoli na oznaczanie modeli jako usuniętych, przywracanie modeli i zaznaczanie jeszcze nie usuniętych, usuniętych oraz wszystkich modeli. W tej recepturze do zaplanowania zachowania i przetestowania, czy implementacja jest poprawna, zastosujemy podejście test-driven development.

Przygotowanie

Wykonaj poniższe czynności:

1. Utwórz bazę danych i dodaj do niej tabelę post:

```
CREATE TABLE `post` (
  `id` int(11) NOT NULL auto_increment,
  `text` text,
  `title` varchar(255) default NULL,
  `is_deleted` tinyint(1) NOT NULL default '0',
  PRIMARY KEY (`id`)
)
```

2. Skonfiguruj Yii, aby korzystał z tej bazy danych w aplikacji głównej (*protected/config/main.php*).
3. Upewnij się, że aplikacja test ma takie same ustawienia (*protected/config/test.php*).
4. Usuń komentarz z komponentu fixture w ustawieniach aplikacji test.
5. Skorzystaj z Gii do wygenerowania modelu Post.

Jak to zrobić...

1. Przygotujmy środowisko testowe, zaczynając od zdefiniowania konfiguracji testów dla modelu Post w pliku *protected/tests/fixtures/post.php*:

```
<?php
return array(
    array(
        'id' => 1,
        'title' => 'post1',
        'text' => 'post1',
        'is_deleted' => 0,
    ),
    array(
        'id' => 2,
        'title' => 'post2',
        'text' => 'post2',
    )
)
```



```

        'is_deleted' => 1,
    ),
    array(
        'id' => 3,
        'title' => 'post3',
        'text' => 'post3',
        'is_deleted' => 0,
    ),
    array(
        'id' => 4,
        'title' => 'post4',
        'text' => 'post4',
        'is_deleted' => 1,
    ),
    array(
        'id' => 5,
        'title' => 'post5',
        'text' => 'post5',
        'is_deleted' => 0,
    ),
);

```

2. Musimy utworzyć przypadek testowy. Umieść następujący kod w pliku *protected/tests/unit/soft_delete/SoftDeleteBehaviorTest.php*:

```

<?php
class SoftDeleteBehaviorTest extends CDbTestCase
{
    protected $fixtures = array(
        'post' => 'Post',
    );

    function testRemoved()
    {
        $postCount = Post::model()->removed()->count();
        $this->assertEquals(2, $postCount);
    }

    function testNotRemoved()
    {
        $postCount = Post::model()->notRemoved()->count();
        $this->assertEquals(3, $postCount);
    }

    function testRemove()
    {
        $post = Post::model()->findPk(1);
        $post->remove()->save();
        $this->assertNull(Post::model()->notRemoved()->findPk(1));
    }
}

```

```

function testRestore()
{
    $post = Post::model()->findByPk(2);
    $post->restore()->save();
    $this->assertNotNull(Post::model()->notRemoved()->findByPk(2));
}

function testIsDeleted()
{
    $post = Post::model()->findByPk(1);
    $this->assertFalse($post->isRemoved());
    $post = Post::model()->findByPk(2);
    $this->assertTrue($post->isRemoved());
}
}

```

3. Następnie musimy zaimplementować zachowanie, dołączyć je do modelu i upewnić się, że test zakończy się powodzeniem. Utwórz nowy podfolder *soft_delete* w folderze *protected/extensions*. Umieść w tym podkatalogu nowy plik *SoftDeleteBehavior.php*. Najpierw dołącz zachowanie do modelu Post:

```

class Post extends CActiveRecord
{
    // ...

    public function behaviors()
    {
        return array(
            'softDelete' => array(
                'class' => 'ext.soft_delete.SoftDeleteBehavior'
            ),
        );
    }

    // ...
}

```

4. Teraz zaimplementuj klasę *protected/extensions/soft_delete/SoftDeleteBehavior.php*:

```

<?php
class SoftDeleteBehavior extends CActiveRecordBehavior
{
    public $flagField = 'is_deleted';

    public function remove()
    {
        $this->getOwner()->{$this->flagField} = 1;
        return $this->getOwner();
    }

    public function restore()
    {

```

```

        $this->getOwner()->{$this->flagField} = 0;
        return $this->getOwner();
    }

    public function notRemoved()
    {
        $criteria = $this->getOwner()->getDbCriteria();
        $criteria->compare($this->flagField, 0);
        return $this->getOwner();
    }

    public function removed()
    {
        $criteria = $this->getOwner()->getDbCriteria();
        $criteria->compare($this->flagField, 1);
        return $this->getOwner();
    }

    public function isRemoved()
    {
        return (boolean)$this->getOwner()->{$this->flagField};
    }
}

```

5. Uruchom test i upewnij się, że zakończył się pomyślnie.
6. To wszystko. Utworzyliśmy zachowanie wielokrotnego użycia, które możemy wykorzystać w kolejnych projektach jedynie poprzez dołączenie ich do modelu.

Opis działania

Zacznijmy od przypadku testowego. Ponieważ zamierzamy użyć zestawu modeli, definiujemy konfiguracje testów. Zestaw konfiguracji jest umieszczany w bazie danych za każdym razem, gdy wykonywana jest metoda testowa. Aby skorzystać z konfiguracji, klasa testu powinna być dziedziczona po klasie `CDbTestCase` i mieć zadeklarowaną właściwość prywatną `$fixtures`.

```

protected $fixtures = array(
    'post' => 'Post',
);

```

W powyższej definicji `post` to nazwa pliku zawierającego definicje konfiguracji testów, zaś `Post` to nazwa modelu, do którego zostaną zastosowane konfiguracje testów.

Najpierw testujemy własne zakresy `removed` i `notRemoved`. Powinniśmy ograniczyć rezultat wyszukiwania jedynie do usuniętych elementów, a następnie do elementów nieusuniętych. Ponieważ wiemy, jakie dane uzyskamy z konfiguracji testów, możemy w następujący sposób sprawdzić w testach liczbę elementów usuniętych i nieusuniętych:

```
$postCount = Post::model()->removed()->count();
$this->assertEquals(2, $postCount);
```

Następnie testujemy metody `remove` i `restore`. Metoda testowa `remove` ma następującą postać:

```
$post = Post::model()->findByPk(1);
$post->remove()->save();
$this->assertNull(Post::model()->notRemoved()->findByPk(1));
```

Element definiujemy poprzez identyfikator `id`, usuwamy go, a następnie próbujemy go odzyskać, korzystając z zakresu `notRemoved`. Ponieważ element został usunięty, powinniśmy w wyniku otrzymać `null`.

Na zakończenie testujemy metodę `isRemoved`, która po prostu zwraca wartość odpowiedniej kolumny w postaci boolowskiej wartości logicznej.

Teraz przejdźmy do interesujących szczegółów implementacji. Ponieważ implementujemy zachowanie modelu aktywnego rekordu, musimy skorzystać z rozszerzenia klasy `CActiveRecordBehavior`. W zachowaniu możemy dodać nasze własne metody. Zostaną one włączone do modelu, do którego dołączone będzie zachowanie. W ten sposób dodajemy metody znajdujące się w zakresach `remove/restore/isRemoved` i `removed/notRemoved`:

```
public function remove()
{
    $this->getOwner()->{$this->flagField} = 1;
    return $this->getOwner();
}
public function removed()
{
    $criteria = $this->getOwner()->getDbCriteria();
    $criteria->compare($this->flagField, 1);
    return $this->getOwner();
}
```

W obydwu metodach używamy metody `getOwner` do uzyskania obiektu, do którego dołączono zachowanie. W naszym przypadku jest to model, dlatego możemy pracować z jego danymi lub zmienić jego kryteria wyszukiwania. Zwracamy instancję modelu, aby umożliwić łańcuchowe wywołania metod w następujący sposób:

```
$post->remove()->save();
```

Dodatkowe informacje

Pozostało jeszcze kilka spraw, które należy omówić w niniejszej recepturze.

CActiveRecordBehavior i CModelBehavior

Niekiedy przydałoby się nieco więcej elastyczności w zachowaniu, na przykład aby umożliwić reagowanie na zdarzenia modelu. Zarówno w CActiveRecordBehavior, jak i w CModelBehavior dostępne są metody odpowiadające zdarzeniom, które można nadpisać, aby obsłużyć zdarzenia modelu. Przykładowo: jeśli chcemy obsłużyć w zachowaniu kaskadowe usuwanie, możemy to zrobić, nadpisując metodę `afterDelete`.

Więcej typów zachowań

Zachowania można dołączyć nie tylko do modelu, ale również do dowolnego komponentu. Każde zachowanie dziedziczy po klasie CBehavior, dlatego możemy użyć jego metod w następujący sposób:

- Metoda `getOwner` służy do uzyskania komponentu, do którego dołączono zachowanie.
- Metody `getEnabled` i `setEnabled` służą do sprawdzania, czy zdarzenie jest aktywne, i do ustawiania jego stanu.
- Metody `attach` i `detach` mogą być odpowiednio używane do inicjalizowania zdarzenia i czyszczenia tymczasowych danych, utworzonych podczas korzystania z zachowania.

Warto przeczytać

Dodatkowe informacje o zachowaniach można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/api/CActiveRecordBehavior>
- <http://www.yiiframework.com/doc/api/CModelBehavior>
- <http://www.yiiframework.com/doc/api/CBehavior>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie komponentów

Jeśli pewien kod w naszej aplikacji wygląda jak kod wielokrotnego użytku, ale nie wiemy, czy jest to zachowanie, widżet, czy coś innego, prawdopodobnie jest to komponent. Komponent powinien dziedziczyć po klasie CComponent lub CApplicationComponent. Po utworzeniu można go dołączyć do aplikacji i skonfigurować w pliku konfiguracyjnym `protected/config/main.php`. Jest to główna korzyść w porównaniu z używaniem czystej klasy PHP. Ponadto uzyskujemy wsparcie dla zachowań, zdarzeń, getterów i setterów.

W naszym przykładzie zaimplementujemy prosty komponent aplikacji EImageManager, który będzie mógł skalować obrazy, korzystając z biblioteki GD, dołączyć ją do aplikacji i jej użyć.

Przygotowanie

Aby wypróbować skalowanie obrazów, musimy zainstalować rozszerzenie PHP o nazwie GD.

Jak to zrobić...

