Technologia i rozwiązania

CakePHP 1.3
Receptury

Poznaj optymalne przepisy na CakePHP!

- Jak uwierzytelnić użytkownika?
- Jak stworzyć usługę sieciową?
- Jak skutecznie przetestować aplikację?

Helion

Mariano Iglesias

PACKT PUBLISHING

Tytuł oryginału: CakePHP 1.3 Application Development Cookbook

Tłumaczenie: Przemysław Pietraszek (rozdz.1),
Krzysztof Rychlicki-Kicior (wstęp, rozdz. 2 – 11)

ISBN: 978-83-246-3542-9

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/ caph3r.zip

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/caph3r
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 9

Internacjonalizacja aplikacji

W tym rozdziale omówimy następujące zagadnienia:

- internacjonalizację tekstów w kontrolerach i widokach;
- internacjonalizację komunikatów walidacji w modelach;
- tłumaczenie tekstów zawierających dynamicznie generowaną treść;
- ekstrakcję i tłumaczenie tekstów;
- tłumaczenie rekordów baz danych za pomocą zachowania `Translate`;
- ustawianie i zapamiętywanie języka.

Wprowadzenie

W tym rozdziale zajmiemy się przykładami, które pozwolą na internacjonalizację — umiędzynarodowienie — wszystkich elementów aplikacji CakePHP, zarówno statycznych (zawartych np. w widokach), jak i dynamicznych (np. rekordy baz danych).

W pierwszych dwóch przykładach pokażemy, jak udostępnić elementy widoków, a także komunikaty walidacji modelu do tłumaczeń. W trzecim przykładzie będziemy tłumaczyć bardziej złożone wyrażenia. Czwarty przykład to pokaz możliwości wbudowanych narzędzi CakePHP, które potrafią wyłuskać statyczną treść aplikacji wymagającą tłumaczenia. Piąty przykład przedstawia mechanizm tłumaczenia rekordów baz danych. Na zakończenie dowiesz się, jak umożliwić użytkownikowi zmianę aktywnego języka aplikacji.

Internacjonalizacja tekstów w kontrolerach i widokach

W tym przykładzie dowiesz się, jak zinternacjonalizować tekst, który znajduje się w widokach naszej aplikacji, a także jak przygotować takie teksty do tłumaczenia.

Zanim zaczniesz

Aby wykonać poniższy przykład, musisz skorzystać z przykładowych danych. Utwórz tabelę `articles`, korzystając z poniższego zapytania SQL:

```
CREATE TABLE 'articles'(
   'id' INT UNSIGNED AUTO_INCREMENT NOT NULL,
   'title' VARCHAR(255) NOT NULL,
   'body' TEXT NOT NULL,
   'created' DATETIME NOT NULL,
   'modified' DATETIME NOT NULL,
   PRIMARY KEY('id')
);
```

Teraz dodaj kilka rekordów, korzystając z poniższych zapytań SQL:

```
INSERT INTO 'articles'('title', 'body', 'created', 'modified') VALUES
   ('First article', 'The body of the first article.', NOW(), NOW()),
   ('Second article', 'The body of the second article.', NOW(), NOW()),
   ('Third article', 'The body of the third article.', NOW(), NOW());
```

Utwórz plik *app/controllers/articles_controller.php* i umieść w nim kontroler o następującej treści:

```
<?php
class ArticlesController extends AppController {
   public function index() {
      $this->paginate['limit'] = 2;
      $articles = $this->paginate();
      $this->set(compact('articles'));
   }
   public function add() {
      if (!empty($this->data)) {
         $this->Article->create();
         if ($this->Article->save($this->data)) {
            $this->Session->setFlash('Article saved!');
            $this->redirect(array('action'=>'index'));
         } else {
            $this->Session->setFlash('Please correct errors!');
         }
      }
```

```
    }
    public function view($id) {
        $article = $this->Article->find('first', array(
            'conditions' => array('Article.id' => $id)
        ));
        if (empty($article)) {
            $this->cakeError('error404');
        }
        $this->set(compact('article'));
    }
}
?>
```

W dalszej kolejności utwórz plik *app/models/article.php* o następującej treści:

```
<?php
class Article extends AppModel {
    public $validate = array(
        'title' => 'notEmpty',
        'body' => 'notEmpty'
    );
}
?>
```

Utwórz podkatalog *app/views/articles*, po czym dodaj do niego plik *index.ctp* o następującej treści:

```
<h1>Articles</h1>
<p>
<?php echo $this->Paginator->counter(); ?>
 - 
<?php echo $this->Paginator->prev(); ?>
 
<?php echo $this->Paginator->numbers(); ?>
 
<?php echo $this->Paginator->next(); ?>
</p>
<p>
<?php echo count($articles) . ' articles: '; ?>
</p>
<ul>
<?php foreach($articles as $article) { ?>
    <li><?php echo $this->Html->link(
        $article['Article']['title'],
        array('action'=>'view', $article['Article']['id'])
    ); ?></li>
<?php } ?>
</ul>
<p><?php echo $this->Html->link('Create article', array('action'=>'add'));
?></p>
```

Do katalogu *app/views/articles* dodaj plik *add.ctp* o następującej treści:

```
<?php
echo $this->Form->create();
echo $this->Form->inputs(array(
   'legend' => 'Create article',
   'title' => array('label' => 'Title'),
   'body' => array('label' => 'Body')
));
echo $this->Form->end('Save');
?>
```

Na zakończenie dodaj plik *app/views/articles/view.ctp* o następującej treści:

```
<h1><?php echo $article['Article']['title']; ?></h1>
<?php echo $article['Article']['body']; ?>
```

Jak to zrobić

1. Zmodyfikuj treść pliku *app/controllers/articles_controller.php*, uwzględniając zmiany zaznaczone pogrubioną czcionką w metodzie `add()`:

```
public function add() {

   if (!empty($this->data)) {
   $this->Article->create();
   if ($this->Article->save($this->data)) {
      $this->Session->setFlash(__('Article saved', true));
      $this->redirect(array('action'=>'index'));
   } else {
      $this->Session->setFlash(__('Please correct the errors', true));
   }
}
}
```

2. Otwórz plik *app/views/articles/add.ctp* i wprowadź zaznaczone pogrubioną czcionką zmiany:

```
<?php
echo $this->Form->create();
echo $this->Form->inputs(array(
   'legend' => __('New Article', true),
   'title' => array('label' => __('Title:', true)),
   'body' => array('label' => __('Body:', true))
));
echo $this->Form->end(__('Save', true));
?>
```

