



Technologia i rozwiązania

CakePHP 1.3

Receptury

Poznaj optymalne przepisy na CakePHP!

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



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

Copyright © Packt Publishing 2011. First published in the English language under the title „CakePHP 1.3 Application Development Cookbook”

© Helion 2012
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 biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

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

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

Spis treści

O autorze	9
O recenzentach	11
Przedmowa	13
O czym jest ta książka?	13
Oprogramowanie wykorzystywane w książce	16
Dla kogo jest ta książka?	17
Konwencje typograficzne	17
Materiały dodatkowe i pomoc	17
Rozdział 1. Uwierzytelnianie	19
Wprowadzenie	19
Konfiguracja prostego systemu uwierzytelniania	20
Używanie i konfiguracja komponentu Auth	24
Logowanie za pomocą nazwy użytkownika lub adresu e-mail	28
Zapisywanie informacji o użytkowniku po zalogowaniu	31
Pobieranie informacji o zalogowanym użytkowniku	33
Używanie prefiksów do kontroli dostępu bazującej na rolach	36
Autoryzacja wykorzystująca warstwę kontroli dostępu (ACL)	38
Integracja z OpenID	45
Rozdział 2. Wiązania modeli	49
Wprowadzenie	49
Dodanie zachowania Containable do wszystkich modeli	50
Ograniczanie wiązań zwracanych przez wyszukiwania	51
Modyfikowanie parametrów wiązań dla wyszukiwań	59
Modyfikowanie warunków wiązań dla wyszukiwań	63
Zmiana typu złączenia dla powiązań jeden-do-jednego	65
Tworzenie wielu powiązań z tym samym modelem	66
Dodawanie wiązań w locie	69

Rozdział 3. Wszystko o pobieraniu danych	73
Wprowadzenie	73
Wykonywanie zapytań GROUP i COUNT	74
Wykorzystywanie pól wirtualnych	80
Tworzenie zapytań z wykorzystaniem złączeń doraźnych	84
Wyszukiwanie elementów spełniających określone kryteria	87
Implementacja własnego typu wyszukiwania	89
Stronicowanie wyszukiwań własnych typów	93
Implementacja stronicowania na bazie technologii AJAX	96
Rozdział 4. Walidacja i zachowania	99
Wprowadzenie	99
Dodawanie wielu reguł walidacji	100
Tworzenie własnych reguł walidacji	104
Wykorzystywanie wywołań zwrotnych w zachowaniach	109
Wykorzystywanie zachowań do dodawania nowych pól	116
Wykorzystywanie zachowania Sluggable	118
Geokodowanie adresów przy użyciu zachowania Geocodable	122
Rozdział 5. Źródła danych	127
Wprowadzenie	127
Udoskonalanie dziennika zapytań źródła danych SQL	127
Parsowanie plików CSV za pomocą źródeł danych	134
Konsumowanie kanałów RSS za pomocą źródeł danych	138
Tworzenie źródła danych przy użyciu serwisu Twitter	142
Dodawanie obsługi transakcji i blokad w źródle danych MySQL	152
Rozdział 6. Magia trasowania	161
Wprowadzenie	161
Wykorzystywanie parametrów named i GET	162
Wykorzystywanie tras z prefiksami	168
Praca z elementami tras	172
Dodawanie tras typu catch-all dla stron profilowych	175
Dodawanie walidacji dla klas typu catch-all	179
Tworzenie własnych klas trasowania	182
Rozdział 7. Tworzenie i wykorzystywanie usług sieciowych	187
Wprowadzenie	187
Tworzenie kanału RSS	188
Konsumowanie usługi JSON	194
Tworzenie usług REST przy użyciu formatu JSON	199
Dodawanie uwierzytelniania do usług REST	208
Implementacja autoryzacji dostępu do API przy użyciu tokenu	213