1. Utwórz plik *protected/components/EImageManager.php*:

```
<?php
class EImageManager extends CApplicationComponent
{
    protected $image;
    protected $width;
    protected $height;

    protected $newWidth;
    protected $newHeight;

    public function resize($width = false, $height = false){
        if($width!==false) $this->newWidth = $width;
        if($height!==false) $this->newHeight = $height;

        return $this;
    }

    public function load($filePath)
    {
        list($this->width, $this->height, $type) = getimagesize($filePath);

        switch ($type)
        {
            case IMAGETYPE_GIF:
                $this->image = imagecreatefromgif($filePath);
                break;
            case IMAGETYPE_JPEG:
                $this->image = imagecreatefromjpeg($filePath);
                break;
            case IMAGETYPE_PNG:
                $this->image = imagecreatefrompng($filePath);
                break;
            default:
                throw new CException('Niewłaściwy typ obrazu ' . $type);
        }
    }
}
```

```

    return $this;
}

public function save($filePath)
{
    $ext = pathinfo($filePath, PATHINFO_EXTENSION);
    $newImage = imagecreatetruecolor($this->newWidth, $this->newHeight);
    imagecopyresampled($newImage, $this->image, 0, 0, 0, 0,
        $this->newWidth, $this->newHeight, $this->width,
        $this->height);
    switch($ext)
    {
        case 'jpg':
        case 'jpeg':
            imagejpeg($newImage, $filePath);
            break;
        case 'png':
            imagepng($newImage, $filePath);
            break;
        case 'gif':
            imagegif($newImage, $filePath);
            break;
        default:
            throw new CException("Niewłaściwy typ obrazu ", $ext);
    }

    imagedestroy($newImage);
    if(!is_file($filePath))
        throw new CException("Niepowodzenie zapisu.");
}

function __destruct()
{
    imagedestroy($this->image);
}
}

```

2. Następnie musimy dołączyć nasz komponent do aplikacji. W pliku *protected/config/main.php* musimy dodać następujący kod:

```

...
// application components
'components'=>array(
    'image' => array(
        'class' => 'EImageManager',
    ),
),
...

```

3. Teraz możemy użyć komponentu w następujący sposób:

```

Yii::app()->image
->load(Yii::getPathOfAlias('webroot').'/src.png')
->resize(100,100)
->save(Yii::getPathOfAlias('webroot').'/dst.png');

```

Opis działania

Aby można było dodać komponent do aplikacji, musi on dziedziczyć po klasie `CApplicationComponent`. Dołączanie jest tak proste jak dodawanie nowej tablicy do sekcji `component` w konfiguracji. Wartość `class` w tablicy określa klasę komponentu, zaś inne wartości są ustawiane poprzez odpowiadające im właściwości publiczne komponentu i `setter`y.

Sama implementacja jest bardzo prosta. Wywołania GD opakowujemy w wygodne API, zawierające metody `save`, `load` i `resize`. Aby można było wykonywać łańcuchowe wywołania API, `load` i `resize` zwracają sam komponent.

Korzystając z `Yii::app()`, możemy uzyskać dostęp do naszej klasy poprzez nazwę jej komponentu. W naszym przypadku będzie to `Yii::app()->image`.

Dodatkowe informacje

Oprócz tworzenia własnych komponentów możemy dokonać wiele więcej.

Nadpisywanie istniejących komponentów aplikacji

Przez większość czasu nie ma potrzeby tworzenia własnych komponentów aplikacji, ponieważ inne rodzaje rozszerzeń takich jak widżety czy zachowania zawierają niemal wszystkie rodzaje kodu wielokrotnego użytku. Jednakże nadpisywanie komponentów podstawowej platformy jest popularną praktyką i można go użyć do dostosowania zachowania platformy dla własnych potrzeb bez konieczności wprowadzania zmian w kodzie platformy.

Przykładowo: aby można było uzyskać rolę użytkownika z bazy danych z wykorzystaniem `Yii::app()->user->role`, możemy w następujący sposób zaimplementować dziedziczenie po komponencie `CWebUser`:

```

<?php
class WebUser extends CWebUser {
    private $_model = null;
    function getRole() {
        if($user = $this->getModel()){
            return $user->role;
        }
        else return 'guest';
    }
}

```



```

private function getModel(){
    if($this->_model === null){
        if($this->id === null) return null;
        $this->_model = User::model()->findByPk($this->id);
    }
    return $this->_model;
}
}

```

Aby zastąpić standardowy komponent użytkownika, należy dostosować plik konfiguracyjny *main.php*:

```

...
// application components
'components'=>array(
    'user'=>array(
        'class' => 'WebUser',
        // other properties
    ),
...

```

W powyższym kodzie określiliśmy nową klasę `class` dla komponentu `user`.

Warto przeczytać

Dodatkowe informacje o komponentach można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/api/CComponent/>
- <http://www.yiiframework.com/doc/api/CApplicationComponent/>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie akcji kontrolerów do wielokrotnego użycia

Popularne akcje, takie jak usuwanie modelu aktywnego rekordu według klucza prywatnego lub uzyskiwanie danych dla automatycznego uzupełniania z wykorzystaniem AJAX, można przenieść do akcji wielokrotnego użyciu i później w razie potrzeby dołączyć ją do kontrolera.

W tej recepturze utworzymy akcję do wielokrotnego użyciu `delete`, która będzie usuwać określony model aktywnego rekordu według jego klucza głównego.

Przygotowanie

1. Utwórz nową aplikację Yii, korzystając z polecenia `yiic webapp`.
2. Utwórz i skonfiguruj nową bazę danych.
3. Wykonaj następujące zapytanie SQL:

```
CREATE TABLE `post` (
  `id` int(11) NOT NULL auto_increment,
  `text` text,
  `title` varchar(255) default NULL,
  PRIMARY KEY (`id`)
);
CREATE TABLE `comment` (
  `id` int(11) NOT NULL auto_increment,
  `text` text,
  PRIMARY KEY (`id`)
);
```

4. Wygeneruj modele dla tabel `post` i `comment`, korzystając z Gii.

Jak to zrobić...

1. Utwórz plik `protected/extensions/actions/EDeleteAction.php` i umieść w nim następujący kod:

```
<?php
class EDeleteAction extends CAction
{
    public $modelName;
    public $redirectTo = array('index');

    /**
     * Uruchamia akcję.
     * Ta metoda jest wywoływana przez kontroler posiadający tę akcję.
     */
    public function run($pk)
    {
        ActiveRecord::model($this->modelName)->deleteByPk($pk);
        if(Yii::app()->getRequest()->getIsAjaxRequest())
        {
            Yii::app()->end(200, true);
        }
        else
        {
            $this->getController()->redirect($this->redirectTo);
        }
    }
}
```

2. Następnie dołączymy akcję do kontrolera *protected/controllers/DeleteController.php*:

```
<?php
class DeleteController extends CController
{
    public function actions()
    {
        return array(
            'deletePost' => array(
                'class' => 'ext.actions.EDeleteAction',
                'modelName' => 'Post',
                'redirectTo' => array('indexPosts'),
            ),
            'deleteComment' => array(
                'class' => 'ext.actions.EDeleteAction',
                'modelName' => 'Comment',
                'redirectTo' => array('indexComments'),
            ),
        );
    }

    public function actionIndexPosts()
    {
        echo "Jestem akcją index dla modelu Post.";
    }

    public function actionIndexComments()
    {
        echo " Jestem akcją index dla modelu Comment.";
    }
}
```

3. To wszystko. Teraz możesz usunąć post, odwiedzając stronę */delete/deletePost/pk/<pk>*, a także usunąć komentarz, odwiedzając */delete/deleteComment/pk/<pk>*. Po usunięciu zostaniemy przekierowani do odpowiedniej akcji *index*.

Opis działania

Aby utworzyć akcję zewnętrznego kontrolera, nasza klasa powinna dziedziczyć po klasie *CAction*. Jedyną metodą, którą należy zaimplementować, jest *run*. W naszym przypadku akceptuje ona parametr o nazwie *pk* ze zmiennej *\$_GET*, korzystając z mechanizmu automatycznego dołączania parametru dostępnego w Yii, i próbuje usunąć odpowiedni model.

Aby zapewnić możliwość dostosowywania komponentu, utworzyliśmy dwie publiczne właściwości, które można konfigurować poprzez kontroler. Jest to właściwość *modelName* przechowująca nazwę modelu, z którym pracujemy, oraz *redirectTo* określająca trasę, do której użytkownik zostanie przekierowany.

Sama konfiguracja odbywa się poprzez implementację metody `actions` w kontrolerze. Możemy tam dołączyć akcję raz lub kilka razy i skonfigurować jej właściwości publiczne.

Dodatkowe informacje

W klasie `CAction` zaimplementowane są dwie przydatne metody. Pierwsza to `getController`. Możemy jej użyć do pobrania instancji kontrolera, do którego dołączona jest akcja. Będzie nam to potrzebne do przekierowania do innej akcji lub na przykład do wygenerowania łańcucha URL.

Drugą metodą jest `getId` zwracająca nazwę akcji, określoną w metodzie `actions` kontrolera.

Warto przeczytać

Dodatkowe informacje o zewnętrznej akcji kontrolera można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/api/CAction/>
- <http://www.yiiframework.com/doc/api/CController/#actions-detail>

Zobacz też

- Receptura „Tworzenie kontrolerów wielokrotnego użytku”.
- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie kontrolerów wielokrotnego użytku

W Yii możemy tworzyć kontrolery wielokrotnego użytku. Jeśli rozwijamy wiele aplikacji lub kontrolerów tego samego rodzaju, przeniesienie całego wspólnego kodu do kontrolera wielokrotnego użytku pozwoli nam zaoszczędzić wiele czasu.

W tej recepturze utworzymy prosty kontroler wielokrotnego użytku o nazwie `api`, który zaimplementuje proste API JSON CRUD dla modelu. Będzie ono pobierać dane z metod POST i GET i odpowiadać danymi w formacie JSON oraz odpowiednim kodem odpowiedzi HTTP.

Przygotowanie

1. Utwórz nową aplikację Yii, korzystając z polecenia `yii create webapp`.
2. Utwórz nową bazę danych i wykonaj następujące zapytanie SQL:

```
CREATE TABLE `post` (
  `id` int(11) NOT NULL auto_increment,
  `text` text,
  `title` varchar(255) default NULL,
  PRIMARY KEY (`id`)
);
```

3. Skonfiguruj aplikację, aby korzystała z utworzonej bazy danych, i wygeneruj model, korzystając z Gii. W naszym przykładzie użyjemy modelu Post, jednak może to być dowolny model.

Jak to zrobić...

1. Utwórz plik *protected/extensions/json_api/JsonApiController.php* i umieść w nim następujący kod:

```
<?php
class JsonApiController extends CController
{
    const RESPONSE_OK = 'OK';
    const RESPONSE_NO_DATA = 'Brak danych';
    const RESPONSE_NOT_FOUND = 'Nie znaleziono';
    const RESPONSE_VALIDATION_ERRORS = 'Błędy walidacji';

    public $modelName;

    public function init()
    {
        parent::init();
        if(empty($this->modelName))
            throw new CException("Należy określić modelName przed użyciem
                JsonApiController.");
    }

    public function actionCreate()
    {
        if(empty($_POST))
            $this->respond(400, self::RESPONSE_NO_DATA);

        $model = new $this->modelName;
        $model->setAttributes($_POST);

        if($model->save())
            $this->respond(200, self::RESPONSE_OK);
        else
            $this->respond(400, self::RESPONSE_VALIDATION_ERRORS,
                $model->getErrors());
    }
}
```

```

public function actionGet($pk)
{
    $model = CActiveRecord::model($this->modelName)->findByPk($pk);
    if(!$model)
        $this->respond(404, self::RESPONSE_NOT_FOUND);
    $this->respond(200, self::RESPONSE_OK, $model->getAttributes());
}

public function actionUpdate($pk)
{
    if(empty($_POST))
        $this->respond(400, self::RESPONSE_NO_DATA);
    $model = CActiveRecord::model
        ($this->modelName)->findByPk($pk);
    if(!$model)
        $this->respond(404, self::RESPONSE_NOT_FOUND);

    $model->setAttributes($_POST);
    if($model->save())
        $this->respond(200, self::RESPONSE_OK);
    else
        $this->respond(400, self::RESPONSE_VALIDATION_ERRORS,
            $model->getErrors());
}

public function actionDelete($pk)
{
    if(CActiveRecord::model($this->modelName)->deleteByPk($pk))
    {
        $this->respond(200, self::RESPONSE_OK);
    }
    else {
        $this->respond(404, self::RESPONSE_NOT_FOUND);
    }
}

protected function respond($statusCode, $status, $data = array())
{
    $response['status'] = $status;
    $response['data'] = $data;
    echo CJSON::encode($response);
    Yii::app()->end($statusCode, true);
}
}

```

- Następnie musimy dołączyć kontroler do naszej aplikacji poprzez plik *protected/config/main.php*. Można to osiągnąć poprzez dodanie konfiguracji kontrolera do właściwości `controllerMap` klasy `CWebApplication`, dlatego musimy umieścić następujący kod na początku tablicy konfiguracji:

```

...
'controllerMap' => array(

```

```

    'api' => array(
        'class' => 'ext.json_api.JsonApiController',
        'modelName' => 'Post',
    ),
),
...