3. Otwórz plik *app/views/articles/index.ctp* i wprowadź następujące zmiany:

```
<h1><?php __('Articles'); ?></h1>
<p>
<?php echo $this->Paginator->counter(__('Showing records %start%-%end% in
page %page% out of %pages%', true)); ?>
 - 
<?php echo $this->Paginator->prev(__('<< Previous', true)); ?>
 
<?php echo $this->Paginator->numbers(); ?>
 
<?php echo $this->Paginator->next(__('Next >>', true)); ?>
</p>
<p>
<?php
$count = count($articles);
echo $count . ' ' . __n('article', 'articles', $count, true) . ': ';
?>
</p>
<ul>
<?php foreach($articles as $article) { ?>
   <li><?php echo $this->Html->link(
      $article['Article']['title'],
      array('action'=>'view', $article['Article']['id'])
   ); ?></li>
<?php } ?>
</ul>
<p><?php echo $this->Html->link(__('Create article', true),
array('action'=>'add')); ?></p>
```

Przejdź na stronę *http://localhost/articles*. Powinieneś otrzymać stronicowaną listę artykułów podobną do przedstawionej na poniższym rysunku.

Jak to działa

Dwie najważniejsze metody udostępniane przez CakePHP do tłumaczenia to `__()` i `__n()`. Nazwy metod mogą wydać się nieco dziwne; ich pochodzenie wywodzi się od implementacji narzędzia gettext w języku Perl — narzędzie to stanowi element Projektu Tłumaczenia GNU (GNU Translation Project).

Metoda `__()` jest używana do tłumaczenia tekstów statycznych i przyjmuje dwa parametry opisane w poniższej tabeli.

Parametr	Opis
`singular`	Tekst, który ma być przetłumaczony.
`return`	Jeśli ten parametr ma wartość `true`, przetłumaczony tekst zostanie zwrócony, a nie przesłany do klienta. Domyślnie `false`.

Metoda `__n()` również pozwala na tłumaczenie tekstów statycznych, jednak uwzględnia ona także sytuacje, w których konkretna wartość może zależeć od liczby — pojedynczej lub mnogiej. W związku z tym przyjmuje ona cztery parametry wymienione w poniższej tabeli.

Parametr	Opis
`singular`	Tekst, który zostanie przetłumaczony na aktywny język, jeśli parametr `count` otrzyma wartość `singular`.
`plural`	Tekst, który zostanie przetłumaczony na aktywny język, jeśli parametr `count` otrzyma wartość `plural`.
`count`	Zmienna lub wartość liczbowa, która zostanie wykorzystana do określenia liczby dla danego tekstu (`singular` lub `plural`).
`return`	Jeśli ten parametr ma wartość `true`, przetłumaczony tekst zostanie zwrócony, a nie przesłany do klienta. Domyślnie `false`.

Rozpoczynamy od zmiany komunikatów generowanych przez klasę `ArticlesController`, wykorzystując funkcję `__()`. Zastrzegamy, że teksty mają być zwracane, a nie przesyłane do klienta. Następnie modyfikujemy plik *add.ctp*, dzięki czemu wszystkie etykiety formularza (wraz z jego legendą) zostaną przetłumaczone.

W podobny sposób opakowujemy tytuł w widoku *index.ctp* za pomocą funkcji tłumaczenia. Następnie korzystamy z pierwszego parametru metod `counter()`, `next()` i `prev()` (stanowiących składowe klasy `PaginatorHelper`), aby przekazać przetłumaczone wersje wszystkich tekstowych elementów mechanizmu stronicowania. Na zakończenie korzystamy z funkcji `__n()`, by wybrać odpowiedni przetłumaczony tekst przy użyciu wartości zmiennej `count`.

Jeśli korzystasz z funkcji `__n()`, musisz pamiętać, że trzecim argumentem w wywołaniach tej funkcji powinna być zawsze zmienna, a nie wyrażenie (np. zawierające indeksy tablic). Wyrażenia mogą doprowadzić do zwrócenia nieoczekiwanych wyników podczas wywoływania powłoki ekstraktora (por. przykład „Ekstrakcja i tłumaczenie tekstów").

Domeny i kategorie

Funkcje tłumaczeń wykorzystywane w tym przykładzie opakowują funkcję `translate()` należącą do klasy `I18n` frameworka CakePHP. Metoda ta pozwala nie tylko na przeprowadzanie prostych tłumaczeń; dzięki niej programista może określić domenę, z której są pozyskiwane tłumaczone teksty, a także kategorię, do której należy tekst do tłumaczenia.

Domeny pozwalają na wydzielanie grup tłumaczonych tekstów do osobnych plików. Domyślnie, gdy domena nie jest określona jawnie, CakePHP korzysta z domeny `default` (domyślnej). Jeśli chcesz określić domenę, w której CakePHP powinien szukać tłumaczonego tekstu, skorzystaj z funkcji `__d()` lub `__dn()`. Wyszukanie tłumaczonego tekstu w domenie `moja_wtyczka` wyglądałoby następująco:

```
$translated = __d('moja_wtyczka', 'Hello World', true);
```

Kategorie umożliwiają jeszcze większą kontrolę nad zarządzaniem tłumaczonymi tekstami. Pozwalają one na grupowanie plików tłumaczeń w odrębnych katalogach; można także powiązać tłumaczony tekst z dodatkowymi metadanymi. Domyślnie CakePHP zakłada, że tłumaczone teksty należą do kategorii `LC_MESSAGES`. Jeśli chcesz zmienić kategorię, skorzystaj z funkcji tłumaczenia `__dc()` i `__dcn()`, ustawiając przedostatni argument — `return` — na wybraną kategorię. Może ona przyjmować jedną z określonych poniżej wartości:

- `LC_ALL`: 0;
- `LC_COLLATE`: 1;
- `LC_CTYPE`: 2;
- `LC_MONETARY`: 3;
- `LC_NUMERIC`: 4;
- `LC_TIME`: 5;
- `LC_MESSAGES`: 6.

Próba znalezienia należącego do kategorii `LC_MESSAGES` tekstu *Hello World* w domenie `default` wygląda następująco:

```
$translated = __dc('default', 'Hello World', 6, true);
```

Korzystając z kategorii, zawsze podawaj wartości liczbowe, a nie nazwy stałych. Te ostatnie są bowiem zależne od wykorzystywanej platformy.