Rozdział 8. Praca z powłokami	219
Wprowadzenie	219
Tworzenie i uruchamianie powłoki	220
Parsowanie parametrów wiersza poleceń	224
Tworzenie zadań powłoki wielokrotnego użytku	229
Wysyłanie wiadomości e-mail z poziomu powłoki	239
Tworzenie automatycznych zadań za pomocą wtyczki Robot	243
Rozdział 9. Internacjonalizacja aplikacji	249
Wprowadzenie	249
Internacjonalizacja tekstów w kontrolerach i widokach	250
Internacjonalizacja komunikatów walidacji w modelach	256
Tłumaczenie tekstów zawierających dynamicznie generowaną treść	259
Ekstrakcja i tłumaczenie tekstów	262
Tłumaczenie rekordów baz danych za pomocą zachowania Translate	266
Ustawianie i zapamiętywanie języka	270
Rozdział 10. Testowanie	273
Wprowadzenie	273
Konfiguracja frameworka do testów	274
Tworzenie testowych danych i metod modeli	278
Testowanie akcji kontrolera i ich widoków	286
Wykorzystywanie zaślepek do testowania kontrolerów	290
Uruchamianie testów w konsoli	294
Rozdział 11. Narzędzia i klasy pomocnicze	297
Wprowadzenie	297
Wykorzystywanie klasy Set	298
Operacje na tekście przy użyciu klasy String	305
Wysyłanie wiadomości e-mail	308
Wykrywanie typów plików za pomocą MagicDb	314
Rzucanie i obsługa wyjątków	319
Skorowidz	325

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!');  
      }  
    }  
  }  
}
```


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));
?>
```


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
<code>singular</code>	Tekst, który ma być przetłumaczony.
<code>return</code>	Jeśli ten parametr ma wartość <code>true</code> , przetłumaczony tekst zostanie zwrócony, a nie przesłany do klienta. Domyślnie <code>false</code> .

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

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);
}

```

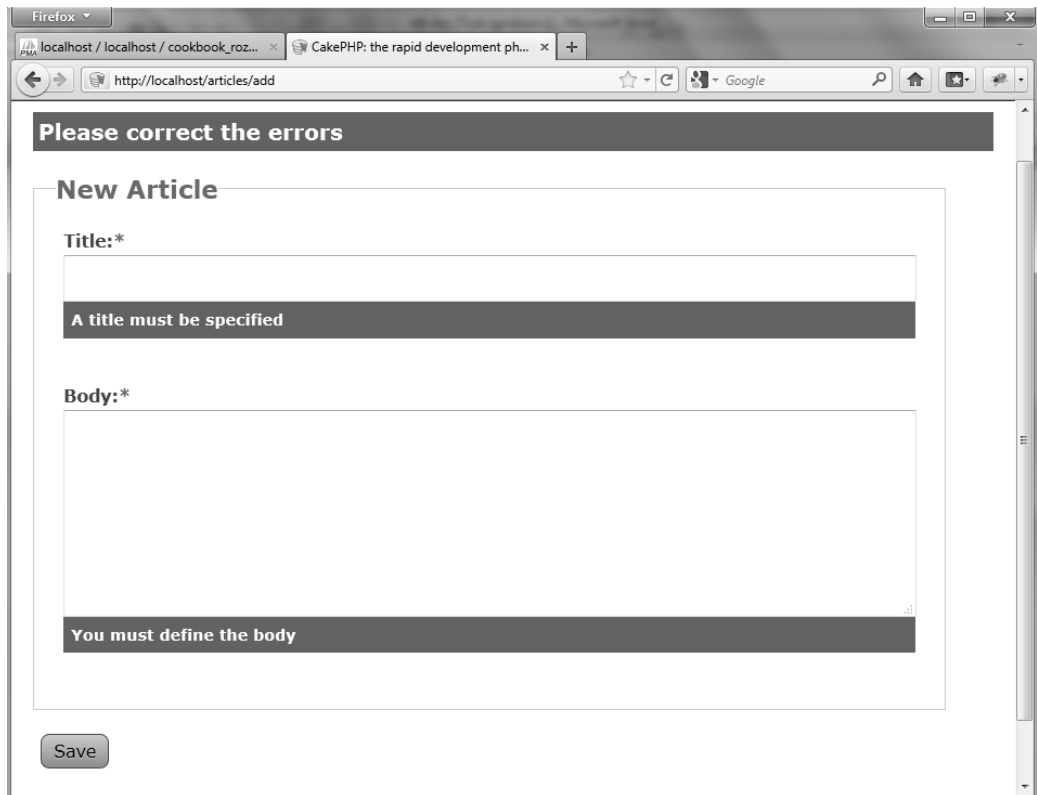
Inną metodą tłumaczenia komunikatów walidacji jest przeniesienie komunikatów do widoku. W ten sposób zamiast przesyłać konstruktor i deklarować w nim komunikaty, wystarczy wprowadzić zmiany w pliku *app/views/articles/add.ctp*:

```

<?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 przeszuwa 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()`:

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
g
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] >

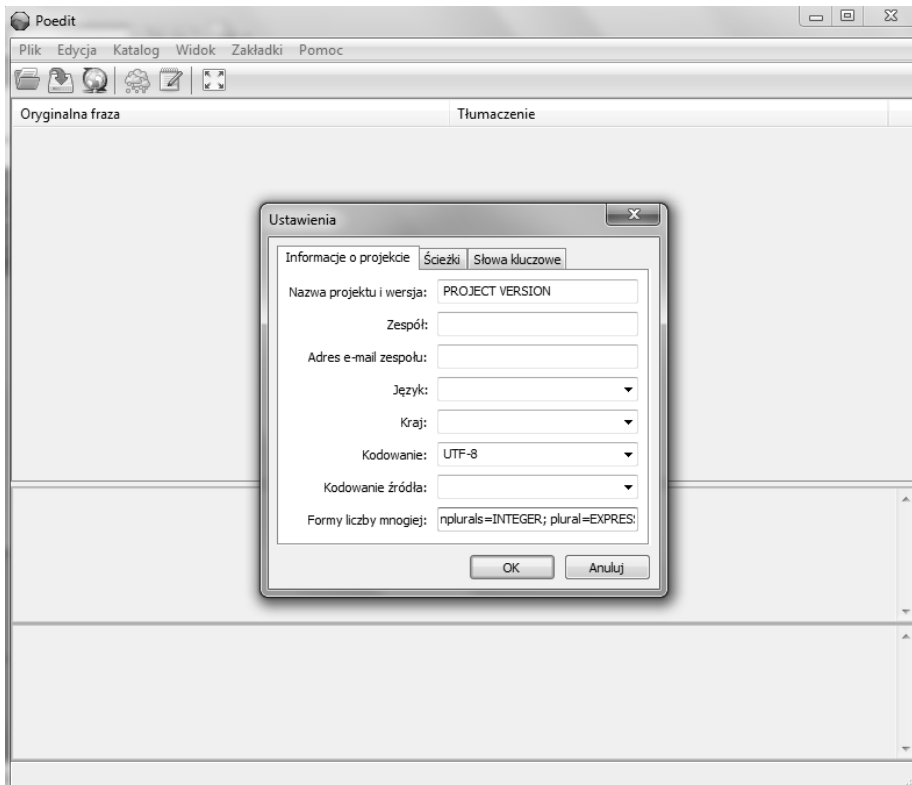
What is the full path you would like to extract?
Example: C:\wamp\www\myapp
[Q]uit [D]one
[D] > D

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] > 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*.