```

3. To wszystko. Dołączyliśmy kontroler i skonfigurowaliśmy go, aby można było pracować z modelem Post.

Potrzebne są nam formularze do przesłania danych, jednak jeśli mamy już jakieś dane, możemy użyć metod `get` bezpośrednio z URL, na przykład `/api/get/pk/1`. Aplikacje powinny zwracać dane w następującym formacie:

```

{"status":"OK","data":{"id":"1","text":"post1",
    "title":"post1","is_deleted":"0"}}

```

Opis działania

Gdy uruchomiona jest aplikacja i przekazujemy do niej trasę taką jak `api/get`, to zanim uruchomimy `ApiController::actionGet` Yii, musimy sprawdzić, czy zdefiniowany został `controllerMap`. Ponieważ zdefiniowany jest tu kontroler `api`, Yii uruchamia go, zamiast postępować w zwykły sposób.

W samym kontrolerze zdefiniowaliśmy właściwość `modelName`, aby można było połączyć się wielokrotnie z kontrolerem i uzyskać API dla kilku modeli. Ustawiamy ją, gdy dołączamy kontroler.

Sam kontroler nie różni się znacznie od zwykłego kontrolera, z wyjątkiem kilku szczegółów:

- Podczas pracy z aktywnym rekordem nową klasę tworzymy w następujący sposób:


```
$model = new $this->modelName;
```
- Model znajdujemy w następujący sposób:


```
$model = CActiveRecord::model($this->modelName);
```
- To pozwala nam unikać sztywnego przypisania do specyficznego modelu i zamiast tego określić nazwę modelu.
- Używamy CJSON, aby zakodować tablice i modele do formatu JSON.
- `Yii::app()->end()` służy do zakończenia wykonywania aplikacji ze zwróceniem określonego kodu odpowiedzi HTTP.

Dodatkowe informacje

Dodatkowe informacje o mapie kontrolera można znaleźć na następującej stronie API:

<http://www.yiiframework.com/doc/api/CWebApplication#controllerMap-detail>

Zobacz też

- Receptura „Tworzenie akcji kontrolerów wielokrotnego użytku”.
- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie widżetów

Widżet to część widoku do wielokrotnego użytku, która nie tylko renderuje pewne dane, ale również wykonuje to według określonej logiki. Może nawet pobierać dane z modeli i używać własnych widoków, dlatego przypomina zredukowaną używalną wersję modułu.

Utwórzmy widżet, który narysuje wykres kołowy z wykorzystaniem Google API.

Przygotowanie

Utwórz nową aplikację Yii, korzystając z polecenia `yiic webapp`.

Jak to zrobić...

1. Utwórz plik `protected/extensions/chart/EChartWidget.php`:

```
<?php
class EChartWidget extends CWidget
{
    public $title;
    public $data=array();
    public $labels=array();
    public function run()
    {
        echo "<img src=\"http://chart.apis.google.com/chart?cht=".urlencode(
            $this->title)."&cht=pc&chs=300x150&chd=".$this->encodeData(
                $this->data)."&chl=".implode('|', $this->labels)."\>";
    }
}
protected function encodeData($data)
{
    $maxValue=max($data);
    $chars='ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $chartData="s:";
    for($i=0;$i<count($data);$i++)
    {
        $currentValue=$data[$i];
        if($currentValue>-1)
            $chartData.=substr($chars,61*($currentValue/$maxValue),1);
    }
}
```



```

        else
            $chartData.='_';
        }
        return $chartData."&chxt=y&chxl=0:|0|".$maxValue;
    }
}

```

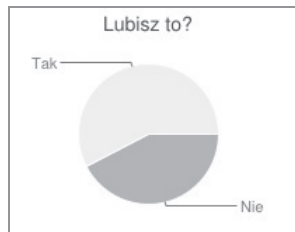
2. Następnie utwórz kontroler *protected/controllers/ChartController.php* i umieść w nim następujący kod:

```

<?php
class ChartController extends CController
{
    public function actionIndex()
    {
        $value = rand(10, 90);
        $this->widget('ext.chart.EChartWidget', array(
            'title' => 'Lubisz to?',
            'data' => array(
                $value, 100-$value
            ),
            'labels' => array(
                'Nie',
                'Tak',
            ),
        ));
    }
}

```

3. Teraz uruchom akcję *index* kontrolera. Powinieneś ujrzeć wykres kołowy podobny do poniższego:



Opis działania

Podobnie jak w każdym innym rodzaju rozszerzenia tworzymy pewne publiczne właściwości, które można skonfigurować podczas wywoływania widżetu metodą `CController::widget`. W tym przypadku konfigurujemy tytuł, zestaw danych oraz etykiety danych.

Główną metodą widżetu jest `run()`. W naszym widżecie generujemy URL wskazujący na API wykresów Google, a następnie wyświetlamy znacznik ``.

Dodatkowe informacje

Dodatkowe informacje o widżetach można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/api/CWidget/>
- <http://www.yiiframework.com/doc/api/CCaptcha/>

Zobacz też

- Receptura „Konfiguracja komponentów” w rozdziale 1. „Dla zaawansowanych”.
- Receptura „Konfiguracja domyślnych ustawień widżetów” w rozdziale 1. „Dla zaawansowanych”.
- Receptura „Tworzenie własnego widżetu do pobierania danych z wykorzystaniem CInputWidget” w rozdziale 4. „Używanie formularzy”.
- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie poleceń CLI

Yii wspiera polecenia wiersza poleceń i pozwala na tworzenie poleceń wielokrotnego użytku. Polecenia konsolowe są szybsze do utworzenia niż graficzny interfejs dla sieci WWW. Jeśli chcemy utworzyć pewien rodzaj użytecznego narzędzia dla aplikacji, która będzie używana przez deweloperów lub administratorów, polecenia konsolowe są najlepszym wyborem.

Aby sprawdzić, jak tworzyć polecenia konsolowe, utworzymy proste polecenie, które wyczyści kilka elementów, takich jak pamięć podręczna, tymczasowe foldery itd.

Przygotowanie

Utwórz nową aplikację Yii, korzystając z polecenia `yii c webapp`.

Jak to zrobić...

1. Utwórz plik `protected/extensions/clean_command/ECleanCommand.php` i umieść w nim następujący kod:

```
<?php
class ECleanCommand extends CConsoleCommand
{
    public $webRoot;
    public function actionCache()
    {
```

```

    $cache=Yii::app()->getComponent('cache');
    if($cache!==null){
        $cache->flush();
        echo "Gotowe.\n";
    }
    else {
        echo "Skonfiguruj komponent cache.\n";
    }
}

public function actionAssets()
{
    if(empty($this->webRoot))
    {
        echo "Podaj ścieżkę do folderu webRoot w parametrze polecenia.\n";
        Yii::app()->end();
    }
    $this->cleanDir($this->webRoot.'/assets');
    echo "Gotowe.\n";
}

public function actionRuntime()
{
    $this->cleanDir(Yii::app()->getRuntimePath());
    echo " Gotowe.\n";
}

private function cleanDir($dir)
{
    $di = new DirectoryIterator ($dir);
    foreach($di as $d)
    {
        if(!$d->isDot())
        {
            echo "Usunięto ".$d->getPathname()."\n";
            $this->removeDirRecursive($d->getPathname());
        }
    }
}

private function removeDirRecursive($dir)
{
    $files = glob($dir.DIRECTORY_SEPARATOR.'{,.*}', GLOB_MARK
| GLOB_BRACE);
    foreach ($files as $file)
    {
        if(basename($file) == '.' ||
basename($file) == '..')
            continue;
        if (substr($file,

```

```
- 1) == DIRECTORY_SEPARATOR)
        $this->removeDirRecursive($file);
    else
        unlink($file);
    }
    if (is_dir($dir))
        rmdir($dir);
}
}
```

2. Następnie musimy dołączyć polecenie do aplikacji konsolowej. Domyślnie aplikacja konsolowa korzysta z oddzielnego pliku *yiic.php*:

```
<?php
// change the following paths if necessary
$yiic=dirname(__FILE__).'/../../framework/yiic.php';
$config=dirname(__FILE__).'/config/console.php';
require_once($yiic);
```

3. Dlatego konfiguracja mieści się w pliku *protected/config/console.php*. Dodajmy nasze polecenie konsolowe do właściwości *commandMap*:

```
// This is the configuration for yiic console application.
// Any writable CConsoleApplication properties can be configured here.
return array(
    'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
    'name'=>'My Console Application',
    'commandMap' => array(
        'clean' => array(
            'class' => 'ext.clean_command.ECleanCommand',
            'webRoot' => 'ściezka/do/folderu/webroot/aplikacji',
        ),
    ),
);
```

4. Nie zapomnij zmienić elementu *webRoot* na rzeczywistą ścieżkę. To wszystko. Teraz przejdź do folderu *protected* i spróbuj uruchomić poniższe polecenia:

```
yiic clean
yiic clean cache
yiic clean assets
yiic clean runtime
```

Opis działania

Wszystkie polecenia konsolowe powinny dziedziczyć po klasie *CConsoleCommand*. Ponieważ wszystkie polecenia konsolowe są uruchamiane w klasie *CConsoleApplication*, a nie w klasie *CWebApplication*, nie ma sposobu na wyznaczenie głównego folderu aplikacji sieci WWW. W tym celu tworzymy konfigurowalną właściwość publiczną zwaną *webRoot*.

Struktura polecenia konsolowego przypomina typowy kontroler. Definiujemy kilka akcji, które można uruchomić poleceniem `yii <pojeczenie konsolowe> <akcja polecenia>`.

Jak widać, nie używamy żadnego widoku, dlatego możemy się skupić na zadaniach programistycznych zamiast na projekcie graficznym, kodzie itd. Nadal jednak musimy zwrócić przydatny wynik, aby użytkownicy wiedzieli, co się dzieje. Wykonywane jest to poprzez zwykłe instrukcje PHP echo.