Zobacz również

- „Internacjonalizacja komunikatów walidacji w modelach";
- „Ekstrakcja i tłumaczenie tekstów".

Internacjonalizacja komunikatów walidacji w modelach

W tym przykładzie wykorzystamy różne sposoby na zrealizowanie tego samego zadania: tłumaczenia komunikatów walidacji w modelach.

Zanim zaczniesz

Aby wykonać ten przykład, musimy skorzystać z podstawowego szkieletu aplikacji. W tym celu musisz wykonać poprzedni przykład.

Jak to zrobić

Zmień treść pliku *app/models/article.php*, wprowadzając zmiany zaznaczone pogrubioną czcionką we właściwości `validate`:

```
public $validate = array(
   'title' => array(
      'required' => 'notEmpty'
   ),
   'body' => array(
      'required' => 'notEmpty'
   )
);
```

Istnieją dwa sposoby tłumaczenia komunikatów walidacji. Pierwszy z nich wymaga przesłonięcia konstruktora modelu. Wystarczy dodać jego poniższą implementację do klasy `Article` w pliku *app/models/article.php*:

```
public function __construct($id = false, $table = null, $ds = null) {
   foreach($this->validate as $field => $rules) {
      if (!is_array($rules)) {
         $rules = (array) $rules;
      }
      foreach($rules as $key => $rule) {
```

```
            if (!is_array($rule)) {
               $rules[$key] = compact('rule');
            }
         }
         $this->validate[$field] = $rules;
      }
      $this->validate = Set::merge($this->validate, array(
         'title' => array(
            'required' => array('message'
            ↪=> __('A title must be specified', true))
         ),
         'body' => array(
            'required' => array('message'
            ↪=> __('You must define the body', true))
         )
      ));
      parent::__construct($id, $table, $ds);
   }
```

Inną metodą tłumaczenia komunikatów walidacji jest przeniesienie komunikatów do widoku. W ten sposób zamiast przesłaniać konstruktor i deklarować w nim komunikaty, wystarczy wprowadzić zmiany w pliku *app/views/articles/add.ctp*:

```
<?php
echo $this->Form->create();
echo $this->Form->inputs(array(
   'title' => array(
      'label' => __('Title:', true),
      'error' => array(
         'required' => __('A title must be specified', true)
      )
   ),
   'body' => array(
      'label' => __('Body:', true),
      'error' => array(
         'required' =>  __('You must define the body', true)
      )
   )
));
echo $this->Form->end(__('Save', true));
?>
```

Obie metody doprowadzą do uzyskania takiego samego efektu. Przejdź na stronę *http://localhost/articles/add* i wyślij formularz bez wprowadzania jakichkolwiek wartości. Powinieneś uzyskać efekt jak na poniższym rysunku.

Jak to działa

Przed utworzeniem komunikatów o błędach dla każdej z reguł walidacji musimy owe reguły nazwać. W tym celu modyfikujemy model `Article`, dzięki czemu każda z reguł jest indeksowana przy użyciu nazwy. W naszym przypadku wybieramy nazwę `required` dla reguły CakePHP o nazwie `notEmpty`.

Pierwsze rozwiązanie problemu tłumaczenia komunikatów walidacji można określić mianem scentralizowanego — wszystkie komunikaty są umieszczone w tym samym miejscu, w konstruktorze modelu. Przesłaniamy konstruktor, dzięki czemu możemy w jego wnętrzu zadeklarować komunikaty o błędach, które powinny być przetłumaczone. Musieliśmy skorzystać z konstruktora, ponieważ właściwości klas mogą zawierać tylko statyczne przypisania. Poniższy blok kodu spowoduje błąd składni PHP:

```
public $validate = array(
   'title' => array(
      'required' => array(
         'rule' => 'notEmpty',
         'message' => __('Nothing defined!', true) // BŁĄD SKŁADNI
```

```
            )
        )
    );
```

W implementacji konstruktora rozpoczynamy od sprawdzenia, czy właściwość `validate` stanowi tablicę reguł (indeksowanych przy użyciu nazw pól). Musimy także sprawdzić, czy każda reguła sama w sobie również stanowi tablicę (indeksowaną za pomocą nazw), której wartościami są ponownie tablice, zawierające przynajmniej ustawienie `rule`.

Po zweryfikowaniu formatu właściwości `validate` możemy połączyć komunikaty walidacji dla każdej z reguł, korzystając z funkcji `__()` w celu przetłumaczenia komunikatów. Na zakończenie wywołujemy konstruktor klasy bazowej, aby poprawnie zakończyć proces tworzenia całego modelu.

Drugie podejście do problemu tłumaczenia przedstawione w tym przykładzie przesuwa odpowiedzialność za obsługę tłumaczeń na widok, korzystając z ustawienia `error` dostępnego w metodzie `input()` klasy `FormHelper`. To ustawienie otrzymuje tablicę indeksowaną za pomocą nazw reguł walidacji, której wartościami są komunikaty o błędach wykorzystywane w razie niespełnienia poszczególnych reguł walidacji.

Zobacz również

- „Ekstrakcja i tłumaczenie tekstów".

Tłumaczenie tekstów zawierających dynamicznie generowaną treść

W tym przykładzie dowiesz się, jak tłumaczyć teksty, które zawierają elementy dynamiczne — np. wartości zmiennych.

Zanim zaczniesz

Aby wykonać ten przykład, musimy skorzystać z podstawowego szkieletu aplikacji. W tym celu musisz wykonać przykład „Internacjonalizacja tekstów w kontrolerach i widokach".

Jak to zrobić

1. Otwórz plik *app/controllers/articles_controller.php* i wprowadź zaznaczone pogrubioną czcionką zmiany w metodzie `add()`:

```
public function add() {
   if (!empty($this->data)) {
      $this->Article->create();
      if ($this->Article->save($this->data)) {
         $this->Session->setFlash(
            sprintf(__('Article "%s" saved', true), $this->Article->
            ↪field('title'))
         );
         $this->redirect(array('action'=>'index'));
      } else {
         $this->Session->setFlash('Proszę poprawić błędy!');
      }
   }
}
```

