Opanuj niesamowite możliwości tego CMS-a
i twórz rozbudowane witryny
oraz aplikacje internetowe!

- Poznaj architekturę Drupala i sposób jego działania
- Naucz się pracować z bazą danych i API formularzy
- Twórz własne moduły i rozszerzaj możliwości tego systemu

Drupal 7

Zaawansowane programowanie

Wydanie III

Todd Tomlinson, John K. VanDyk

Helion

Apress

Drupal 7. Zaawansowane programowanie

Autorzy: Todd Tomlinson, John K. VanDyk
Tłumaczenie: Krzysztof Rychlicki-Kicior
ISBN: 978-83-246-3367-8
Tytuł oryginału: Pro Drupal 7 Development
Format: 168×237, stron: 528

Opanuj niesamowite możliwości tego CMS-a i twórz rozbudowane witryny oraz aplikacje internetowe!

- Poznaj architekturę Drupala i sposób jego działania
- Naucz się pracować z bazą danych i API formularzy
- Twórz własne moduły i rozszerzaj możliwości tego systemu
- Zobacz, jak optymalnie wykorzystać jQuery w Drupalu

Drupal to nie tylko kolejny, nieco lepszy od innych CMS. Ten system zarządzania treścią zrobił ostatnio oszałamiającą karierę jako doskonały framework do tworzenia wszelkiej maści aplikacji internetowych. Dzięki niezwykłej łatwości i szybkości, z jaką Drupal pozwala programistom tworzyć rozbudowane blogi, profesjonalne witryny korporacyjne czy serwisy społecznościowe, zainteresowanie tą technologią zaczęło dynamicznie wzrastać. Co więcej, choć już dziś wydaje się, że sposoby wykorzystania tego CMS-a ogranicza jedynie ludzka wyobraźnia, system jest wciąż intensywnie rozwijany przez liczną społeczność entuzjastów na całym świecie. Możliwość czerpania z ogromnych zasobów wiedzy tysięcy programistów poprawi jakość Twoich nawet najbardziej wyrafinowanych internetowych projektów.

Jeśli chcesz tworzyć nowoczesne, rozbudowane witryny internetowe z wykorzystaniem Drupala 7, właśnie znalazłeś idealny podręcznik dla siebie! Omówiono tu wszystko, co będzie Ci potrzebne, począwszy od podstawowych zagadnień, takich jak architektura i struktura plików systemu, przez sposoby wykorzystania API formularzy czy dodawania własnych modułów w celu rozszerzania możliwości, aż po metody tworzenia bezpiecznego, wydajnego kodu. Dowiesz się więcej na temat pracy z bazą danych, uruchomienia własnej strony wyszukiwarki oraz korzystania z jQuery w Drupalu. Nauczysz się także sprawnie optymalizować Drupal i przeprowadzać skuteczne testy oraz poznasz wiele innych praktycznych zagadnień, które sprawią, że bez trudu zrozumiesz zasady działania tego CMS-a oraz pokonasz wszelkie problemy napotykane w trakcie pracy.

Z tej książki dowiesz się między innymi:

- jak działa Drupal i jaką ma architekturę
- jak tworzyć dodatkowe, funkcjonalne moduły
- jak pracować z bazą danych i API formularzy
- jak wygląda obsługa użytkowników, segmentów i pól
- jak przechowywać dane w sesjach
- jak zapewnić wyszukiwanie i indeksowanie informacji
- jak wykorzystać jQuery w Drupalu
- jak tworzyć czysty, bezpieczny kod
- jak optymalizować działanie Drupala
- jak przeprowadzić efektywny proces testowania

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

Spis treści

ROZDZIAŁ 2

Tworzenie modułów

Moduły stanowią podstawowy materiał, budulec, na bazie którego funkcjonuje Drupal. Moduły pozwalają także na rozszerzanie możliwości standardowej wersji Drupala, nazywanej też rdzeniem Drupala. Osobom, które nie miały wcześniej kontaktu z Drupalem, tłumaczymy, że moduły Drupala można porównać do klocków Lego. Korzystając z precyzyjnie określonych reguł, można bez przeszkód łączyć różne moduły i uzyskiwać rozbudowane oraz funkcjonalne rozwiązania.