Dodatkowe informacje

Jeśli polecenie jest dość złożone, podobnie jak `message` lub `migrate`, które są wbudowane w Yii, dobrą decyzją jest dostarczenie pewnych dodatkowych informacji na temat dostępnych opcji i akcji. Można tego dokonać poprzez nadpisanie metody `getHelp`.

```
public function getHelp()
{
    $out = "Polecenie clean pozwala wyczyścić różne dane tymczasowe Yii oraz
          wygenerowane przez aplikację.\n\n";
    return $out.parent::getHelp();
}
```

Po uruchomieniu samego polecenia `yii clean` uzyskamy następujący wynik:

```
Polecenie clean pozwala wyczyścić różne dane tymczasowe
Yii oraz wygenerowane przez aplikację.

Usage: yii clean <action>
Actions:
  cache
  assets
  runtime

C:\xampp\htdocs\example6_7\protected>
```

Warto przeczytać

Dodatkowe informacje o tworzeniu aplikacji i poleceń konsolowych można znaleźć na następujących stronach:

- <http://www.yiiframework.com/doc/guide/1.1/pl/topics/console>
- <http://www.yiiframework.com/doc/api/CConsoleCommand/>
- <http://www.yiiframework.com/doc/api/CConsoleApplication/>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie filtrów

Filtr to klasa, którą można uruchomić przed lub po wykonaniu akcji. Można jej użyć także do zmodyfikowania kontekstu wykonywania lub udekorowania wyniku. W naszym przykładzie zaimplementujemy prosty filtr wyniku, który skompresuje uzyskany kod HTML, usuwając wszystkie opcjonalne elementy formatowania.

Nie używaj tego filtra w produkcji. GZIP wykona to zadanie lepiej, oszczędzając przepustowość. Ponadto usuwanie formatowania obszaru preformatowanego nie jest bezpieczne.

Przygotowanie

Utwórz nową aplikację Yii, korzystając z polecenia `yii c webapp`.

Jak to zrobić...

1. Utwórz plik `protected/extensions/compress_html/ECompressHtmlFilter.php` i umieść w nim następujący kod:

```
<?php
class ECompressHtmlFilter extends CFilter
{
    protected function preFilter($filterChain)
    {
        ob_start();
        return parent::preFilter($filterChain);
    }

    protected function postFilter($filterChain)
    {
        $out = ob_get_clean();
        echo preg_replace("~>(\s+|\t+|\n+)<~", "><", $out);
        parent::postFilter($filterChain);
    }
}
```

2. To wszystko. Teraz musimy dołączyć filtr do aplikacji. Ponieważ mamy już SiteController, dodajmy do niego filtr, nadpisując metodę `filters`:

```
public function filters()
{
    return array(
        array(
```

```

        'ext.compress_html.ECompressHtmlFilter'
    ),
);
}

```

3. Teraz uruchom aplikację i sprawdź kod źródłowy HTML. Powinien to być pojedynczy wiersz tekstu, taki jak poniżej:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
dtd"><html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en"><head><meta http-equiv="Content-Type" content="text/
html; charset=utf-8" /><meta name="language" content="en" /><!--
blueprint CSS framework -->
...

```

Opis działania

Filtr powinien implementować przynajmniej interfejs `IFilter`, jednak ponieważ chcemy dokonać końcowego i wstępnego filtrowania, możemy bezpośrednio dziedziczyć po klasie `CFilter`. Planujemy uzyskać w pewien sposób zawartość wygenerowaną przez akcję i nieco przetworzyć wynik przed wysłaniem go do przeglądarki. Buforowanie danych wyjściowych PHP świetnie się nadaje do wykonania tego zadania, dlatego nadpisujemy metodę `preFilter`, aby rozpocząć buforowanie poprzez wywołanie w niej metody `ob_start`. Nadpisujemy też metodę `postFilter`, aby uzyskać zawartość bufora przy użyciu `ob_get_clean`, przetworzyć wynik z wykorzystaniem wyrażeń regularnych i go wyświetlić.

Zauważmy, że podczas nadpisywania metod `preFilter` i `postFilter` wywołujemy metody macierzyste. Pozwala to programiście na utworzenie łańcucha filtrów podczas konfigurowania kontrolera.

Dodatkowe informacje

Dodatkowe informacje o filtrach można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/api/CFilter>
- <http://www.yiiframework.com/doc/api/IFilter>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Tworzenie modułów

Jeśli utworzyłeś złożoną część aplikacji i chcesz ją wykorzystać po uprzednim dostosowaniu w następnym projekcie, prawdopodobnie powinieneś utworzyć moduł.

W tej recepturze zobaczysz, jak utworzyć moduł wiki. Dla uproszczenia nie będziemy się skupiać na zarządzaniu użytkownikami i uprawnieniami, lecz pozwolimy wszystkim na nieograniczoną edycję.

Przygotowanie

1. Utwórz nową aplikację Yii, korzystając z polecenia `yii ic webapp`.
2. Skonfiguruj bazę danych MySQL i wykonaj następujące zapytanie SQL:

```
CREATE TABLE `wiki` (
  `id` varchar(255) NOT NULL,
  `text` text NOT NULL,
  PRIMARY KEY (`id`)
)
```

3. Wygeneruj model Wiki, korzystając z Gii.
4. Przenieś plik *protected/models/Wiki.php* do folderu *protected/modules/wiki/models/*.
5. Dodaj wiki do sekcji `modules` pliku *protected/config/main.php*:

```
'modules'=>array(
  // uncomment the following to enable the Gii tool
  'gii'=>array(
    'class'=>'system.gii.GiiModule',
    'password'=>false,
  ),
  'wiki'
),
```

Jak to zrobić...

Najpierw zaplanujmy działanie:

- **wiki** to zestaw stron, na których można umieszczać łącza do innych stron na podstawie tytułu.
- Zwykle wiki używa prostszej składni, łatwiejszej do odczytu od kodu HTML.
- Jeśli użytkownik przejdzie do strony, która jeszcze nie istnieje, zostanie zachęony do jej utworzenia.
- Aby usunąć stronę, użytkownik musi zapisać ją wraz z pustą zawartością.

Bazując na powyższych wytycznych, za język znaczników przyjmijmy język markdown. Udobnym sposobem tworzenia łączy do innych stron na podstawie ich tytułu. Załóżmy, że składnia będzie miała postać `[[nazwa strony]]` lub `[[własny tytuł|nazwa strony]]`, jeśli potrzebne będzie łączy o niestandardowym tekście.

1. Najpierw dodajmy łączy wiki do klasy `CMarkdownParser`. Utwórz plik `protected/modules/wiki/components/WikiMarkdownParser.php`:

```
<?php
class WikiMarkdownParser extends CMarkdownParser
{
    public function transform($text)
    {
        $text = preg_replace_callback('~\[([.*?])(?:\[|.*?])?\]~', array($this,
            'processWikiLinks'), $text);
        return parent::transform($text);
    }

    protected function processWikiLinks($matches)
    {
        $page = $matches[1];
        $title = isset($matches[2]) ? $matches[2] : $matches[1];
        return CHtml::link(CHtml::encode($title), array('view', 'id' => $page,
            ));
    }
}
```

2. Następnie użyjmy nowej klasy, dodając metodę `getHtml` do modelu `protected/modules/wiki/models/Wiki.php`:

```
public function getHtml()
{
    $parser = new WikiMarkdownParser();
    return $parser->transform($this->text);
}
```

Przechodzimy do dostosowania kontrolera `protected/modules/wiki/controller/DefaultController.php`. Potrzebne nam będą dwie akcje — `view` i `edit`. Ponadto utworzymy akcję `index`, która po prostu wywoła widok poprzez przypisanie `id = index`. Ponieważ nie potrzebujemy widoku dla akcji `index`, można go bezpiecznie usunąć.

3. Teraz utwórz plik `protected/modules/wiki/controller/DefaultController.php`:

```
class DefaultController extends Controller
{
    public function actionIndex()
    {
        $this->actionView('index');
    }

    public function actionView($id)
```

```

{
    $model = Wiki::model()->findByPk($id);
    if(!$model)
    {
        $this->actionEdit($id);
        Yii::app()->end();
    }
    $this->render('view', array(
        'model' => $model,
    ));
}

public function actionEdit($id)
{
    $model = Wiki::model()->findByPk($id);
    if(!$model)
    {
        $model = new Wiki();
        $model->id = $id;
    }
    if(!empty($_POST['Wiki']))
    {
        if(!empty($_POST['Wiki']['text']))
        {
            $model->text = $_POST['Wiki']['text'];
            if($model->save())
                $this->redirect(array('view', 'id' => $id));
        }
        else
        {
            Wiki::model()->deleteByPk($id);
        }
    }
    $this->render('edit', array(
        'model' => $model
    ));
}
}

```

4. Potrzebne nam będą dwa widoki. Utwórz plik *protected/modules/wiki/views/default/view.php*:

```

<h2>
    <?php echo CHtml::encode($model->id)?>
    [<?php echo CHtml::link('edit', array('edit', 'id' => $model->id))?>]
</h2>
<?php echo $model->html ?>

```

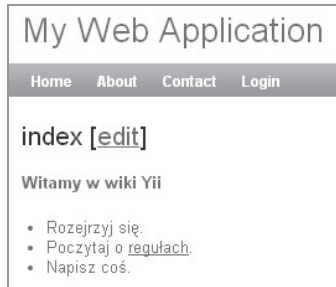
5. Utwórz jeszcze jeden plik *protected/modules/wiki/views/default/edit.php*:

```

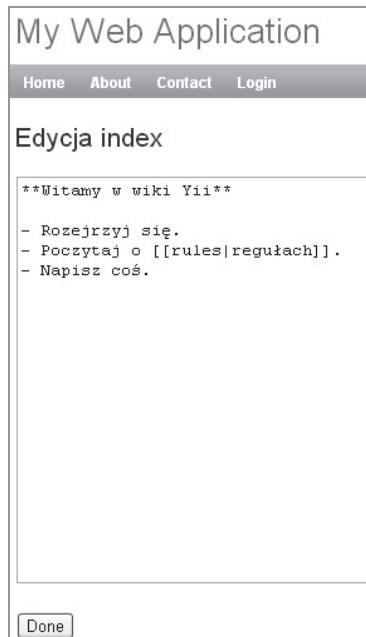
<h2>Edycja <?php echo CHtml::encode($model->id)?></h2>
<?php echo CHtml::beginForm()?>
  <?php echo CHtml::activeTextArea($model, 'text',
    array('cols' => 100, 'rows' => 20))?>
  <br /><br />
  <?php echo CHtml::submitButton('Gotowe')?>
<?php echo CHtml::endForm()?>

```

6. To wszystko. Teraz uruchom moduł wiki i sprawdź go, wpisując jakiś tekst. Nie zapomnij dodać wewnętrznego łącza, na przykład `[[rules]]`:



7. Powyższa strona podczas edycji będzie wyglądać następująco:



8. Ponieważ przenieśliśmy model Wiki z aplikacji do modułu, model nie posiada żadnych zależności związanych z samą aplikacją. Dlatego możemy go przenieść

do pakietu rozszerzenia, takiego jak *protected/extensions/wiki/*, zmieniając konfigurację modułu *protected/config/main.php* na następującą:

```
'modules'=>array(
    // uncomment the following to enable the Gii tool
    'gii'=>array(
        'class'=>'system.gii.GiiModule',
        'password'=>false,
    ),
    'wiki'=>array(
        'class' => 'ext.wiki.WikiModule'
    ),
),
```

Opis działania

Każdy utworzony moduł zawiera główną klasę modułu, taką jak *WikiModule*, w której można zdefiniować konfigurowalne właściwości czy importowane moduły, zmienić ścieżki, dołączyć kontrolery itd. Domyślnie moduł wygenerowany przez Gii uruchamia akcję *index* kontrolera *default*:

```
public function actionIndex()
{
    $this->actionView('index');
}
```

Na stronie *index* naszego modułu *wiki* wywołujemy akcję *view* poprzez przekazanie do niej *id = index*:

```
$model = Wiki::model()->findByPk($id);
if(!$model)
{
    $this->actionEdit($id);
    Yii::app()->end();
}
$this->render('view', array(
    'model' => $model,
));
```

Jeśli istnieje model o takim identyfikatorze, wyświetlamy go w widoku.

Jeśli nie istnieje żadna strona o takim identyfikatorze, ponownie delegujemy procesowanie do innej akcji. Tym razem jest to akcja *edit*:

```
$model = Wiki::model()->findByPk($id);
if(!$model)
{
    $model = new Wiki();
```

```

    $model->id = $id;
}

if(!empty($_POST['Wiki']))
{
    if(!empty($_POST['Wiki']['text']))
    {
        $model->text = $_POST['Wiki']['text'];
        if($model->save())
            $this->redirect(array('view', 'id' => $id));
    }
    else
    {
        Wiki::model()->deleteByPk($id);
    }
}

$this->render('edit', array(
    'model' => $model
));

```

Jeśli nie istnieje model o przekazanym identyfikatorze, tworzymy nowy; jeśli model istnieje, edytujemy go. Dane formularza edycji pochodzą z metody POST i jeśli pole tekstowe formularza jest puste, usuwamy model. Jeśli tekst istnieje, zapisujemy model.

Dodatkowe informacje

W Yii dostępny jest bardzo elastyczny i konfigurowalny moduł wiki o nazwie **Yeeki**. Znajduje się nadal w fazie rozwoju i świetnie nadaje się do nauki tworzenia prawdziwie konfigurowalnych modułów, które można wykorzystać wielokrotnie:

<https://github.com/samdark/Yeeki>

Dodatkowe informacje o modułach można znaleźć na następujących stronach API:

- <http://www.yiiframework.com/doc/guide/1.1/pl/basics.module>
- <http://www.yiiframework.com/doc/api/CWebModule/>
- <http://www.yiiframework.com/doc/api/CModule/>
- <http://www.yiiframework.com/doc/api/GiiModule/>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Niestandardowe renderowanie widoków

Na rynku dostępnych jest wiele silników szablonów PHP. Yii wspiera jedynie dwa rodzaje szablonów: **natywne PHP** i szablony w **stylu Prado**. Jeśli chcemy użyć jednego z istniejących silników szablonów lub utworzyć własny, musimy go zaimplementować, o ile nie zostało to jeszcze zrobione przez społeczność Yii.

W tej recepturze zaimplementujemy wsparcie dla szablonów Smarty.

Przygotowanie

1. Utwórz nową aplikację Yii, korzystając z polecenia `yii create webapp`.
2. Pobierz ostatnią wersję Smarty 3 ze strony <http://www.smarty.net/>.
3. Wyodrębnij zawartość folderu `libs` do folderu `protected/vendors/smarty/`.

Jak to zrobić...

1. Utwórz plik `protected/extensions/smarty/ESmartyViewRenderer.php` i umieść w nim następujący kod:

```
<?php
class ESmartyViewRenderer extends CApplicationComponent implements
    IViewRenderer
{
    public $fileExtension='.tpl';
    public $filePermission=0755;

    private $smarty;

    function init()
    {
        Yii::import('application.vendors.smarty.*');

        spl_autoload_unregister(array('YiiBase','autoload'));
        require_once('Smarty.class.php');
        spl_autoload_register(array('YiiBase','autoload'));
        $this->smarty = new Smarty();
        $this->smarty->template_dir = '';
        $compileDir = Yii::app()->getRuntimePath().'/smarty/compiled/';

        if(!file_exists($compileDir)){
            mkdir($compileDir, $this->filePermission, true);
        }
        $this->smarty->compile_dir = $compileDir;
    }
}
```

```

        $this->smarty->assign('Yii', Yii::app());
    }

    /**
     * Renderuje plik widoku.
     * Ta metoda jest wymagana przez {@link IViewRenderer}.
     * @param CBaseController kontroler lub widżet, który renderuje plik widoku.
     * @param string ścieżka pliku widoku
     * @param mieszany, dane przekazywane do widoku
     * @param boolean, czy należy zwrócić wynik renderowania
     * @return mieszany wynik renderowania lub null, jeśli wynik renderowania jest niepotrzebny.
     */
    public function renderFile($context,$sourceFile,$data,$return) {
        // właściwości bieżącego kontrolera będą dostępne w postaci {this.property}
        $data['this'] = $context;
        if(!is_file($sourceFile) || ($file=realpath($sourceFile))==false)
            throw new CException(Yii::t('ext','Plik widoku "$sourceFile" nie istnieje.', array('{file}'=>$sourceFile)));
        $this->smarty->assign($data);
        if($return)
            return $this->smarty->fetch($sourceFile);
        else
            $this->smarty->display($sourceFile);
    }
}

```

2. Następnie musimy dołączyć do aplikacji nową klasę renderującą widok. W tym celu należy nadpisać komponent `viewRenderer` w pliku `protected/config/main.php`:

```

...
// application components
'components'=>array(
    ...
    'viewRenderer'=>array(
        'class'=>'ext.smarty.ESmartyViewRenderer',
    ),
),
...

```

3. Teraz możemy przetestować nasze rozwiązanie. Utwórz plik `protected/controllers/SmartyController.php` i umieść w nim następujący kod:

```

<?php
class SmartyController extends Controller
{
    function actionNative()
    {
        $this->render('native', array(
            'username' => 'Aleksandrze',
        ));
    }
}

```

```

    }

    function actionSmarty()
    {
        $this->render('smarty', array(
            'username' => 'Aleksandrze',
        ));
    }
}

```

4. Potrzebne są również widoki. Utwórz plik *protected/views/smarty/native.php*:

```

Witaj, <?php echo $username?>!
protected/views/smarty/smarty.tpl:
Witaj, {$username}!

```

5. Spróbujmy teraz uruchomić akcje kontrolera. W obydwu przypadkach powinniśmy uzyskać napis **Witaj, Aleksandrze!**.

Opis działania

Klasa renderująca widok jest potomkiem klasy `CApplicationComponent`, która implementuje interfejs `IViewRenderer`, zawierający tylko jedną metodę o nazwie `renderFile`:

```

/**
 * Renderuje plik widoku.
 * Ta metoda jest wymagana przez {@link IViewRenderer}.
 * @param CBaseController kontroler lub widżet, który renderuje plik widoku.
 * @param string ścieżka pliku widoku
 * @param mieszany dane przekazywane do widoku
 * @param boolean, czy należy zwrócić wynik renderowania
 * @return mieszany wynik renderowania lub null, jeśli wynik renderowania jest niepotrzebny.
 */
public function renderFile($context,$file,$data,$return);

```

W ten sposób uzyskujemy kontekst, ścieżkę szablonu, dane i flagę, która określa, czy musimy wyświetlić wynik przetwarzania natychmiast, czy tylko go zwrócić. W naszym przypadku przetwarzanie odbywa się w silniku szablonów Smarty, dlatego musimy go właściwie zainicjalizować i wywołać jego metody przetwarzania.

Importowanie i inicjalizacja Smarty nie są zbyt łatwe:

```

Yii::import('application.vendors.smarty.*');

spl_autoload_unregister(array('YiiBase','autoload'));
require_once('Smarty.class.php');
spl_autoload_register(array('YiiBase','autoload'));

$this->smarty = new Smarty();

```



```

$this->smarty->template_dir = '';
$compileDir = Yii::app()->getRuntimePath().'/smarty/compiled/';

if(!file_exists($compileDir)){
    mkdir($compileDir, $this->filePermission, true);
}

$this->smarty->compile_dir = $compileDir;
$this->smarty->assign('Yii', Yii::app());

```

Najpierw używamy instrukcji `Yii::import`, aby przekazać Yii, że chcemy użyć klas z folderu *protected/vendors/smarty/*. Ponieważ mechanizm automatycznego ładowania Yii pozostaje w konflikcie z podobnym mechanizmem zawartym w Smarty i ponieważ nie możemy kontrolować tego, co się dzieje w Smarty, musimy wyrejestrować wspomniany mechanizm Yii i dołączyć Smarty. Dopiero wtedy możemy ponownie zarejestrować mechanizm automatycznego ładowania Yii. Ponieważ folder zawierający szablony Yii nie jest stały, zmieniamy domyślną opcję Smarty na pusty łańcuch. Dobrą praktyką jest przechowywanie tymczasowych plików Yii w folderze *application*. To dlatego ustawiamy folder kompilacji, w którym zostaną umieszczone szablony Smarty skompilowane do PHP, na *runtime/smarty/compiled*. Aby nie przeszkadzać zbyt programistom, tworzymy ten folder automatycznie, o ile jeszcze nie istnieje. Ponadto tworzymy specjalną zmienną szablonu Smarty, zwaną *Yii*, do której przypisujemy `Yii::app()`. Zmienna ta pozwala nam uzyskać właściwości aplikacji wewnątrz szablonu.

Samo renderowanie jest nieco prostsze:

```

// właściwości bieżącego kontrolera będą dostępne w postaci {this.property}
$data['this'] = $context;

if(!is_file($sourceFile) || ($file=realpath($sourceFile))===false)
    throw new CException(Yii::t('ext','View file "$sourceFile" does not exist.', array('{file}'=>$sourceFile)));
$this->smarty->assign($data);