2. Otwórz plik *app/views/articles/index.ctp* i wprowadź w nim zaznaczone pogrubioną czcionką zmiany:

```
<h1><?php __('Articles'); ?></h1>
<p>
<?php echo $this->Paginator->counter(__('Showing records %start%-
%end% in page %page% out of %pages%', true)); ?>
 - 
<?php echo $this->Paginator->prev(__('<< Previous', true)); ?>
 
<?php echo $this->Paginator->numbers(); ?>
 
<?php echo $this->Paginator->next(__('Next >>', true)); ?>
</p>
<p>
<?php
$count = count($articles);
printf(__n('%d article', '%d articles', $count, true), $count);
?>
</p>
<ul>
<?php foreach($articles as $article) { ?>
   <li><?php echo $this->Html->link(
      $article['Article']['title'],
      array('action'=>'view', $article['Article']['id'])
   ); ?></li>
<?php } ?>
</ul>
<p><?php echo $this->Html->link(__('Create article', true),
array('action'=>'add')); ?></p>
```

Jak to zrobić

Gdy podczas tworzenia aplikacji napotyka się problem tłumaczenia treści zawierających elementy dynamiczne — np. wartość zmiennej lub wartość pola z bazy danych — może się pojawić pokusa dołączenia zmiennej do łańcucha statycznego, a następnie przekazania takiego wyrażenia do funkcji tłumaczenia:

```
$translated = __('Hello ' . $name, true); // ŹLE
```

To wyrażenie nie jest poprawne, ponieważ ekstraktor CakePHP (omówiony w przykładzie „Ekstrakcja i tłumaczenie tekstów") działa poprawnie tylko dla tekstów statycznych. W innych językach może na przykład wystąpić konieczność zmiany kolejności słów w zdaniu. W związku z tym musimy skorzystać z innej techniki przetwarzania łańcuchów. Rozwiązanie jest proste i stosunkowo popularne — funkcje PHP `printf()` i `sprintf()`.

Obie funkcje przyjmują te same argumenty. Pierwszy z nich jest obowiązkowy i określa łańcuch znaków do sformatowania. Wszystkie kolejne argumenty przekazane do funkcji zostaną wykorzystane do wygenerowania wynikowego łańcucha znaków. Jedyna różnica pomiędzy funkcjami `printf()` a `sprintf()` polega na tym, że pierwsza z nich wyświetli efekt swojej pracy, druga zaś — zwróci go.

Przejdźmy teraz do kodu naszej aplikacji. Rozpoczynamy od zmiany komunikatu zwracanego przez klasę `ArticlesController` po utworzeniu artykułu. Korzystamy z funkcji `sprintf()`, ponieważ efekt jej działania chcemy przekazać do metody `setFlash()` komponentu `Session`. W naszej sytuacji wyrażenie `%s` pozwala na wstawienie do łańcucha znaków tytułu nowo utworzonego artykułu.

W podobny sposób podstawiamy wartość zmiennej `count` pod ciąg `%d`. Tym razem korzystamy z funkcji `printf()`, aby wyświetlić od razu efekt działania funkcji.

Zmiana kolejności argumentów

Gdy korzystamy z wyrażeń `%s` lub `%d` w funkcjach `printf()` i `sprintf()`, nie mamy kontroli nad sposobem pozycjonowania wartości; nie możemy też użyć dwa razy jednej wartości, ponieważ każde z wyrażeń jest dopasowywane do konkretnego, pojedynczego argumentu. Załóżmy, że dysponujemy następującym wyrażeniem:

```
printf('Your name is %s and your country is %s', $name, $country);
```

Pierwsze wyrażenie `%s` zostanie zastąpione wartością zmiennej `name`, drugie — wartością zmiennej `country`. Problem pojawiłby się w sytuacji, w której chcielibyśmy zmienić kolejność argumentów w łańcuchu znaków, zachowując jednocześnie kolejność argumentów w obrębie wywołania funkcji `printf()`.

Na szczęście do argumentów funkcji `printf()` możemy się odwoływać, korzystając z ich numerów porządkowych (określających ich pozycję wśród wszystkich argumentów przekazanych w danym wywołaniu). W poniższym przykładzie `name` jest argumentem `1`, a `country` — argumentem `2`:

```
printf('You are from %2$s and your name is %1$s', $name, $country);
```

Takie podejście pozwala na ponowne użycie argumentu bez konieczności podawania go wielokrotnie w wywołaniu funkcji `printf()`:

```
printf('You are from %2$s and your name is %1$s . Welcome %1$s!', $name,
↪$country);
```

Zobacz również

- „Ekstrakcja i tłumaczenie tekstów".

Ekstrakcja i tłumaczenie tekstów

W tym przykładzie nauczymy się pozyskiwać wszystkie łańcuchy znaków, które podlegają tłumaczeniu w naszych aplikacjach CakePHP, a następnie przeprowadzimy proces tłumaczenia, korzystając z darmowego oprogramowania.

Zanim zaczniesz

Aby wykonać ten przykład, musimy skorzystać z podstawowego szkieletu aplikacji. W tym celu musisz wykonać przykład „Internacjonalizacja tekstów w kontrolerach i widokach".

Musisz także zainstalować aplikację **Poedit** w swoim systemie. Przejdź na stronę *http://www.poedit.net/download.php*, a następnie pobierz plik dla swojego systemu operacyjnego.

Jak to zrobić

Przejdź do podkatalogu *app/* Twojej aplikacji w wierszu poleceń, a następnie wykonaj poniższe polecenie:

- jeśli pracujesz w systemach GNU Linux/Mac/Unix:

```
../cake/console/cake i18n extract
```

- jeśli jesteś użytkownikiem systemu Microsoft Windows:

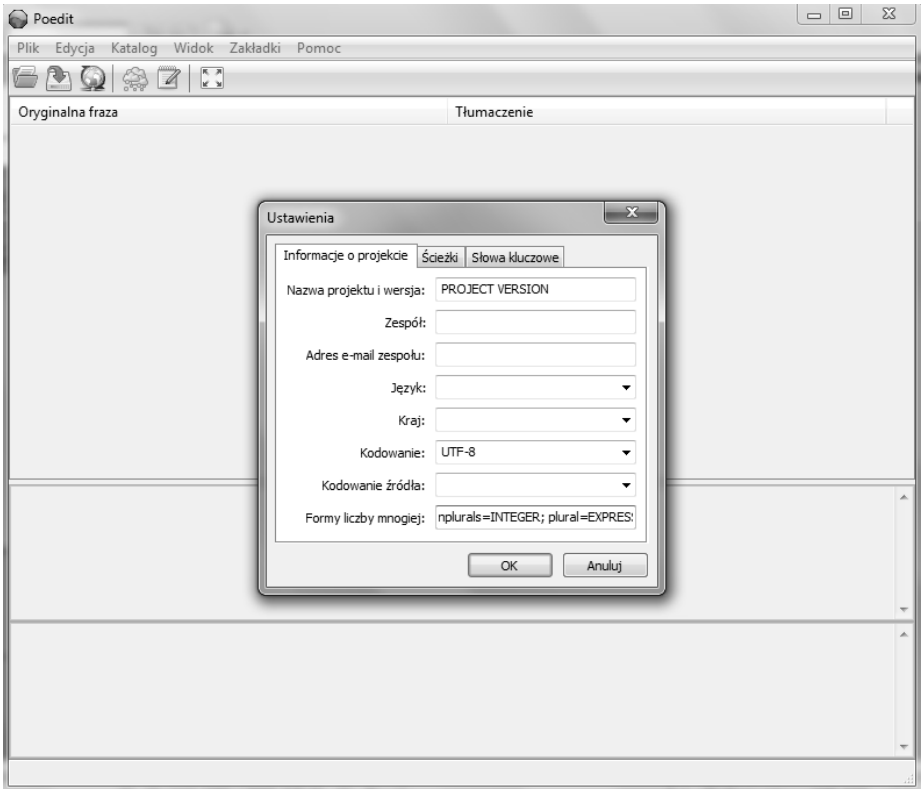
```
..\cake\console\cake.bat i18n extract
```

Powinieneś skorzystać z ustawień domyślnych, jak przedstawiono na kolejnym rysunku.

Po udzieleniu odpowiedzi na ostatnie pytanie powłoka przeszuka wszystkie pliki Twojej aplikacji i na ich podstawie wygeneruje szablon tłumaczenia. Zostanie on umieszczony w pliku *app/locale/default.pot*.

```
C:\wamp\www\app>..\cake\console\cake.bat i18n extract

?
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : app
Path: C:\wamp\www\app
---------------------------------------------------------------
What is the full path you would like to extract?
Example: C:\wamp\www\myapp
[Q]uit [D]one
[C:\wamp\www\app] >

What is the full path you would like to extract?
Example: C:\wamp\www\myapp
[Q]uit [D]one
[D] > D

What is the full path you would like to output?
Example: C:\wamp\www\app\locale
[Q]uit
[C:\wamp\www\app\locale] >

Would you like to merge all domains strings into the default.pot file? (y/n)
[n] > _
```

Otwórz plik Poedit, a następnie wybierz opcję *Nowy katalog z pliku POT* z menu *Plik*. Program wyświetli okno wyboru pliku. Przejdź do podkatalogu *app/locale* Twojej aplikacji, zaznacz plik *default.pot* i kliknij przycisk *Otwórz*. Zostanie wyświetlone okno ustawień przedstawione na poniższym rysunku.



W oknie *Ustawienia* wprowadź nazwę projektu i informacje z nim związane. W polu *Formy liczby mnogiej* powinieneś wprowadzić wyrażenie, przy użyciu którego Poedit będzie w stanie rozpoznać tłumaczenia dla liczb mnogich. W wielu językach (np. angielskim, hiszpańskim, niemieckim i portugalskim) wystarczy wprowadzić poniższe wyrażenie:

```
nplurals=2; plural=(n != 1);
```

> Więcej informacji na temat liczb mnogich i wartości, które powinno się wobec nich stosować (w zależności od tłumaczonego języka) znajdziesz na stronie *http://drupal.org/node/17564*.

Po wprowadzeniu wszystkich istotnych informacji kliknij przycisk *OK*. W tym momencie program zapyta, gdzie chcesz zapisać przetłumaczony plik. Utwórz katalog o nazwie *pol* i umieść go w katalogu *app/locale/*. Wewnątrz katalogu *pol* utwórz podkatalog *LC_MESSAGES*. Następnie przy użyciu okna dialogowego programu Poedit wybierz folder *app/locale/pol/LC_MESSAGES*, po czym kliknij przycisk *Zapisz*, nie zmieniając przy tym nazwy pliku — powinna mieć wartość *default.po*.

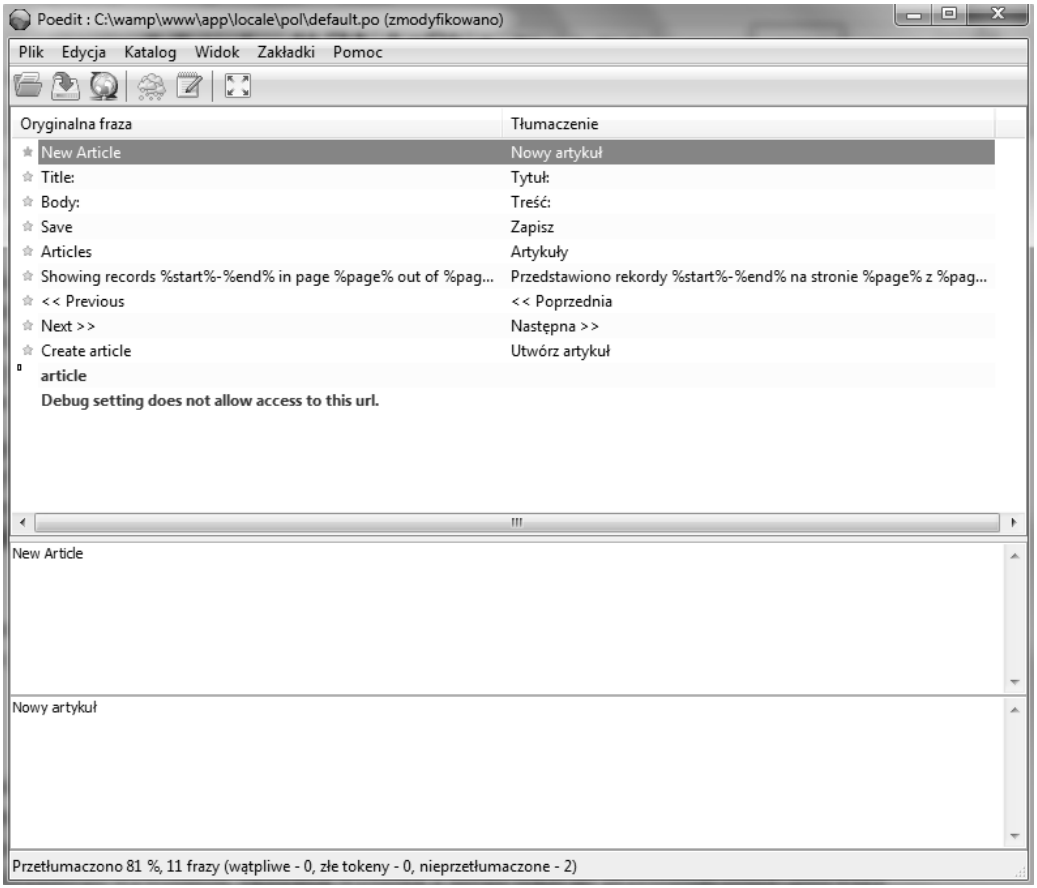
Program Poedit zaprezentuje wszystkie oryginalne łańcuchy znaków. Do każdego z nich można dodać tłumaczenie. Wystarczy wybrać jeden z oryginalnych tekstów, a następnie wprowadzić treść tłumaczenia w polu umieszczonym w dolnej części okna. Po wprowadzeniu tłumaczeń okno programu Poedit powinno wyglądać tak, jak na kolejnej rysunku.