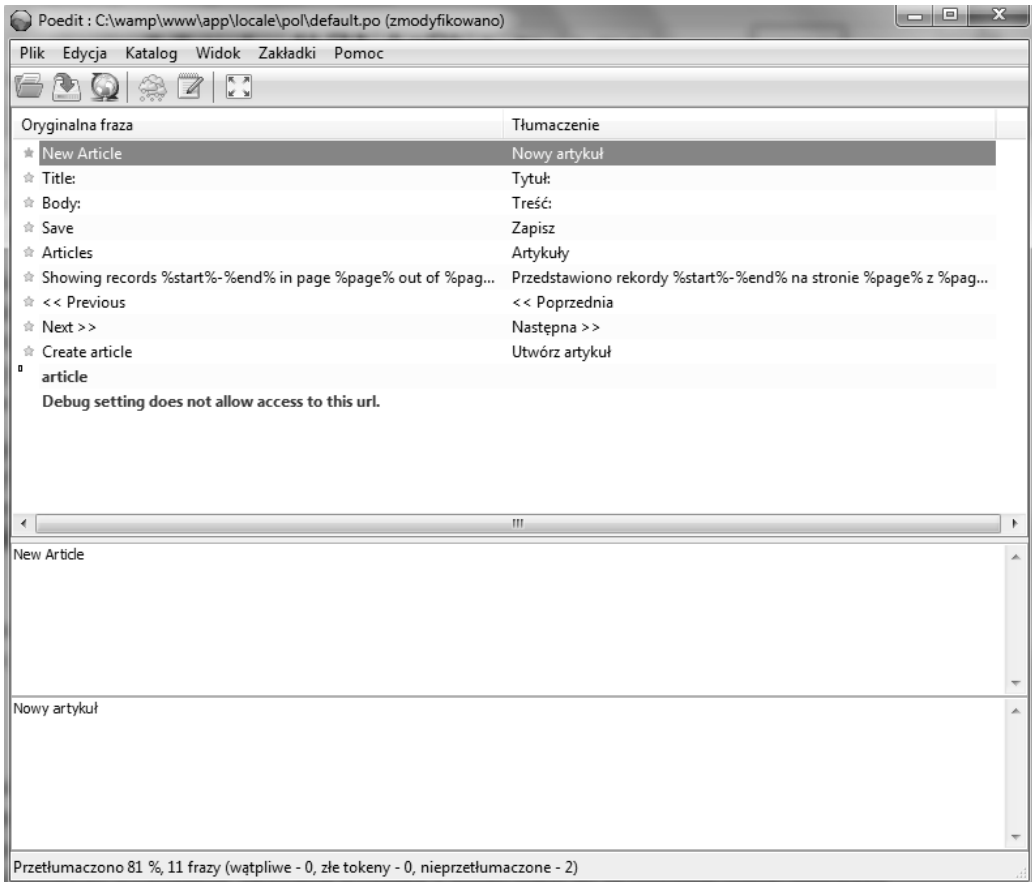
Program Poedit zaprezentuje wszystkie oryginalne łańcuchy znaków. Do każdego z nich można dodać tłumaczenie. Wystarczy wybrać jeden z oryginalnych tekstów, a następnie wprowadzić treść tłumaczenia w polu umieszczonym w dolnej części okna. Po wprowadzeniu tłumaczeń okno programu Poedit powinno wyglądać 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 znalezionej wywołania ekstraktor wyluska 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 wyluskiwane 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>
[ny] >

The following table(s) will be created.
i18n
Are you sure you want to create the table(s)? <y/n>
[yl] >
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
<code>locale</code>	Język tłumaczenia danego rekordu.
<code>model</code>	Model, do którego należy tłumaczony rekord.
<code>foreign_key</code>	ID (klucz główny) w modelu, który identyfikuje tłumaczony rekord.
<code>field</code>	Nazwa pola, które podlega tłumaczeniu.
<code>content</code>	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:

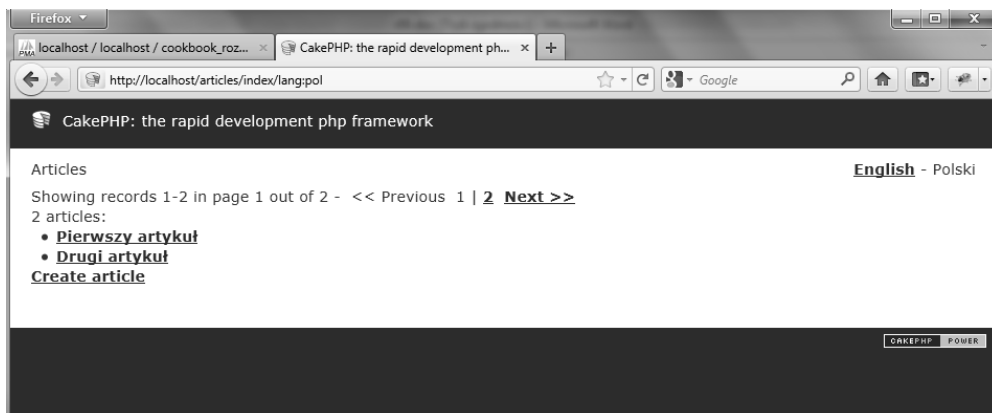
```
<?php
class LanguageComponent extends Object {
    public $controller = null;
    public $components = array('Cookie');
    public $languages = array();
    public function initialize($controller) {
        $this->controller = $controller;
        if (empty($languages)) {
            $this->languages = 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

- `__()`, 254
- `__n()`, 254, 255
- `_authorize()`, 151
- `_checkArgs()`, 228
- `_findMethods`, właściwość, 92
- `_help()`, 238
- `_helpCommand()`, 238
- `_importCSV()`, 229
- `_isJSON()`, 207
- `_parseCSV()`, 228
- `_randomPassword()`, 223
- `_restLogin()`, 212
- `_stop()`, 223, 318
- `_usageCommand()`, 238

A

- Access Control Layer, *Patrz* warstwa kontroli dostępu
- ACL, *Patrz* warstwa kontroli dostępu
- `acl_extras`, plugin, 45
- `add()`, 228
- `afterFind()`, 77, 83, 114
- AJAX, 96, 98
- `allow()`, 23
- allowedActions, parametr, 23
- `analyze()`, 318
- AppController, klasa, 24, 26
- AppException, klasa, 322, 323
- ArticlesController, klasa, 196, 254, 261
- ArticlesController::index(), 191
- `assertEqual()`, 283
- `assertFalse()`, 283
- `assertIsA()`, 283
- `assertNull()`, 283

- `assertPattern()`, 283
- `assertTags()`, 284, 289
- `assertTrue()`, 283
- Auth, komponent, 23, 24, 26, 30, 33, 212
 - haszowanie haseł, 224
 - schemat autoryzacji, 27
 - używanie i konfiguracja, 24
- authorize, parametr, 23
- automatyzacja zadań, 243
- autoryzacja
 - przy użyciu tokenu, 213
 - schemat, 27
 - wykorzystanie warstwy kontroli dostępu, 38

B

- backAutoCommit, właściwość, 159
- `beforeFilter()`, 23, 26
- `beforeFind()`, 77, 83, 114
- `beforeRender`, wywołanie zwrotne, 133
- `beforeSave`, wywołanie zwrotne, 117, 122
- `beforeValidate()`, 318
- `bindModel()`, 50, 63, 65, 71
- `blackHole()`, 213, 217

C

- Cache, klasa, 114
- CakePHP, 13, 14, 15, 16
 - ekstraktor, 261
 - konwencja nazewnicza, 68

- powłoki, 219
- reguły walidacji, 104
- tłumaczenia, 254
- trasy domyślne, 162
- wyjątki, 319
- wyszukiwanie, 73, 74
- CakeTestCase, klasa, 282, 283
- CakeTestFixture, klasa, 281
- catch-all, trasy, 175
 - dodawanie walidacji, 179
- ciasteczka, 270
- ClassRegistry::init(), 282
- `cleanInsert()`, 306
- commands, właściwość, 237, 238
- comma-separated values, *Patrz* CSV
- Config.language, 271, 272
- Configure, klasa, 292
- Configure::listObjects(), 182
- ConnectionManager::getDataSources(), 133
- `contain()`, 58
- contain, parametr, 58
 - format, 57
- Containable, zachowanie, 50, 51, 53, 55, 56, 57, 59, 63, 65
- cookies, *Patrz* ciasteczka
- COUNT, 74, 83
- Crookes, Neil, 142
- CSV, 136
 - dynamiczne ładowanie plików, 137
 - parsowanie plików, 134

D

dane testowe, 273, 278, 281
 datasources, wtyczka, 134,
 136, 139
 diff(), 305
 Dispatcher, klasa, 294
 download(), 317, 318

E

e-mail
 wysyłanie, 308
 wysyłanie z poziomu
 powłoki, 239
 Email, komponent, 239, 308,
 311, 312, 313
 właściwości, 242
 endCase(), 283
 endTest(), 283
 error(), 223
 execute(), 236, 238
 expectAtLeastOnce(), 293
 expectException(), 283
 expectNever(), 293
 expectOnce(), 293
 EXPLAIN, 133
 extract(), 237, 301

F

fclose(), 228
 fgetcsv(), 228
 fiksturki, 278, 281, 283, 284
 filter(), 305
 fixture, zadanie, 284
 fixtures, *Patrz* dane testowe
 fopen(), 228