Istnieją dwie główne kategorie modułów Drupala — moduły rdzenia i dodatkowe. Moduły rdzenia są dostarczane wraz ze standardową instalacją systemu. W ich skład wchodzą ankiety, menu, kategorie, wyszukiwarka, subskrybent kanałów i fora. Moduły dodatkowe to wszystkie moduły tworzone przez społeczność Drupala. Rozszerzają one funkcjonalność oferowaną przez rdzeń Drupala. Na stronie *http://drupal.org/project/modules* znajdziesz tysiące modułów, które realizują przeróżne funkcje, począwszy od prostych modułów wyświetlających aktualną datę i czas, aż do zaawansowanych rozwiązań, takich jak sklep internetowy.

W tym rozdziale utworzymy nowy moduł od podstaw. W trakcie pracy poznasz standardy i reguły, jakie muszą spełniać tworzone przez Ciebie (i innych programistów) moduły. Warto utworzyć coś praktycznego, dlatego zajmiemy się potrzebnym zagadnieniem — przypisami. Podczas przeglądania stron typowej witryny Drupala może zdarzyć się, że będzie trzeba dodać krótki przypis — informację o danej stronie. Można by, co prawda, skorzystać z tradycyjnych komentarzy, jednak na ogół komentarze są dostępne dla każdego użytkownika odwiedzającego stronę, ewentualnie tylko dla zalogowanych użytkowników. Przypisy, według naszego założenia, mogłyby być widoczne tylko dla autora segmentu.

Tworzenie plików

Na początku musimy — rzecz jasna — wybrać nazwę dla naszego modułu. Skorzystamy z nazwy **`annotate`** („stworzyć przypis, adnotację") — krótkiej i treściwej. Następnie dla naszego modułu musimy znaleźć miejsce w strukturze katalogów. Moduły dodatkowe i własne są przechowywane w katalogu */sites/all/modules*. Każdy moduł jest przechowywany w podkatalogu mającym taką samą nazwę jak moduł.

Uwaga: Moduły rdzenia są przechowywane w katalogu */modules*. W ten sposób nie musisz obawiać się, że wszystkie dodawane przez Ciebie moduły zostaną nadpisane lub usunięte w trakcie wykonywania aktualizacji.

Jeśli chcesz, możesz utworzyć podkatalog */sites/all/modules/custom* do przechowywania wyłącznie modułów utworzonych przez Ciebie. Dzięki temu inni programiści bez problemu rozróżnią moduły utworzone przez Ciebie od modułów pobranych z witryny *Drupal.org*. Tak też uczynimy w tym przypadku — w podkatalogu */sites/all/modules/custom/annotate* znajdą się wszystkie pliki związane z modułem `annotate`.

Pierwszym plikiem, który utworzymy, będzie plik *annotate.info*. Każdy moduł Drupala 7. musi mieć plik składający się z nazwy modułu i rozszerzenia *.info*. W przypadku modułu `annotate` podstawowy zestaw informacji, który pozwoli Drupalowi prawidłowo rozpoznać nasz moduł, ma następującą postać:

```
name = Przypisy
description = "Moduł pozwala na tworzenie przypisów"
package = Drupal 7. Zaawansowane programowanie.
core = 7.x
files[] = annotate.module
files[] = annotate.install
files[] =  annotate.admin.inc
configure=admin/config/content/annotate/settings
```

Struktura tego pliku jest identyczna dla wszystkich modułów Drupala 7. Atrybut `name` określa nazwę modułu wyświetlaną na stronie konfiguracyjnej *Moduły*. Atrybut `description` zawiera opis modułu, wyświetlany w tym samym miejscu. Atrybut `package` definiuje pakiet (grupę), do której przynależy moduł. Moduły na stronie konfiguracyjnej *Moduły* są grupowane i wyświetlane według pakietów. Pole `core` określa wersję Drupala, do której został dostosowany moduł. Atrybut `php` definiuje wersję PHP wymaganą przez moduł. Wreszcie, atrybut `files` określa nazwy plików związanych z modułem. Dla modułu `annotate` są to pliki *annotate.module* i *annotate.install*.