Wybierz opcję *Zapisz* z menu *Plik*, aby zapisać przetłumaczony plik. W katalogu *app/locale/pol/LC_MESSAGES* powinny być dostępne dwa pliki: *default.po* i *default.mo*.

Jak to działa

Na samym początku ekstraktor CakePHP musi otrzymać ścieżki do katalogów, które mają być przez niego przetworzone. Następnie mechanizm rekursywnie przegląda wszystkie katalogi, próbując znaleźć wszystkie wywołania funkcji tłumaczenia (`__()`, `__n()`, `__d()`, `__dn()`, `__dcn()` i `__c()`) w plikach PHP i plikach widoków. Dla każdego znalezionego wywołania ekstraktor wyłuska z niego tekst do przetłumaczenia (pierwszy argument wywołań funkcji `__n()` i `__c()`; drugi argument wywołań funkcji `__d()` i `__dc()`; pierwszy i drugi argument wywołań funkcji `__n()`; drugi i trzeci argument wywołań funkcji `__dn()` i `__dcn()`).

> W argumentach, które są wyłuskiwane przez ekstraktor, należy stosować jedynie statyczne łańcuchy znaków PHP. Nie można w żadnym przypadku używać wyrażeń PHP. Jeśli w tłumaczonych tekstach chcesz stosować zmienne lub inne wyrażenia dynamiczne, zapoznaj się z przykładem „Tłumaczenie tekstów zawierających dynamicznie generowaną treść".



Po pobraniu wszystkich tekstów do tłumaczenia przez ekstraktor zostaną utworzone odpowiednie szablony plików tłumaczeń. Jeśli w swojej aplikacji korzystasz z funkcji tłumaczeń, które mają związek z domenami (`__d()`, `__dn()`, `__dc()` i `__dcn()`), możesz uwzględnić wszystkie łańcuchy w jednym pliku szablonu albo umieścić każdą domenę w odrębnym pliku szablonu. Pliki szablonów mają rozszerzenie *pot*; ich nazwy są zaś nazwami domen (np. *default.pot* jest domyślnym (*default*) plikiem szablonu).

Jeśli otworzysz plik *default.pot* w zwykłym edytorze tekstowym, zauważysz, że rozpoczyna się on od nagłówka zawierającego wiele różnych ustawień, po którym następuje właściwa część tłumaczenia. Każdy łańcuch do tłumaczenia jest reprezentowany za pomocą dwóch linijek — pierwsza z nich zawiera element oznaczony ciągiem `msgid` (łańcuch do tłumaczenia), w drugiej zaś znajduje się ciąg `msgstr` — to właśnie tam należy wstawiać przetłumaczone łańcuchy znaków.

Program Poedit umożliwia wygodną edycję plików w formacie *pot*, a także zapisuje pliki do odpowiedniego katalogu (*app/locale/pol/LC_MESSAGES*). W katalogu tym znajdują się dwa pliki — *default.po* i *default.pot*. Jeśli otworzysz plik *.po* za pomocą zwykłego edytora, powinieneś natychmiast dostrzec podobieństwo tego pliku do pliku szablonu. Wyjątek stanowią ustawie-

nia nagłówka — zawierają one wartości określone przez nas w programie Poedit — i przede wszystkim treść tłumaczeń wprowadzonych przez nas w aplikacji Poedit. Plik *default.mo* stanowi binarną wersję pliku *default.po*, także wygenerowanego przez Poedit. Plik *default.mo* jest wykorzystywany przez CakePHP w celu szybszego przetwarzania pliku tłumaczeń.

Tłumaczenie rekordów baz danych za pomocą zachowania Translate

W tym przykładzie nauczymy się tłumaczyć rekordy z bazy danych za pomocą zachowania `Translate`.

Zanim zaczniesz

Aby wykonać ten przykład, musimy skorzystać z podstawowego szkieletu aplikacji. W tym celu musisz wykonać przykład „Internacjonalizacja tekstów w kontrolerach i widokach".

Jak to zrobić

Uruchom wiersz poleceń, a następnie przejdź do podkatalogu *app/* aplikacji i wykonaj poniższe polecenie.

- jeśli pracujesz w systemach GNU Linux/Mac/Unix:

```
../cake/console/cake i18n initdb
```

- jeśli jesteś użytkownikiem systemu Microsoft Windows:

```
..\cake\console\cake.bat i18n initdb
```

Zaakceptuj domyślne opcje. Po wybraniu wszystkich opcji powłoka utworzy tabelę `i18n`, co przedstawiono na kolejnym rysunku.

Zmodyfikuj treść pliku *app/models/article.php* zgodnie z poniższym listingiem:

```
<?php
class Article extends AppModel {
   public $validate = array(
      'title' => 'notEmpty',
      'body' => 'notEmpty'
   );
   public $actsAs = array(
      'Translate' => array('title', 'body')
   );
}
?>
```

```
C:\wamp\www\app>..\cake\console\cake.bat i18n initdb

♀
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : app
Path: C:\wamp\www\app
---------------------------------------------------------------
♀
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : app
Path: C:\wamp\www\app
---------------------------------------------------------------
Cake Schema Shell
---------------------------------------------------------------

The following table(s) will be dropped.
i18n
Are you sure you want to drop the table(s)? (y/n)
[n] >

The following table(s) will be created.
i18n
Are you sure you want to create the table(s)? (y/n)
[y] >
Creating table(s).
i18n updated.
End create.
```