if($return)
    return $this->smarty->fetch($sourceFile);
else
    $this->smarty->display($sourceFile);

```

Deklarujemy kolejną zmienną Smarty, zwaną *this*, która pozwoli uzyskać właściwości kontrolera, takie jak tytuł strony.

Aby ułatwić debugowanie, sprawdzamy, czy szablon istnieje, a jeśli nie, wyświetlany jest komunikat błędu z nazwą szablonu.

Cały zestaw danych jest przekazywany bez zmian do szablonu Smarty poprzez wywołanie `$this->render`. Następnie renderujemy szablon lub go zwracamy — w zależności od parametru `$return`.

Dodatkowe informacje

Dostępny na stronie <http://www.yiiframework.com/extension/smarty-view-renderer> plugin i wskazówki dotyczące konfiguracji ułatwią korzystanie z renderowania widoków Smarty.

W aplikacji możemy używać własnych silników szablonów. Jeśli jednak tworzymy rozszerzenie wielokrotnego użytku, spróbujmy unikać silników szablonów innych niż natywne PHP.

Warto przeczytać

Dodatkowe informacje o Smarty i o renderowaniu widoków można znaleźć na następujących stronach:

- <http://www.smarty.net/>
- <http://www.yiiframework.com/doc/api/IViewRenderer/>
- <http://www.yiiframework.com/doc/api/CViewRenderer/>
- <http://www.yiiframework.com/doc/api/CPradoViewRenderer/>

Zobacz też

- Receptura „Przygotowywanie rozszerzeń do dystrybucji”.

Przygotowywanie rozszerzeń do dystrybucji

W tym rozdziale nauczyłeś się, jak należy tworzyć różne rodzaje rozszerzeń Yii. Teraz dowiesz się, jak udostępnić swoje wyniki innym osobom i dlaczego to takie ważne.

Przygotowanie

Przygotujmy najpierw listę kontrolną dla dobrego rozszerzenia. Dobry produkt programistyczny powinien spełniać następujące warunki:

- Spójne i łatwe do odczytu i użycia API.
- Dobra dokumentacja.
- Łatwy do znalezienia.
- Rozszerzenie powinno być dostosowane do większości przypadków użycia.
- Rozszerzenie powinno być utrzymywane.

- Dobrze przetestowany kod; w idealnej sytuacji powinien zawierać testy jednostkowe.
- Zapewnione wsparcie dla rozszerzenia.

Oczywiście spełnienie wszystkich tych wymagań wymaga wiele pracy, jednak jest potrzebne do uzyskania dobrego produktu.

Jak to zrobić...

1. Przyjrzyj się dokładnie naszej liście, počawszy od API. API powinno być spójne oraz łatwe do odczytu i użycia. Spójność oznacza, że ogólny styl nie powinien się zmieniać. Dlatego nie powinniśmy wprowadzać różnych konwencji nazewnictwa, na przykład `camelCasedVariableNames` i `underscored_variable_names`; niespójnych nazw, na przykład `isFlag1()` i `isNotFlag2()`; itd. Wszędzie należy przestrzegać określonych dla kodu reguł. Pozwala to ograniczyć konieczność sprawdzania dokumentacji i skupić się na kodzie.
2. Kod bez żadnej dokumentacji jest niemal nieprzydatny. Wyjątkiem jest dość prosty kod, jednak nawet jeśli kod liczy jedynie kilka wierszy, pozostawienie go bez żadnej wzmianki o instalacji i użyciu nie jest właściwe. Co składa się na dobrą dokumentację? Cele kodu i jego zalety powinny być możliwie jasne i powinny być napisane wyraźnie i przejrzysto. Jeśli na przykład rozszerzenie wysyła wiadomość e-mail, opis powinien być podobny do poniższego:

EMailer pozwala na przesyłanie wiadomości e-mail w prosty i wygodny sposób. Prawidłowo obsługuje tekst i tytuł w języku angielskim. Ponadto pozwala na użycie widoków Yii jako szablonów wiadomości e-mail.

Kod jest bezużyteczny, jeśli programiści nie wiedzą, gdzie go umieścić i co powinno się znaleźć w konfiguracji aplikacji. Nie oczekuj, że programiści będą wiedzieć, jak wykonywać zadania specyficzne dla platformy. Wskazówki instalacji powinny być wyczerpujące. Większość programistów preferuje wskazówki krok po kroku. Jeśli kod do swojego działania wymaga schematu SQL, należy go dołączyć.

Nawet jeśli metody i właściwości naszego API mają odpowiednie nazwy, nadal powinniśmy je udokumentować w komentarzach phpDoc, określając typy argumentów i zwracanych elementów oraz zapewniając krótki opis dla każdej metody. Ponadto należy rozważyć wypisanie w dokumentacji listy metod i właściwości publicznych, aby uzyskać zestaw odsyłaczy.

Obok dobrze skomentowanego kodu należy udostępnić przykładowe przypadki użycia. Spróbuj uwzględnić większość popularnych sposobów użycia rozszerzeń. W przykładzie postaraj się nie rozwiązywać wielu problemów jednocześnie, ponieważ może to być mylące.

3. Rozszerzenie powinno posiadać numer wersji i rejestr zmian. Pozwoli to społeczności na sprawdzenie, czy posiadana wersja jest aktualna, oraz na zapoznanie się z ostatnimi zmianami przed przystąpieniem do aktualizacji.

Ważne jest, aby kod był elastyczny i aby można go było wykorzystać w wielu przypadkach. Ponieważ jednak nie ma możliwości utworzenia kodu dla każdego możliwego przypadku użycia, spróbuj uwzględnić większość tych popularnych.

4. Ważne jest, aby programiści czuli się wygodnie z rozszerzeniem. Dostarczenie dobrej dokumentacji jest pierwszym krokiem. Drugi to udostępnienie dowodu, że kod działa zgodnie z zamierzeniem i że będzie działać wraz z przyszłymi aktualizacjami. Najlepszym sposobem, aby to osiągnąć, jest dołączenie zestawu testów jednostkowych.
5. Rozszerzenie należy utrzymywać przynajmniej do osiągnięcia stabilności i upewnienia się, że nie ma już żadnych propozycji funkcji i raportów o błędach. Dlatego należy się spodziewać pytań i raportów. Trzeba też zarezerwować w przyszłości nieco czasu na pracę nad kodem. Jeśli nie możemy poświęcić więcej czasu na utrzymanie rozszerzenia, które jest jednak bardzo innowacyjne, i nikt wcześniej nie zaproponował niczego podobnego, nadal warto je udostępnić. Jeśli społeczność doceni rozszerzenie, ktoś z pewnością zaoferuje swoją pomoc.
6. Musimy wreszcie udostępnić rozszerzenie. Popularnym miejscem publikacji rozszerzenia Yii jest strona <http://www.yiiframework.com/extensions/>. Zanim będzie można przesłać własny kod, należy się zarejestrować na forum i przesłać kilka postów. Spróbuj znaleźć dobrą opisową nazwę, napisać opisowe podsumowanie i określić odpowiednie kategorie i znaczniki. Nie używaj archiwów innych niż ZIP lub GZIP, ponieważ są to najpopularniejsze formaty archiwów i możemy być pewni, że każdy będzie mógł je rozpakować.
7. Nawet jeśli rozszerzenie jest dość proste, a dokumentacja dobra, możemy się spodziewać pytań i, przynajmniej na początku, jedynym człowiekiem, który może na nie odpowiedzieć, będziesz Ty. Zwykle pytania są zadawane na oficjalnych forach, dlatego lepiej jest utworzyć nowy wątek, na którym programiści mogą prowadzić dyskusję o kodzie, oraz podać łącze do strony rozszerzenia.

Opis działania

Jeśli chcesz udostępnić rozszerzenie społeczności i upewnić się, że będzie użyteczne i popularne, musisz dostarczyć nie tylko sam kod. Przygotowanie rozszerzeń do dystrybucji wymaga nieco więcej pracy. Może to być nawet więcej niż samo tworzenie rozszerzenia. Dlaczego więc warto udostępnić rozszerzenie społeczności? W porównaniu z kodem używanym we własnych projektach otwarte projekty mają swoje zalety. Możemy wykorzystać do przetestowania gotowych źródeł projektu o wiele więcej osób, niż moglibyśmy znaleźć osobiście. Osoby korzystające z rozszerzenia testują je, dostarczając wartościowych opinii i raportując błędy. Jeśli kod jest popularny, znajdują się programiści pasjonaci, którzy będą chcieli usprawniać kod, aby uczynić go bardziej kompleksowym, stabilnym i zdatnym do wielokrotnego użytku. Ponadto sprawi nam to przyjemność, ponieważ robimy dobry uczynek.

Dodatkowe informacje

Omówiono już większość ważnych zagadnień. Jednak nadal mamy wiele do sprawdzenia. Zanim napiszesz własne rozszerzenia, wypróbuj istniejące. Jeśli rozszerzenie spełnia prawie wszystkie wymagania, spróbuj skontaktować się z autorem rozszerzenia i podzielić się swoimi pomysłami. Przeglądanie istniejącego kodu pomaga sprawdzić przydatne sztuczki i zasady. Przeglądaj też od czasu do czasu artykuły wiki i oficjalne fora; dostępnych jest wiele przydatnych informacji o tworzeniu rozszerzeń i ogólnie programowaniu z wykorzystaniem Yii.

Skorowidz

A

Active Record, *Patrz:* rekord aktywny
adres
 URL, 42, 44, 275
 generowanie według ścieżek, 45
 krótki, 53, 138
 pełny, 47
 względny, 47
AJAX, 93
AJAX helper, *Patrz:* helper AJAX CHtml
akcja
 getQuery, 83
 getQuote, 81, 83
 identyfikator, 44
 kontrolera, *Patrz:* kontroler akcja
 zewnętrznego, 231
 wielokrotnego użytku, 229, 231, 232
 zewnętrzna, 64
alias, 25, 65
 application, 25
 definiowanie, 25
 ext, 25
 system, 25
 webroot, 25
 zii, 25
APC, 310
API, 27
API JSON CRUD, 232
aplikacja
 dostęp, 44
 internacjonalizacja, *Patrz:* internacjonalizacja

 konsolowa, 31, 364
 przenoszenie poza folder główny, 360, 362
 sieci WWW, 31
 wdrażanie, 357
AR, *Patrz:* rekord aktywny
atak
 CSRF, 277, 291, 292, 294
 SQL injection, 277, 286, 289, 290, 291
 XSS, 277, 282, 284, 286