Plik konfiguracyjny może zawierać także atrybuty opcjonalne. Oto przykład modułu, który wykorzystuje PHP w wersji 5.2, zaś do jego poprawnego działania Drupal musi mieć zainstalowane dwa moduły: forum i kategorie.

```
name = Chochlik na forum
description = Moduł losowo przypisuje odpowiedzi do różnych wątków
core = 7.x
dependencies[] = forum
dependencies[] = taxonomy
files[] = forumconfusion.module
files[] = forumconfusion.install
package = "Forum BonusPak Złego Boba"
php = 5.2
```

Teraz możemy zająć się tworzeniem kodu dla naszego modułu. Utwórz plik o nazwie *annotate.module* wewnątrz katalogu *sites/all/modules/custom/annotate*. Rozpocznij plik od otwierającego znacznika PHP i identyfikacyjnego znacznika CVS, po których następuje komentarz:

```
<?php

/**
 * @file
 * Moduł pozwala użytkownikom na dodawanie prywatnych przypisów do segmentów.
 *
 * Moduł dodaje pole tekstowe w momencie wyświetlenia segmentu.
 * Dzięki temu zalogowani użytkownicy mogą dodawać przypisy.
 */
```

Na początku zwróć uwagę na styl komentarza. Na początku znajduje się ciąg /**, po czym każdy kolejny wiersz rozpoczyna się od gwiazdki poprzedzonej spacją (*). Wreszcie, na końcu komentarza znajduje się sekwencja */. Atrybut `@file` informuje, że następny akapit zawiera opis pliku. Ten jednowierszowy opis jest wykorzystywany również przez `api.module` (*http://drupal.org/project/api*) — moduł odpowiedzialny za tworzenie i formatowanie dokumentacji na podstawie kodu. Przy okazji, zapamiętaj adres *http://api.drupal.org*. Na tej stronie znajdziesz dokumentację wszystkich API wchodzących w skład Drupala. Warto, abyś zapoznał się z tą stroną. Jest to nieocenione źródło informacji dla programistów, którzy tworzą bądź modyfikują moduły.

Tuż za pustym wierszem znajduje się dłuższy opis, przeznaczony dla programistów, którzy będą korzystać (i zapewne poprawiać) nasz kod. Zwróć uwagę, że nie używamy znacznika zamykającego (?>). Element ten jest w PHP opcjonalny. Jego dołączenie mogłoby spowodować problemy z białymi znakami na końcu pliku (*http://drupal.org/coding-standards#phptags*).

■ **Uwaga:** Zastanawiasz się, czemu przykładamy taką wagę do sposobu organizacji plików, katalogów i wszelkich innych elementów w obrębie instalacji Drupala? Odpowiedź jest prosta. Gdy setki programistów z całego świata pracują nad jednym projektem, zachowanie jednolitych standardów znacznie upraszcza i skraca czas pracy związany z łączeniem efektu prac różnych ludzi w jedną całość. Szczegółowe konwencje programistyczne obowiązujące w Drupalu znajdziesz w sekcji *Coding standards* w dokumencie *Developing for Drupal Handbook* (*http://drupal.org/coding-standards*).

Implementacja haka

Drupal intensywnie wykorzystuje mechanizm haków, zwanych też niekiedy wywołaniami zwrotnymi. W trakcie wykonywania swoich zadań Drupal daje modułom możliwość zareagowania na pewne zdarzenia. Przykładowo tuż po załadowaniu segmentu z bazy danych (a jeszcze przed wyświetleniem go na stronie) Drupal sprawdza, czy włączone moduły mają zaimplementowaną funkcję `hook_node_load()`. Następnie funkcje znalezione w modułach są wywoływane jeszcze przed wyświetleniem modułu na stronie. Przykład takiego zachowania sprawdzimy, korzystając z naszego modułu `annotate`.

Pierwszy hak, który będziemy implementować, jest zdefiniowany za pomocą haka `hook_menu()`. Za pomocą tej funkcji będziemy mogli dodać dwa elementy do menu administracyjnego naszej strony. W ten sposób do głównego menu konfiguracyjnego dodamy opcję *Przypisy*, a także podmenu (wewnątrz menu *Przypisy*) o nazwie *Ustawienia*. Kliknięcie tej pozycji spowoduje uruchomienie strony konfiguracyjnej naszego modułu. Elementy menu definiujemy przy użyciu tablic, składających się z kluczy i wartości określających zachowanie Drupala w momencie nadejścia żądania o określonej ścieżce. Tym zagadnieniem zajmiemy się szczegółowo w rozdziale 4., poświęconym menu i wywołaniom zwrotnym Drupala. Odwołanie do haka `hook_menu` nazwiemy `annotate_menu` — zastępując słowo `hook` nazwą tworzonego modułu. Ta zasada obowiązuje dla wszystkich haków — zawsze należy zastępować słowo hak nazwą konkretnego modułu.