Teraz musimy przenieść wartości pól `title` i `body` z tabeli `articles` do tabeli `i18n`, a następnie — usunąć oryginalne kolumny z tabeli `articles`. Wykonaj poniższe zapytania SQL:

```
INSERT INTO 'i18n'('locale', 'model', 'foreign_key', 'field', 'content')
SELECT 'eng', 'Article', 'articles'.'id', 'title', 'articles'.'title'
FROM 'articles';

INSERT INTO 'i18n'('locale', 'model', 'foreign_key', 'field', 'content')
SELECT 'eng', 'Article', 'articles'.'id', 'body', 'articles'.'body'
FROM 'articles';

ALTER TABLE 'articles'
   DROP COLUMN 'title',
   DROP COLUMN 'body';
```

Dodaj polskie tłumaczenia naszych artykułów, wykonując poniższe zapytania SQL:

```
INSERT INTO 'i18n'('locale', 'model', 'foreign_key', 'field', 'content') VALUES
   ('pol', 'Article', 1, 'title', 'Pierwszy artykuł'),
   ('pol', 'Article', 1, 'body', 'Treść pierwszego artykułu'),
   ('pol', 'Article', 2, 'title', 'Drugi artykuł'),
   ('pol', 'Article', 2, 'body', 'Treść drugiego artykułu'),
   ('pol', 'Article', 3, 'title', 'Trzeci artykuł'),
   ('pol', 'Article', 3, 'body', 'Treść trzeciego artykułu');
```

Na zakończenie wstaw poniższą instrukcję na końcu pliku *app/config/bootstrap.php*, tuż przed znacznikiem zamykającym PHP:

```
Configure::write('Config.language', 'eng');
```

Przejdź na stronę *http://localhost/articles*. Powinieneś zobaczyć ten sam wykaz artykułów, który przedstawiono w pierwszym przykładzie w tym rozdziale.

Jak to działa

Rozpoczynamy od utworzenia tabeli wymaganej przez zachowanie `Translate`, korzystając z powłoki `i18n`. Nosi ona taką samą nazwę i zawiera (poza kluczem głównym) pola opisane w poniższej tabeli.

Pole	Opis
`locale`	Język tłumaczenia danego rekordu.
`model`	Model, do którego należy tłumaczony rekord.
`foreign_key`	ID (klucz główny) w modelu, który identyfikuje tłumaczony rekord.
`field`	Nazwa pola, które podlega tłumaczeniu.
`content`	Przetłumaczona wartość dla danego pola.

Następnie możemy dodać zachowanie `Translate` do modelu `Article` i ustawić je tak, aby były tłumaczone pola `title` i `body`. Taki zapis sprawi, że pola te nie będą wchodziły w skład tabeli `articles` — od tego momentu będą one przechowywane w tabeli `i18n`. Korzystając z wartości `model` i `foreign_key` w tabeli `i18n`, zachowanie `Translate` pobierze odpowiednie wartości dla danych pól przy każdym pobraniu rekordu modelu `Article` (dla aktywnego języka).

Kopiujemy wartości pól `title` i `body` do tabeli `i18n`, dzięki czemu możemy usunąć pola z tabeli `articles`. Wywołanie funkcji `call()` wykorzystywane w klasie `ArticlesController` nie wymaga żadnych zmian. Co więcej, proces tworzenia artykułów będzie przebiegał bez żadnych modyfikacji, ponieważ zachowanie `Translate` skorzysta z aktywnego języka podczas zapisywania rekordów modelu `Article`.

Na zakończenie musimy wybrać język domyślny. W tym celu korzystamy z ustawienia `Config.language`. Pominięcie tego kroku spowoduje ustawienie języka na podstawie wartości nagłówka `HTTP_ACCEPT_LANGUAGE` wysyłanego przez przeglądarki.

Wykorzystywanie odrębnych tabel tłumaczenia

Wszystkie modele, które korzystają z zachowania `Translate`, będą zapisywać tłumaczenia swoich pól domyślnie do tabeli `i18n`. Takie zachowanie może być nieco problematyczne, jeśli dysponujemy dużą liczbą rekordów lub dużą liczbą tłumaczonych modeli. Na szczęście zachowanie `Translate` pozwala na skonfigurowanie innego modelu tłumaczenia.

W ramach przykładu zapiszemy wszystkie tłumaczenia związane z artykułami do tabeli `article_translations`. Utwórz tabelę, a następnie skopiuj rekordy z tabeli `i18n`, korzystając z poniższych zapytań SQL:

```
CREATE TABLE 'article_translations'(
   'id' INT UNSIGNED AUTO_INCREMENT NOT NULL,
   'model' VARCHAR(255) NOT NULL,
   'foreign_key' INT UNSIGNED NOT NULL,
   'locale' VARCHAR(6) NOT NULL,
   'field' VARCHAR(255) NOT NULL,
   'content' TEXT default NULL,
   KEY 'model__foreign_key'('model', 'foreign_key'),
   KEY 'model__foreign_key__locale'('model', 'foreign_key', 'locale'),
   PRIMARY KEY('id')
);
INSERT INTO 'article_translations'
SELECT 'id', 'model', 'foreign_key', 'locale', 'field', 'content'
FROM 'i18n';
```

Utwórz plik *app/models/article_translation.php* i wstaw do niego poniższy kod:

```
<?php
class ArticleTranslation extends AppModel {
   public $displayField = 'field';
}
?>
```

Właściwość `displayField` w modelu tłumaczeń poinformuje zachowanie `Translate` o tym, które pole tabeli przechowuje nazwę tłumaczonego pola.

Na zakończenie wprowadź zaznaczone pogrubioną czcionką zmiany w pliku *app/models/article.php*:

```
<?php
class Article extends AppModel {
   public $validate = array(
      'title' => 'notEmpty',
      'body' => 'notEmpty'
   );
   public $actsAs = array(
      'Translate' => array('title', 'body')
   );
   public $translateModel = 'ArticleTranslation';
}
?>
```

Zobacz również

- „Ustawianie i zapamiętywanie języka”.

Ustawianie i zapamiętywanie języka

W tym przykładzie dowiesz się, jak udostępnić użytkownikom możliwość zmiany języka aplikacji, a także jak zapamiętać ich wybór za pomocą ciasteczek (ang. *cookies*).

Zanim zaczniesz

Aby wykonać ten przykład, niezbędna będzie w pełni umiędzynarodowiona aplikacja. W tym celu musisz wykonać cały przykład „Tłumaczenie rekordów baz danych za pomocą zachowania Translate".

Musimy także skorzystać z układu aplikacji, który możemy modyfikować. Skopiuj plik *default.ctp* z katalogu *cake/libs/view/layouts* do katalogu *app/views/layouts*.