 foreignKey, ustawienie, 68

G

Geocodable, 15, 100, 122
 Geocode, wtyczka, 122, 124, 126
 geokodowanie adresów, 122
 GET, parametr, 162, 166
 get_class_vars(), 238
 getInfo(), 323
 getLog(), 134

getMagicDb(), 318
 Google Maps, 122, 124, 126
 GROUP, 74

H

hasła
 hasz, 20
 haszowanie, 223, 224
 szyfrowanie, 24
 help(), 228, 236
 HelpTask, klasa, 236, 238
 HttpSocket, klasa, 107, 194
 HttpSocket::get(), 197
 HttpSocketOauth, klasa, 142

I

I18n, klasa, 255
 import(), 228, 237
 in(), 222, 223
 index(), 311
 initialize(), 238
 INNER JOIN, 66
 InnoDB, 152
 inputs, metoda, 23
 insert(), 306, 307
 internacjonalizacja, 249
 komunikatów walidacji
 w modelach, 256
 tekstów w kontrolerach
 i widokach, 250
 isAuthorized, metoda, 23
 isInterfaceSupported(), 133

J

JavaScript Object Notation,
Patrz JSON
 język
 ustawianie, 270
 zapamiętywanie, 270
 JOIN, złączenia, 66
 jQuery, 96
 JSON, 194
 konsumowanie usługi, 194
 json_decode(), 151, 198
 json_encode(), 207
 JSON-C, 194

K

Kairys, Donatas, 139
 konwencja nazewnictwa, 68

L

Language, komponent, 272
 layout, właściwość, 312
 LEFT JOIN, 66
 LIKE, 87, 88
 listSources(), 138
 lock(), 159
 lockTimeoutErrorCode,
 właściwość, 159
 logException(), 323
 login(), 22, 23, 30
 logout(), 22, 23
 logowanie, 28
 pobieranie informacji
 o zalogowanym
 użytkowniku, 33
 zapisanie informacji
 o użytkowniku, 31

Ł

łańcuchy znaków, 305
 łączy trwale, 118

M

MagicDb, 314, 317, 318
 main(), 222, 236
 mapowanie obiektowo-
 relacyjne, *Patrz* ORM
 merge(), 305
 mocks, *Patrz* zaślepki
 Model, klasa, 50
 Model-View-Controller,
Patrz MVC
 model-widok-kontroler,
Patrz MVC
 MVC, 196
 MyISAM, 152
 MySQL
 blokady w źródle danych, 152
 transakcje, 152

N

named parameters, *Patrz*
parametry nazwane
named, parametr, 162, 166
numeric(), 305

O

OAuth, 142, 149, 151
OpenAuth, komponent, 46, 47
OpenID, 45
biblioteka, 45
integracja, 45, 46
plugin, 46, 47
options(), 98
optymalizacja dla wyszukiwarek
internetowych, 161
ORM, 298

P

paginate(), 95
Paginator, 98
pamięć podręczna, 192
parametry nazwane, 164, 166
parsowanie
parametrów wiersza
poleceń, 224
plików CSV, 134
partial mocks, *Patrz* zaślepki
częściowe
permanent links,
Patrz łącza trwałe
PHP OAuth, biblioteka, 142
Poedit, 262, 263, 264, 265, 266
pola wirtualne, 80, 82, 83
PostsController, klasa, 193
powiązania, 49
anulowanie zmian, 58
powłoki, 219
tworzenie, 220
tworzenie zadań
wielokrotnego użytku, 229
uruchamianie, 220, 221, 222
wysyłanie wiadomości
e-mail, 239
prefiksy, 36, 168, 172
printf(), 261, 262

ProfileRoute, klasa, 184
ProfilesController, klasa, 172, 217
przypadki testowe, 273, 282
pushDiff(), 305

Q

QueryLog, komponent, 133

R

read(), 151
readfile(), 318
redirect(), 291, 292
renderException(), 323
Representational State Transfer,
Patrz REST
RequestHandler, komponent,
191, 207
requireLogin(), 212
reset(), 242
resetBindings(), 58
REST, 199, 207
dodawanie uwierzytelniania,
208
reverse(), 305
RIGHT JOIN, 66
Robot, wtyczka, 243, 246
RobotTask, model, 246
Router::connect(), 174
Router::connectNamed(), 167
Router::parseExtensions(), 190
Routing.prefixes, 172
RSS, 138, 188
a pamięć podręczna, 192
tworzenie kanału, 188
RssHelper, klasa, 192
RssHelper::item(), 192

S

schedule(), 246
schema(), 82
Search Engine Optimization,
Patrz SEO