Oto kod, który musimy dodać do naszego modułu:

```
/**
 * Implementacja haka hook_menu().
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Przypis segmentu',
    'description' => 'Dostosuj opcje przypisów segmentu.',
    'position' => 'right',
    'weight' => -5,
    'page callback' => 'system_admin_menu_block_page',
    'access arguments' => array('administer site configuration'),
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
  );
  $items['admin/config/annotate/settings'] = array(
    'title' => 'Ustawienia przypisów',
    'description' => 'Dostosuj sposób działania przypisów.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_settings'),
    'access arguments' => array('administer site configuration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
  );
  return $items;
}
```

Nie musisz analizować treści powyższego kodu w tej chwili. Jego działanie jest następujące: gdy użytkownik wejdzie pod adres *http://example.com/?q=admin/config/annotate/settings*, zostanie wywołana funkcja

`drupal_get_form()`, która otrzyma jako argument identyfikator formularza `annotate_admin_settings`. W związku z tym, nastąpi próba znalezienia funkcji opisującej ten formularz w pliku *annotate.admin.inc*. Element menu (a co za tym idzie, możliwość wykonania operacji) otrzymają jedynie użytkownicy z uprawnieniem *administer site configuration*. W momencie wyświetlenia formularza Drupal poprosi o określenie definicji formularza. Po zakończeniu odpytywania wszystkich modułów w celu uzyskania ich elementów menu następuje próba wybrania funkcji odpowiadającej ścieżce, z której nastąpiło żądanie.

■ **Uwaga:** Jeśli chcesz poznać bliżej mechanizm sterujący wszystkimi hakami, zajrzyj na stronę z dokumentacji funkcji `module_invoke_all()`, zadeklarowanej w pliku *includes/module.inc* (*http://api.drupal.org/api/function/module_invoke_all/7*).

Teraz powinno być już jasne, dlaczego funkcja haka nosi nazwę `hook_menu()`.

■ **Wskazówka:** Haki Drupala pozwalają na obsługę przeróżnych zdarzeń systemu. Pełna lista obsługiwanych haków znajduje się w dokumentacji API Drupala (*http://api.drupal.org/api/group/hooks/7*).

Dodawanie ustawień charakterystycznych dla modułu

Drupal zawiera wiele różnych rodzajów segmentów (nazywanych też rodzajami zawartości w interfejsie użytkownika), takich jak artykuły czy zwykłe strony. W związku z tym, chcielibyśmy, aby przypisy mogły być stosowane tylko do niektórych rodzajów segmentów. W tym celu musimy utworzyć stronę z możliwością wyboru rodzajów zawartości, przy których będzie pojawiać się opcja wstawienia przypisu. Strona będzie mieć prostą budowę — obok nazwy każdego dostępnego rodzaju zawartości znajdzie się pole typu checkbox. Zaznaczenie bądź usunięcie znaczenia tego pola będzie stanowiło o możliwości wstawiania przypisów przy danym rodzaju zawartości (rysunek 2.1). Opisana strona powinna być dostępna jedynie dla administratora, dlatego jej kod powinien być ładowany i przetwarzany tylko wtedy, kiedy będzie trzeba. Z tego względu kod strony umieścimy w osobnym pliku (a nie w dotychczas edytowanym pliku `annotate.module`, który będzie ładowany i wykorzystywany przy każdym żądaniu). W poprzednim listingu poinformowaliśmy Drupal, że powinien szukać formularza ustawień w pliku *annotate.admin.inc*, dlatego musimy dotrzymać słowa i utworzyć plik *sites/all/modules/annotate/annotate.admin.inc* o poniższej treści:

```
<?php

/**
 * @file
 * Strona administracyjna z wywołaniami zwrotnymi modułu annotate.
 */

/**
 * Tworzenie formularza dla ustawień przypisów.
 *
 * @ingroup forms
 * @see system_settings_form().
 */
function annotate_admin_settings() {
  // Pobierz tablicę rodzajów segmentów, zawierającą wewnętrzne nazwy jako klucze i przyjazne nazwy jako wartości,
  // np. array('page' => 'Prosta strona', 'article' => 'Artykuły')

  $types = node_type_get_types();
  foreach($types as $node_type) {
```

```
    $options[$node_type->type] = $node_type->name;
  }
$form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Użytkownicy mogą tworzyć przypisy dla następujących rodzajów zawartości'),
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('Przy wybranych rodzajach zawartości zostaną wyświetlone pola tekstowe
    ↪w celu umożliwienia dodawania prywatnych uwag przez użytkowników.'),
  );

  $form['#submit'][] = 'annotate_admin_settings_submit';
  return system_settings_form($form);

}
```