 nietrwały, 285
 trwały, 285
automatyzacja znaczników czasu, 183
autor, 185, 187

B

baza danych, 161, 162, 166, 291, 306, 326
 noSQL, 309
 połączenie, 167
 Redis, 309
 relacyjna, 187
 Sakila, 162, 193
 SQLite, 150
 zabezpieczenia, 286
 zależność, 316
BBCode, 175
bezpieczeństwo, 31, 37, 202, 280, 282, 293, 294, 372
biblioteka
 AngularJS, 117
 Backbone.js, 117
 cURL, 121
 doT, 105

biblioteka

- Ember.js, 117
 - GD, 226
 - KnockoutJS, 117
 - podświetlania kodu, 183
 - Purifier, 284, 285
 - SPL, 23, 34, 340, 341
 - udostępnianie, 357
 - Zii, 193
- blog, 20, 28, 74, 183, 317, 320
- komentarz, 20
 - w wielu językach, 170
- błąd, 143, *Patrz też:* kod HTTP
- 404, 272
 - krytyczny, 144, 265
 - niekrytyczny, 29
 - obsługa, 259, 268, 275
 - własna, 272
 - PIIP, 31
 - stos wywołań, *Patrz:* stos wywołań
- bot spamowy, 131
- bufor APC, 305
- buforowanie, *Patrz też:* sesja buforowanie
- IITTP, 327, 330
 - po stronie serwera, 327

C

- CAPTCHA, 126, 128, 129, 133
- dostosowywanie, 131
- CConsoleApplication, *Patrz:* aplikacja konsolowa
- chmura tagów, 32
- ciasteczka, 32, 37, 40, 151, 293, 370, 375
- CLDR, 170
- CList, 18
- CLogRouter, 264
- Comment, 20
- Common Locale Data Repository, *Patrz:* CLDR
- Composer, 351, 353, 355
- cross-site request forgery, *Patrz:* CSRF
- cross-site scripting, *Patrz:* XSS
- CRUD, 327
- CSRF, 291
- CSS, 33, 85, 91, 92, 306
- selektor, 155
- CWebApplication, *Patrz:* aplikacja sieci WWW
- czas, 184

D

dane

- dostawca, *Patrz:* dostawca danych
 - model, *Patrz:* model danych
 - wejściowe, 152
 - filtrowanie, 277, 290
 - wyjściowe, 152
 - buforowanie, 243
 - kodowanie, 277, 282
 - kompresja, 18
- DAO, 161, 162, 165, 167, 168
- data, 184
- debugowanie, 259, 304, 309, 370
- dekorator, 73, 74
- dokumentacja, 255
- dostawca danych, 76, 193, 194, 198, 199, 200, 207, 208
- drzewo klas, 190
- dziedziczenie, 16, 187
- odzworowanie do pojedynczej tabeli, 187, 191

E

- eAccelerator, 309
- e-mail, 19, 334, 336
- szablon, 69, 70

F

- fabryka widżetów, *Patrz:* widżet fabryka
- FIFO, 36
- filtr, 242, 243, 316
- aplikacji, 330
 - CAccessControlFilter, 277
 - CInlineFilter, 277
 - dostępu, 281
 - kontrolera, 277
 - łańcuch, 243
 - optymalizacja, 330
- filtrowanie, 205, 206
- Firebug, 260, 262
- Firefox Firebug, *Patrz:* Firebug
- folder, 357, 359, 360
- definiowanie, 359
 - framework, 362
 - główny, 360
 - webroot, 362
- format JSON, *Patrz:* JSON

formularz, 32, 66, 119, 292
 ochrona przeciwpamowa, 126
 pole pobierania danych, 98, 99
 przysyłanie pliku na serwer, 122, 125, 126
 token, 292
 UserForm, *Patrz:* UserForm
 walidator, *Patrz:* walidator
 widżet, *Patrz:* widżet formularza
 Fowler Martin, 187

G

GD, 129, 226, 228
 generator zapytań, 161, 162, 165, 167, 168, 306
 getter, 13, 14, 225
 Gi, 200, 202, 248
 GitHub, 156
 Gmail, 219
 Google
 API, 236, 334
 walidacja własności witryny, 119
 GZIP, 18, 242

H

haszowanie, 31
 helper
 AJAX CIhtml, 82
 CActiveForm, 125
 CIhtml, 125
 hierarchia RBAC, *Patrz:* RBAC
 hosting współdzielony, 18
 HTML, 175
 HTML Purifier, 284

I

informacja kontekstowa, 268, 271, 272
 instrukcja
 include, 23, 367
 require, 23, 367
 interfejs
 ArrayAccess, 34
 Countable, 34
 graficzny, 238
 IFilter, 243
 IteratorAggregate, 34
 IViewRenderer, 252
 SPL, 34
 Traversable, 34
 internacjonalizacja, 170

J

JavaScript, 85, 92
 skrypt, *Patrz:* skrypt JavaScript
 język
 BBCode, *Patrz:* BBCode
 IITML, *Patrz:* IITML
 markdown, *Patrz:* markdown
 Textile, *Patrz:* Textile
 znaczników, 245
 jQuery, 79
 JSON, 93, 105

K

klasa
 automatycznego ładowania, 341
 bazowa, 22
 CAction, 232
 CActiveRecordBehavior, 224
 CApplicationComponent, 225, 228, 252
 CArrayDataProvider, 198, 199
 CAttributeCollection, 34, 36
 CBehavior, 225
 CButtonColumn, 202
 CClientScript, 90
 CComponent, 13, 16, 225
 CConsoleApplication, 240
 CConsoleCommand, 240
 CController, 68
 CCookieCollection, 39
 CDbConnection, 30
 CDbCriteria, 191, 192
 CDbTestCase, 223
 CEvent, 23
 CFileLogRoute, 264
 CFilter, 243
 CIhtml, 282
 CIhtmlPurifier, 282
 CIHttpRequest, 37
 CJavaScriptExpression, 82
 CList, 34
 CLogFilter, 271
 CMap, 34, 35, 36
 CMenu, 24
 CQueue, 34, 36
 criteria, 191, 192
 CSqlDataProvider, 199
 CStack, 34, 36
 CTestCase, 146

- CTypedList, 34
 - CUploadedFile, 125
 - CValidator, 122
 - CViewAction, 64
 - CWebApplication, 240
 - CWebUser, 67
 - Kohana, 345
 - kolekcji, 34
 - metoda statyczna, 17
 - NotFoundHandler, 273
 - PIIPUnit, 146
 - PIIPUnit_Framework_TestCase, 146
 - podświetlająca kod, 178
 - potomna, 17
 - rozszerzeń, 24
 - tasks-add, 104
 - Text_Highlighter, 178
 - Text_Password, 349
 - właściwość, 151
 - Zii, 24
 - klip, 71, 72
 - klucz, 35, 199, 204
 - jquery, 91
 - tablicy, 91
 - yii, 91
 - yiiactiveform, 91
 - yiiitab, 91
 - kod, *Patrz też:* język
 - HTML kompresja, 242
 - HTTP
 - 200, 113
 - 400, 113
 - 404, 113, 272
 - 500, 113
 - kuponu, 146
 - podświetlanie, 178, 182, 183
 - raportów pokrycia kodu, 138
 - wielokrotnego użytku, 228
 - kodowanie PDO, 289
 - kolejka, 34, 36
 - komunikatów, 373
 - kolekcja, 34
 - komponent, 15, 193
 - aplikacji nadpisywany, 228
 - assetManager, 32
 - authManager, 32, 298, 299
 - CAPTCHIA, *Patrz:* CAPTCHIA
 - clientScript, 32
 - coreMessages, 31
 - errorHandler, 31
 - format, 31
 - konfiguracja, 29, 30, 31
 - messages, 31
 - przekierowywania logów, 263
 - request, 32
 - securityManager, 31
 - session, 32
 - standardowy, 31
 - statePersister, 31
 - testowanie, 137
 - themeManager, 32
 - urlManager, 32, 365
 - widgetFactory, 32
 - Yii::app, 30
 - komunikat, 66
 - kolejka, 373
 - logowanie, *Patrz:* logowanie komunikatów
 - o błędzie, 143
 - o trybie konserwacji, 373
 - tłumaczenie, 31
 - konto bankowe, 293
 - kontrola dostępu, 277, 281, 294, 301
 - oparta na rolach, *Patrz:* RBAC
 - kontroler, 47, 92
 - akcja, 60
 - api, 235
 - bazowy, 59
 - customer, 201
 - identyfikator, 44, 47
 - kontekst w widoku, 67, 68
 - SecureController, 60
 - snippet, 181
 - TestController, 60
 - WebsiteController, 44
 - wielokrotnego użytku, 232
 - zewnętrzny, 231
 - kopia zapasowa, 373
 - kupon, 146
- ## L
- LIFO, 36
 - lista, 34, 76, 193
 - biała, 290
 - procedur, 18
 - Zii, 207
 - logu rejestr, *Patrz:* rejestr logów
 - logowanie komunikatów, 260, 262, 265, 268, 271, 272
 - natychmiastowe, 265

Ł

ładowanie automatyczne, 23, 25, 338, 341, 342, 344
 łańcuch
 MSIE, 281
 tekstowy, 19, 49, 298
 zależności, 311

M

mapa, 34
 YiiBase::\$_coreClasses, 24
 markdown, 175, 177, 245, 306
 mechanizm
 APC, *Patrz:* APC
 memcached, *Patrz:* memcached
 platformy Zend, 309
 memcached, 310
 menedżer zależności, 351
 metoda
 __get, 16
 __isset, 16
 __set, 16
 __unset, 16
 accessRules, 130, 278
 actionAr, 165
 actionPage, 44, 47
 actionQueryBuilder, 165
 actionSQL, 165
 actionTask, 112
 activeTextField, 136
 addRules, 57
 afterConstruct, 175
 afterDelete, 175, 225
 afterFind, 175
 afterSave, 175
 afterValidate, 22, 175, 182
 ajaxLink, 46
 assertRegExp, 146
 assertTrueassertFileExists, 146
 assign, 298
 attachEventHandler, 17
 autoload, 23, 340
 beforeDelete, 175
 beforeFind, 175
 beforeSave, 174, 175
 beforeValidate, 175, 177, 184, 185, 187
 beginContent, 73
 beginForm, 293
 beginProfile, 265
 CCaptchaAction, 132
 CClientScript, 88
 checkAccess, 299, 300
 checkRequirements, 129
 CIhtml, 46
 CIHttpCacheFilter, 330
 CMarkdownParser, 177
 collectRules, 57
 createAbsoluteUrl, 47
 createUrl, 47, 48
 createWebApplication, 337
 CTimestampBehavior, 184, 185
 CUrlManager, 48
 CViewAction, 64, 65
 encode, 98, 284
 encodelooks, 284
 endContent, 73
 endProfile, 265
 filters, 278
 find, 191
 findAll, 191
 form, 46
 generateVerifyCode, 132
 GET, 292, 294
 getController, 232
 getCookies, 39
 getCsrftoken, 293
 getEventHandlers, 18
 getHelp, 241
 getHostInfo, 37
 getId, 232
 getInstance, 125
 getIsAjaxRequest, 37
 getIsPostRequest, 37
 getOwner, 224
 getPathInfo, 37
 getPreferredLanguage, 38
 getQueryString, 37
 getRequestType, 37
 getRequestUri, 37
 getSupportedLanguages, 183
 getUrl, 37
 handle, 275
 handle*, 113
 hasEvent, 18
 hasEventHandler, 18
 IHTTP, 112
 import, 24, 337
 init, 216
 instantiate, 190

- klasy, 120
 - bazowej, 22
 - macierzystej, 22
 - link, 46
 - loadModel, 113
 - log, 263, 265
 - macierzysta, 243
 - mergeArray, 35
 - niestandardowa, 14
 - normalizeUrl, 46, 216
 - obsługi sesji, 32
 - POST, 293, 294
 - postFilter, 243
 - preFilter, 243
 - process, 141
 - processRequest, 57
 - przechowywania globalnych stanów, 31
 - przypominająca zdarzenie, 173, 175
 - publish, 88, 89
 - raiseEvent, 17
 - refresh, 46
 - registerAutoloader, 341, 348
 - registerClientScript, 91
 - registerCoreComponents, 31
 - registerScriptFile, 91
 - rekordu aktywnego, 191, 192
 - render, 71, 73, 92
 - renderDataCellContent, 216
 - renderPartial, 71
 - renderPartials, 68