Security, komponent, 212,
213, 217
Security.salt, parametr, 24

Security::hash(), 223, 224
send(), 242, 312, 313
SEO, 118, 161
Session, komponent, 261
Set, klasa, 298, 301, 305
Set::combine(), 303
Set::extract(), 301, 302, 303
Set::format(), 304
Set::map(), 304
Set::matches(), 137
setAutoCommit(), 160
setConfig(), 138, 141, 151
setFlash(), 261
setup(), 113
Shell, klasa, 222
SimpleTest, biblioteka, 276, 278
slug, 118, 122
Sluggable, 15, 100, 118, 122
sort(), 305
sprintf(), 261, 304
startCase(), 283
startTest(), 283
startup(), 236
Story, Mark, 45, 294
String, klasa, 305, 306
stronicowanie, 93, 95
AJAX, 96
Syrup, wtyczka, 120
szperacze sieciowe, 192

T

TagsController::view(), 162
tekst, manipulacja, 305
template, właściwość, 313
test cases, *Patrz* przypadki
testowe
test jednostkowy, 282
test_suite, 282
testAction(), 288, 289
testowanie, 273
akcji kontrolera i ich
widoków, 286
konfiguracja frameworka, 274
tworzenie danych testowych,
278
zaślepki, 290
testRedirect, właściwość, 292
testsuite, powłoka, 294

testy
 jednostkowe, 273
 uruchamianie w konsoli, 294
 timeline(), 113, 114
 tłumaczenie, 254, 262
 rekordów baz danych, 266
 tekstów zawierających
 dynamicznie generowaną
 treść, 259
 token(), 151
 tokenize(), 306, 307
 tokeny, 213, 217
 transakcje, 152
 translate(), 255
 Translate, zachowanie, 266, 268
 trasowanie, 161, 168, 172
 odwrotne, 175
 tworzenie własnych klas, 182
 Twitter, rejestracja aplikacji,
 143, 144
 TwitterAccountBehavior,
 klasa, 113

U

unbindModel(), 50, 56, 57
 uniwersalnie unikalny
 identyfikator, *Patrz* UUID
 unlock(), 159, 160
 UploadsController, klasa, 322
 User::useToken(), 218
 UsersController, klasa, 217
 UserShell, klasa, 222, 237, 238
 useToken(), 217
 usługi sieciowe, 187
 UUID, 217, 307
 uuid(), 307

uwierzytelnianie, 19
 konfiguracja prostego
 systemu, 20
 użytkownik
 pobieranie informacji, 33
 zapisanie informacji po
 zalogowaniu, 31
 zmiana domyślnego
 modelu, 27

V

validate, właściwość, 102, 259
 validateTwitter(), 106, 107
 Validation, klasa, 104
 Video::search(), 198

W

walidacja, 99
 dodawanie wielu reguł, 100,
 102
 tworzenie własnych reguł,
 104, 107
 warstwa kontroli dostępu, 38
 web crawlers, *Patrz* szperacze
 sieciowe
 web services, *Patrz* usługi
 sieciowe
 welcome(), 246
 wiązania modeli, 49
 modyfikowanie
 parametrów, 59
 modyfikowanie warunków, 63
 tworzenie wielu powiązań, 66

wiersz poleceń, parsowanie
 parametrów, 224
 wirtualne pola, 80, 82, 83
 write(), 151
 wyjątki, 319
 wyszukiwanie, 73, 74, 87
 stronicowanie, 93, 95
 własne typy, 89, 92, 93
 wywołania zwrotne, 109

X

Xdebug, 285
 XmlHelper, klasa, 192
 X-Path 2.0, 301, 302

Z

zachowania, 99
 dodawanie nowych pól, 116
 zadania, automatyzacja, 243
 zaślepki, 290, 292, 293
 częściowe, 293
 złączenia, 65, 66
 złączenia doraźne, 84

Ź

źródła danych, 127
 konsumowanie kanałów RSS,
 138
 parsowanie plików CSV, 134
 tworzenie, 142

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>

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.pl
księgarnia
internetowa

Nr katalogowy: 7710



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/newsy>

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

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-3542-9



Cena: 57,00 zł

Informatyka w najlepszym wydaniu