Formularze w Drupalu są reprezentowane za pomocą struktury drzewiastej, tzn. tablicy tablic (tablic zagnieżdżonych). Przy użyciu tej struktury danych Drupal przekazuje niezbędne informacje do silnika renderującego (odpowiedzialnego za wygenerowanie kodu HTML). Dla zwiększenia czytelności każdy element tablicy został umieszczony w osobnym wierszu. Każdy atrybut formularza został oznaczony znakiem krzyżyka (#). Atrybuty te pełnią dodatkowo rolę kluczy w tablicy. Na początku określamy rodzaj elementów formularza za pomocą wartości checkboxes. Oznacza ona, że pola tego typu będą tworzone z wykorzystaniem tablicy z kluczami, do której odwołujemy się za pomocą zmiennej `$options`.

Opcje zostaną pobrane z zastosowaniem funkcji `node_type_get_types()`, która zwraca tablicę obiektów. Efekt działania tej funkcji mógłby wyglądać tak:

```
[article] => stdClass Object (
  [type] => article
  [name] => Artykuł
  [base] => node_content
  [description] => Artykuły są wykorzystywane do tworzenia treści o istotnej dacie publikacji,
  ↪np. aktualności, wiadomości prasowych czy też wpisów na blogu.
  [help] =>
  [has_title] => 1
  [title_label] => Tytuł
  [has_body] => 1
  [body_label] => Treść
  [custom] => 1
  [modified] => 1
  [locked] => 0
  [orig_type] => article
)
```

Kluczem powyższego obiektu (`[article] =>`) jest identyfikator rodzaju segmentu, używany wewnętrznie przez Drupala do przetwarzania segmentów, podczas gdy przyjazna nazwa to wartość atrybutu `name`, znajdująca się wewnątrz definicji obiektu.

API formularzy Drupala zobowiązuje nas do umieszcza w atrybucie `#options` par *`klucz => wartość`*. Dlatego właśnie w pętli `foreach` kluczem pary staje się atrybut `type`, a wartością pary — atrybut `name`. Korzystając z tablicy `$options`, Drupal wygeneruje w formularzu pola typu checkbox dla prostej strony, artykułu oraz wszystkich innych rodzajów zawartości, jakie są dostępne w Twojej aplikacji.

Tytuł formularza ustawiamy na podstawie wartości atrybutu `#title`.

■ **Uwaga:** Wszystkie teksty wyświetlane na stronie (takie jak wartości pól `#title` i `#description` naszego formularza) są umieszczane jako argumenty wywołania funkcji `t()`. Funkcja ta znacznie ułatwia tłumaczenie tekstów. Przetworzenie wszystkich tekstów w aplikacji przez funkcję tłumaczącą `t` znacznie ułatwi utworzenie kolejnych wersji językowych. Nie zastosowaliśmy tej funkcji dla elementów menu, ponieważ są one tłumaczone automatycznie.