Jak to zrobić

1. Dodaj poniższe instrukcje na końcu pliku *app/config/bootstrap.php*, tuż przed znacznikiem zamykającym PHP:

```
Configure::write('Config.languages', array(
   'eng' => __('English', true),
   'pol' => __('Polski', true)
));
```

2. Zmień układ pliku *app/views/layouts/default.ctp*, dodając w wybranym przez siebie miejscu listę języków (np. przed wywołaniem metody `flash()` w komponencie `Session`):

```
<div style="float: right">
<?php
$links = array();
$currentLanguage = Configure::read('Config.language');
foreach(Configure::read('Config.languages') as $code => $language)
{
   if ($code == $currentLanguage) {
      $links[] = $language;
   } else {
      $links[] = $this->Html->link($language, array('lang' => $code));
   }
}
echo implode(' - ', $links);
?>
</div>
```

Wykorzystywane w kodzie ustawienie `Config.language` zostało ustawione w pliku *app/config/bootstrap.php* w ramach przykładu „Tłumaczenie rekordów baz danych za pomocą zachowania Translate".

3. Utwórz plik *app/app_controller.php* o następującej treści:

```
<?php
class AppController extends Controller {
   public $components = array('Language', 'Session');
}
?>
```

4. Utwórz plik *app/controller/components/language.php* o następującej treści:

```
<?php
class LanguageComponent extends Object {
   public $controller = null;
   public $components = array('Cookie');
   public $languages = array();
   public function initialize($controller) {
      $this->controller = $controller;
      if (empty($languages)) {
         $this->languages = Configure::read('Config.languages');
      }
      $this->set();
   }
   public function set($language = null) {
      $saveCookie = false;
      if (empty($language) && isset($this->controller)) {
         if (!empty($this->controller->params['named']['lang'])) {
            $language = $this->controller->params['named']['lang'];
         } elseif (!empty($this->controller->params['url']['lang'])) {
            $language = $this->controller->params['url']['lang'];
         }
         if (!empty($language)) {
            $saveCookie = true;
         }
      }
      if (empty($language)) {
         $language = $this->Cookie->read('language');
         if (empty($language)) {
            $saveCookie = true;
         }
      }
      if (empty($language) && !array_key_exists($language, $this->
      ↪languages)) {
         $language = Configure::read('Config.language');
      }
      Configure::write('Config.language', $language);
      if ($saveCookie) {
```

```
            $this->Cookie->write('language', $language, false, '1 year');
        }
    }
}
?>
```

Przejdź na stronę *http://localhost/articles*. Powinieneś zobaczyć listę artykułów, a w prawym górnym rogu powinno się pojawić łącze, które pozwoli na zmianę aktywnego języka na polski. Kliknięcie łącza spowoduje wyświetlenie polskich wersji artykułów. Efekt został przedstawiony na poniższym rysunku.

Jak to działa

Zaczynamy od zadeklarowania listy wszystkich dostępnych języków, dzięki czemu możemy bez problemu umieścić łącze do zmiany aktywnego języka. Lista służy nam do stworzenia listy hiperłączy, która następnie jest umieszczana w pliku układu *default.ctp*. Jednocześnie faktyczne łącza (a nie teksty) są generowane dla wszystkich języków poza aktywnym.

Aktywny język ustawiamy w zmiennej konfiguracyjnej CakePHP o nazwie `Config.language`. Otrzymuje ona pewną wartość — w naszym przypadku *eng* — w pliku konfiguracyjnym *bootstrap.php*. Jeśli konieczna jest zmiana języka, wartość tego ustawienia powinna ulec zmianie przed pierwszym użyciem funkcji tłumaczenia.

Aby zachować porządek w kontrolerze, postanowiliśmy utworzyć komponent `Language`, który obsługuje zmianę języka. Komponent ten przeszuka parametry nazwane, a także te przekazane w adresie URL, pod kątem parametru `lang`. Jeśli język nie został określony w ten sposób, komponent spróbuje znaleźć język, korzystając z ciasteczka.

Jeśli nie zostało ustawione żadne ciasteczko, a także w sytuacji, gdy nastąpiło żądanie zmiany języka, komponent zapisze aktywny język w ciasteczku `language`, które będzie przechowywane przez okres jednego roku.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Ź

CakePHP 1.3
Receptury

CakePHP jest jednym z tych frameworków dla języka PHP, które dzięki swoim licznym zaletom już zdobyły dużą popularność wśród programistów. Pozwala na błyskawiczne tworzenie skalowalnych aplikacji internetowych, korzystających ze wzorca MVC oraz zaawansowanych narzędzi (na przykład mapowania obiektowo-relacyjnego baz danych). W tej książce znajdziesz sześćdziesiąt gotowych przepisów na rozwiązanie różnego rodzaju problemów pojawiających się podczas pracy z CakePHP. Część przedstawionych receptur poświęcono bezpieczeństwu, a część współpracy z bazami danych czy wykorzystaniu technologii AJAX. Ponadto podczas lektury nauczysz się korzystać z geolokalizacji, usług REST oraz funkcji pomocnych przy testowaniu. Poznasz przepis na stworzenie aplikacji obsługującej wiele języków oraz dowiesz się więcej o współpracy z powłoką systemu. Ta pełna gotowych rozwiązań książka powinna znaleźć się na półce każdego programisty PHP używającego CakePHP!

Sięgnij po skuteczne rozwiązania najczęstszych problemów z CakePHP!

Ta książka pozwoli Ci...

- projektować eleganckie i skalowalne aplikacje webowe z wykorzystaniem CakePHP
- rozszerzać możliwości wyszukiwania za pomocą wirtualnych pól, zapytań ad hoc i własnych typów wyszukiwania
- włączać międzynarodową obsługę aplikacji, w tym tłumaczenie rekordów w bazach danych
- automatyzować zadania niewymagające interakcji ze strony użytkowników, które można uruchamiać z poziomu konsoli
- zabezpieczać aplikacje za pomocą systemów uwierzytelniania, z wykorzystaniem nazwy użytkownika lub adresu e-mail, a także zapamiętać szczegóły kont użytkowników w systemie
- wykorzystać zachowanie Containable do współpracy z wiązaniami modeli
- tworzyć usługi sieciowe w różnej postaci i korzystać z nich

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

ISBN 978-83-246-3542-9

9 788324 635429

Cena: 57,00 zł

Informatyka w najlepszym wydaniu