 - run, 237
 - saveTask, 113
 - search, 203, 205
 - sendResponse, 113
 - setFlash, 67
 - setPathOfAlias, 25, 359
 - setUp, 150
 - setUpBeforeClass, 150
 - spl_autoload_register, 340
 - statyczna, 35
 - klasy, *Patrz:* klasa metoda statyczna
 - t, 19
 - tableName, 190
 - teardown, 151
 - tearDownAfterClass, 151
 - testCodeAcceptance, 151
 - testCodeNotFound, 151
 - testowa, 223
 - textField, 136
 - trace, 263, 265
 - validateAttribute, 122
 - value, 216
 - widget, 68, 136, 237
 - zdarzenia, 23
 - zorientowana obiektowo, 37
 - model
 - Car, 190
 - Customer, 201, 203
 - danych, 194
 - edytowanie, 66
 - FamilyCar, 191
 - instancja nieprzetworzonych danych, 191
 - Post, 300
 - rekordu aktywnego, *Patrz:* rekord aktywny
 - snippet, 182
 - SportCar, 191
 - moduł
 - tworzenie, 244
 - wiki, 244, 247
 - Yeeki, *Patrz:* Yeeki
 - MySQL, 162, 169, 326
- ## N
- nagłówek
 - IHTTP, 327
 - If-Modified-Since, 330
 - If-None-Match, 330
 - narzędzie
 - Apache, 308
 - cron, 307
 - numer IP, 280
- ## O
- obiekt, 15
 - CException, 29
 - obraz, 306
 - skalowanie, 226, 306, 342, 348
 - odzworowanie dziedziczenia do pojedynczej tabeli, 187, 191
 - operacja, 297, 298, 299
 - optymalizacja, 317, 320, 326
- ## P
- pamięć podręczna, 58, 311, 326
 - parser
 - kodu markdown, 177
 - URL, 32, 49, *Patrz:* URL parser

- PEAR, 138, 178, 333, 349, 351
 - instalowanie, 138
 - PHP
 - przypisanie natywne, 14
 - szablon, *Patrz:* szablon PIIP
 - PHPUnit, 138, 146, 155, 156
 - globalne ustawienia, 140
 - instalowanie, 138
 - podręcznik, 158
 - platforma
 - Kohana, 333, 342, 345, 349
 - PEAR, *Patrz:* PEAR
 - Zend, *Patrz:* Zend
 - plik
 - bootstrap.php, 140
 - index.php, 336, 359, 362, 363
 - index-test.php, 359, 362, 363
 - konfiguracyjny, 19, 29, 96, 169, 364, 367, 370, 375
 - main.php, 29
 - natywne, 367
 - phpunit.xml, 140
 - protected/config/main.php, 364
 - przesyłanie na serwer, 122, 125, 126
 - testu, 141
 - WebTestCase.php, 140
 - yiiLite.php, 304, 305
 - pole wymagane, 182
 - polecenie
 - clean, 241
 - konsolowe, 238, 241, 371, 372, 373
 - message, 241
 - migrate, 241
 - yii webapp, 60
 - portfolio, 74
 - procedura
 - obsługi błędu, 273
 - 404, 272
 - obsługi zdarzenia, 17, 18, 22, 274
 - protokół RFC, 331
 - przeładowanie, 330
 - emulowanie, 371, 372
 - przepustowość, 242
 - przycisk, 152
 - Purifier, 285
- R**
- raport pokrycia kodu, 138, 156, 157, 158
 - RBAC, 32, 294, 297, 299, 300
 - RDBMS, 169
 - Redis, 309
 - reguła
 - bezpieczeństwa, 182, 202
 - biznesowa, 298
 - bizRule, 298, 299
 - domyślna, 54
 - dostępu, 280
 - kolejność, 280
 - parametr domyślny, 54
 - sparametryzowana, 44
 - reguła URL, *Patrz:* URL reguła
 - rejestr logów, 260
 - rekord aktywny, 20, 30, 76, 103, 161, 162, 165, 167, 168, 172, 173, 190, 191, 192, 224, 289
 - implementacja, 187, 191
 - wydajność, 305, 306
 - REST, 112
 - rola, 297, 298, 299
 - hierarchia, 295
 - role-based access control, *Patrz:* RBAC
 - rozszerzenie, 254
 - dystrybucja, 256
 - numer wersji, 255
 - rejestr zmian, 255
 - udostępnianie, 256
 - rozszerzenie Yii, 219
- S**
- selektor CSS, *Patrz:* CSS selektor
 - serwer
 - buforowanie, 305
 - produkcyjny, 305
 - Selenium, 138
 - instalowanie, 139
 - sieciowy, 32, 37
 - Zend Server, 309
 - zmienna, 151
 - sesja
 - buforowanie, 309, 310, 311, 316, 317, 320
 - mechanizm obsługi, 309
 - natywne, 307, 309
 - zamykanie, 310
 - setter, 13, 14, 225
 - siatka, 76, 193, 198, 206
 - niestandardowa, 212
 - Zii, 200
 - silnik szablonu, 252, 254
 - PHP, 250

skrypt, 92
 jQuery, 104
 klienta, 32
 wewnętrzny, 90
 wpleciony w kod, 90, 91
 wstrzykiwanie po stronie klienta, 282
 zewnętrzny, 90, 91
 skrypt JavaScript, 82
 sortowanie, 76, 78, 194, 205, 209
 spam, 126
 stała YII_DEBUG, 29
 stos, 34, 36
 wywołań, 259, 266, 268
 strona
 statyczna, 52, 64
 wczytywana asynchronicznie, 79
 stronicowanie, 76, 78, 194
 struktura danych
 JavaScript, 98
 PIIP, 98
 system plików, 309
 szablon, 32, 69, 252
 doT, 117
 natywne PIIP, 250
 PIIP, 250
 silnik, *Patrz:* silnik szablonu
 Smarty, 250, 252, 254
 w stylu Prado, 250
 szyfrowanie, 31

Ś

ścieżka
 bazowa, 65
 wewnętrzna, 42

T

tabela car, 190
 tablica, 191, 198
 asocjacyjna, 35
 TDD, 137, 145
 technika Test-driven Development, *Patrz:* TDD
 test, 137
 automatyczny, 137
 czarnej skrzynki, 152
 dla modelu Post, 220
 funkcjonalny, 138, 152, 153, 156
 jednostkowy, 138, 141
 konfiguracja, 146, 150

konfiguracja, 223
 narzędzia, 140
 platformy Yii, 156
 Test-driven Development, *Patrz:* TDD
 Textile, 175
 token, 292, 293
 trasa, 305
 komponentu urlManager, 365
 URL, *Patrz:* URL trasa
 wewnętrzna, 47
 tryb konserwacji, 373, 374, 375

U

układ, 65, 73
 region treści, 71, 72
 zróżnicowanie, 74
 URL, 92
 adres, *Patrz:* adres URL
 łańcuch, 46, 173
 parser, 44
 reguła, 42
 dla stron statycznych, 52
 przekazywanie, 55
 trasa, 42
 UserForm, 22
 usuwanie nietrwałe, 220
 użytkownik
 autoryzacja, 298
 autoryzowany, 279, 281
 hasło, 287
 identyfikator, 15
 język preferowany, 38
 nazwa, 287
 rola, *Patrz:* rola
 sesja, 32

W

walidacja, 166, 175, 177, 182, 185, 187, 202, 306
 walidator, 119
 CCaptchaValidator, 130
 plików CFileValidator, 124
 pliku, 126
 reguła, 121
 wartość skalarna, 35
 wiadomość
 e-mail, *Patrz:* e-mail
 typu flash, 66, 67

wideo, 69
 widok, 67, 68, 81
 częściowy, 69, 81
 renderowanie, 254
 widżet, 32, 85, 86, 92, 136, 194, 236, 238
 CAPTCHA, 131
 CCaptcha, 130
 CListView, 208
 do pobierania danych, 133, 135
 fabryka, 33
 formularza, 133, 135
 JavaScript, 82
 siatki, 198, 214
 skórka, 32
 zaznacz wszystko, 152
 wiki, 244, 247
 WinCache, 309
 właściwość
 class, 30
 dataProvider, 203
 definiowanie, 13
 expression, 281
 filter, 203
 handled, 23
 klasy, 151
 lastModified, 331
 lastModifiedExpression, 331
 pageTitle, 68
 publiczna, 64, 240
 schemaCachingDuration, 305
 sender, 23
 sortableAttributes, 209
 tylko do odczytu, 14
 urlRules, 57
 wydajność, 303, 309, 316, 317
 po stronie serwera, 306
 wyjątek, 26, 31
 CHttpException, 28
 LyricsFinderHttpException, 27
 niestandardowy, 27
 wyrażenie regularne, 49, 51, 53, 174
 kwantyfikatory leniwy, 144
 wywołanie zwrotne, 17, 82

X

XCache, 309
 Xdebug, 138, 139, 156
 XSRF, *Patrz:* CSRF
 XSS, 282, 283

Y

Yeeki, 249
 YouTube, 69

Z

zachowanie, 225
 zadanie, 297, 298, 299
 cron, 371, 372
 w tle, 371
 zależność, 317, 351
 bazy danych, 316
 stanu globalnego, 316
 zapytanie, 194, 289, 306
 SQL, 161, 162, 165, 167, 168, 306, 316, 326
 zasoby, 85
 zdarzenie, 16, 18, 225
 deklarowanie, 17
 onBeginRequest, 18
 onClick, 104
 onEndRequest, 18
 onError, 29
 onException, 29, 273
 onMissingTranslation, 19
 Zend, 309, 333, 334, 337
 Zend_Loader_Autoloader, 336
 Zend_Mail, 334, 335
 zmienna
 \$_GET, 37
 \$_POST, 37
 \$_SERVER, 37
 superglobalna PIIP, 37
 znacznik, 92
 BBCode, 141
 meta, 92

Ż

żądanie
 AJAX, 104, 216
 delete, 112
 get, 112, 114
 przechwytywanie, 375
 put, 112
 typ, 37

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

Tworzenie aplikacji z Yii Receptury

PHP to jeden z języków programowania najczęściej wybieranych do tworzenia stron internetowych i aplikacji webowych. Yii to platforma MVC napisana w języku PHP, która sprawi, że Twój kod będzie bardziej przejrzysty, a osiągnięcie założonych celów — znacznie prostsze. Jeżeli połączysz potencjał PHP ze wsparciem Yii, otrzymasz kombajn, który pozwoli Ci w okamgnieniu poradzić sobie z dowolnym problemem.

Książka ta należy do cenionej przez programistów serii „Receptury”. Znajdziesz w niej najczęściej spotykane problemy wraz z ich najlepszymi rozwiązaniami. W trakcie lektury dowiesz się, jak skonfigurować reguły URL, przekazywać wiadomości pomiędzy żądaniami oraz stosować widoki częściowe. Ponadto poznasz najlepsze przepisy na wykorzystanie technologii AJAX oraz biblioteki jQuery wspólnie ze szkieletem Yii. Jeżeli stoisz przed problemem zabezpieczenia aplikacji przed spamem i chciałbyś wykorzystać mechanizm CAPTCHA, to znajdziesz tu szczegółową instrukcję, jak to zrobić. Co jeszcze zawiera ta książka? Najlepsze porady dotyczące wydajności, współpracy z bazą i bezpieczeństwa. Jest to lektura obowiązkowa dla każdego programisty chcącego wykorzystać możliwości Yii w swojej aplikacji!

Poznaj możliwości szkieletu Yii!

helion.pl
księgarnia
internetowa

Nr katalogowy: 17806

Księgarnia internetowa
<http://helion.pl>

Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nawosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Informatyka w najlepszym wydaniu



Dzięki tej książce:

- zabezpieczysz formularz przy użyciu CAPTCHA
- wyciśniesz siódme poty ze szkieletu Yii
- zwiększysz bezpieczeństwo Twojej aplikacji
- zbudujesz działającą aplikację szybko i bez problemów

sięgnij po **WIĘCEJ**



KOD KORZYSCI

ISBN 978-83-246-8596-7



9 788324 685967

Cena: 67,00 zł