Kolejny z atrybutów, `#default_value`, określa wartość domyślną elementu formularza. Skoro wartości typu checkboxes mogą zawierać wiele elementów typu checkbox, wartość domyślna dla takiego typu musi być tablicą. Wyrażenie zdefiniowane dla atrybutu `#default_value` jest warte dłuższej analizy:
`variable_get('annotate_node_types', array('page'))`

Drupal pozwala na przechowywanie i odczytywanie dowolnych wartości za pomocą dwóch specjalnych funkcji: `variable_get()` i `variable_set()`. Wartości są zapisywane w tabeli `variables` w bazie danych i dostępne przez cały czas przetwarzania żądania. Oczywiście, nie należy nadużywać tych funkcji zarówno pod względem liczby przechowywanych obiektów, jak i ich wielkości. Wartości są ładowane przy okazji każdego żądania, dlatego takie nadużycia mogłyby spowolnić pracę całego systemu. Mechanizm ten nadaje się natomiast znakomicie do przechowywania wszelkich ustawień. Zwróć uwagę, że do funkcji `variable_get` przekazujemy klucz, który jednoznacznie identyfikuje szukaną wartość (dzięki czemu Drupal może znaleźć ją w tabeli `variables`), a także wartość domyślną. Powyższy zapis możemy interpretować następująco: jeśli w tabeli `variables` nie ma określonych rodzajów segmentów, dla których są włączone przypisy, skorzystaj z tablicy określonej w wartości domyślnej. Domyślnie pozwalamy na dodawanie przypisów jedynie dla prostych (ang. *Basic*) stron.

■ **Wskazówka:** Korzystając z funkcji `system_settings_form()`, pamiętaj, że nazwa elementu formularza (np. `annotate_node_types`) musi być taka sama jak klucz określony w funkcji `variable_get()`.

Opis pełni jedynie funkcję informacyjną — dla administratora strony jest to informacja, do czego służy dane pole. Formularze omówimy szczegółowo w rozdziale 11.

Następnym krokiem w tworzeniu naszego modułu będzie dodawanie i usuwanie pola przypisu do rodzajów zawartości. Jeśli administrator strony włączy dany rodzaj zawartości, pole przypisu zostanie do niego dodane. Usunięcie zaznaczenia pola będzie równoznaczne z usunięciem pola przypisu z danego rodzaju zawartości. Field API Drupala (API związane z polami) pomoże zdefiniować pole i powiązać je z typem zawartości. Field API wykonuje wszystkie operacje związane z konfigurowaniem pola, włącznie z tworzeniem w bazie danych tabeli, w której będą przechowywane treści przypisów, i tworzeniem elementów formularza niezbędnych do pobrania treści przypisu od autora. Field API pozwoli także na powiązanie pola z rodzajem segmentu i dołączenie go do formularza edycji danego segmentu, a także uwzględnienie pola przy wyświetlaniu tegoż segmentu. Field API szczegółowo opisujemy w rozdziale 8.

Najpierw zajmiemy się obsługą wysyłanych formularzy, a zatem napiszemy kod obsługujący wysłanie formularza przez administratora. Chodzi — oczywiście — o formatkę wyboru rodzajów segmentów, które mają mieć włączone dodawanie przypisów. Jeśli pole checkbox danego segmentu nie jest zaznaczone, musimy upewnić się, że segment ten nie ma powiązanego pola przypisu — a jeśli tak jest, pole to należy usunąć wraz z istniejącymi przypisami. Włączenie pola typu checkbox spowoduje sprawdzenie, czy pole przypisu dla danego rodzaju zawartości istnieje. Jeśli nie, zostanie dodane (w przeciwnym przypadku nie trzeba podejmować żadnej akcji).

```
/**
* Przetwarzanie formularza ustawień przypisów.
*/
function annotate_admin_settings_submit($form, $form_state) {
  // Przeiteruj kolekcję checkboksów zawartych w formularzu
  foreach ($form_state['values']['annotate_node_types'] as $key => $value) {
    // Jeśli checkbox dla danego rodzaju zawartości nie jest zaznaczony, sprawdź, czy dany rodzaj zawartości ma zdefiniowane
    // pole przypisu, korzystając z funkcji field_info_instance. Jeśli tak, usuń pole przypisu, ponieważ administrator usunął
    // zaznaczenie danego pola formularza.
    if (!$value) {
      $instance = field_info_instance('node', 'annotation', $key);
      if (!empty($instance)) {
        field_delete_instance($instance);
        watchdog("Przypis", 'Usunięto pole przypisu z następującego rodzaju zawartości:
        %key', array('%key' => $key));
      }
    } else {
```

```
      // Jeśli checkbox dla danego rodzaju zawartości jest zaznaczony, sprawdź, czy pole przypisu dla danego rodzaju
      // zawartości istnieje. Jeśli nie, dodaj pole przypisu do tego rodzaju zawartości.
      $instance = field_info_instance('node', 'annotation', $key);
      if (empty($instance)) {
        $instance = array(
          'field_name' => 'annotation',
          'entity_type' => 'node',
          'bundle' => $key,
          'label' => t('Przypis'),
          'widget_type' => 'text_textarea_with_summary',
          'settings' => array('display_summary' => TRUE),
          'display' => array(
            'default' => array(
              'type' => 'text_default',
            ),
            'teaser' => array(
              'type' => 'text_summary_or_trimmed',
            ),
          ),
        );
        $instance = field_create_instance($instance);
        watchdog('Przypis', 'Dodano pole przypisu do następującego rodzaju zawartości: %key',
          array('%key' => $key));
        }
      }
    } // Koniec pętli foreach.
}
```

Po zbudowaniu skryptu obsługi formularza możemy zająć się utworzeniem pliku *.install* dla naszego modułu. Plik instalacyjny zawiera jedną lub więcej funkcji, które są wywoływane w momencie instalacji lub deinstalacji modułu. W naszym przypadku w trakcie instalacji chcemy utworzyć pole przypisu, które mogłoby być przypisane do rodzajów zawartości przez administratora. Deinstalacja modułu wiązać się będzie z usunięciem powiązań pola przypisu ze wszystkimi rodzajami zawartości, a także usunięciem samego pola z bazy danych Drupala. W tym celu musimy utworzyć nowy plik o nazwie *annotate.install* w katalogu modułu `annotate`.

Pierwszą funkcją, jaką wywołamy, będzie funkcja obsługi haka `hook_install()`. Zgodnie z konwencją nazewniczą Drupala, funkcja otrzyma nazwę `annotate_install()`. W funkcji tej za pomocą Field API sprawdzimy, czy pole istnieje, a jeśli nie, nie pozostanie nam nic innego, jak je utworzyć:

```
<?php

/**
 * Implementacja haka hook_install()
 */

function annotate_install() {

  // Sprawdź, czy pole istnieje.
  $field = field_info_field('annotation');

  // Jeśli nie, utwórz pole.
  if (empty($field)) {
    $field = array(
      'field_name' => 'annotation',
      'type' => 'text_with_summary',
      'entity_types' => array('node'),
      'translatable' => TRUE,
    );
```

```
    $field = field_create_field($field);
  }

}
```

Teraz możemy przejść do tworzenia funkcji deinstalacyjnej, korzystając z haka `hook_uninstall`. Stosowna funkcja otrzyma nazwę `annotate_uninstall`. Skorzystamy w niej z funkcji `watchdog` w celu zapisania w dzienniku (logu) informacji o deinstalacji modułu. Następnie skorzystamy z funkcji `node_get_types()`, aby pobrać wszystkie dostępne rodzaje zawartości. Dzięki temu sprawdzimy, które z rodzajów zawartości mają swoje pola przypisu i usuniemy je. Na koniec będziemy mogli usunąć definicję pola przypisu.

```
/**
 * Kod obsługi haka hook_uninstall()
 */
function annotate_uninstall() {

   watchdog("Moduł Przypisy", "Deinstalacja modułu i usuwanie pól");

   $types = node_type_get_types();
   foreach($types as $type) {
      annotate_delete_annotation($type);
   }
   $field = field_info_field('annotation');

   if ($field) {
      field_delete_field('annotation');
   }

}

function annotate_delete_annotation($type) {

  $instance = field_info_instance('node', 'annotation', $type->type);

  if ($instance) {
     field_delete_instance($instance);
  }

}
```

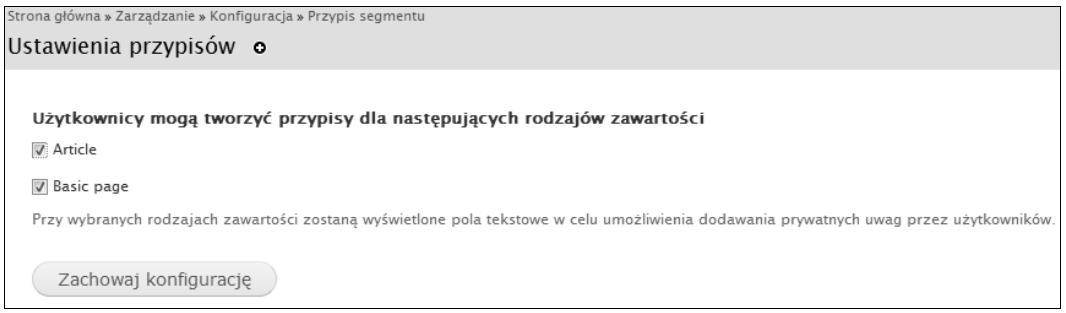
Zmierzając powoli do końca tworzenia naszego modułu, możemy zająć się ostatnią czynnością — dodaniem kodu do pliku *.module* w celu sprawdzenia, czy osoba przeglądająca dany segment jest jego autorem. Jeśli ten warunek nie jest spełniony, przypis musi zostać ukryty. Do realizacji tej funkcjonalności posłuży hak `hook_node_load()`, wywoływany w trakcie ładowania segmentu. Wewnątrz funkcji obsługi tego haka sprawdzimy, czy osoba przeglądająca segment jest jego autorem. Jeśli tak nie jest, przypis zostanie ukryty przez zastosowanie operacji `unset`:

```
/**
 * Kod obsługi haka hook_node_load()
 */
function annotate_node_load($nodes, $types) {

  global $user;

  // Sprawdź, czy osoba przeglądająca segment jest jego autorem. Jeśli nie, ukryj przypis.
  foreach ($nodes as $node) {
     if ($user->uid != $node->uid) {
```

```
        unset($node->annotation);
      }
    }
}
```

Zapisz utworzone pliki (*.info*, *.install*, *.admin.inc*, *.module*), a następnie kliknij łącze *Moduły* w menu administratora na górze strony. Twój moduł powinien znaleźć się w grupie zatytułowanej *Drupal 7. Zaawansowane programowanie.* (Jeśli tak nie jest, sprawdź dokładnie pliki *annotate.info* i *annotate.module* pod kątem składni i upewnij się, że oba pliki znajdują się w katalogu *sites/all/modules/custom*). Po całej pracy wykonanej w tym rozdziale możesz wreszcie włączyć moduł.

Po włączeniu modułu na stronie *admin/config/annotate/settings* powinien znaleźć się formularz konfiguracji dla modułu `annotate.module` (rysunek 2.1).



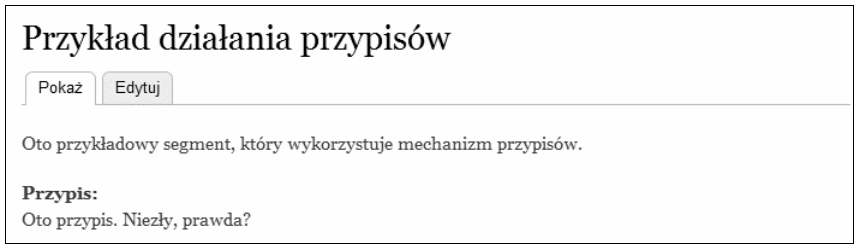
Rysunek 2.1. *Formularz konfiguracji dla modułu annotate.module*

Zaledwie kilka wierszy kodu dało w pełni funkcjonalny formularz konfiguracji naszego modułu, który automatycznie zapisze i zapamięta ustawienia. Jest to dobitny przykład możliwości Drupala.

Sprawdźmy teraz główną funkcjonalność naszego modułu. Najpierw musimy włączyć przypisy dla wszystkich rodzajów zawartości. Zaznacz wszystkie pola na stronie konfiguracyjnej i kliknij przycisk *Zachowaj konfigurację*. Następnie utwórz prostą stronę i przewiń ją na sam dół, aż zobaczysz pole *Przypis* (rysunek 2.2).

Rysunek 2.2. *Gotowy formularz przypisu*

Utwórz nowy segment, wprowadzając tytuł, treść i przypis. Po zakończeniu zachowaj zmiany, a powinieneś zobaczyć efekt, taki jak na rysunku 2.3.



Rysunek 2.3. *Segment zawierający przypis*

Możesz zastanawiać się, gdzie Drupal przechowuje wszystkie informacje związane z przypisami — w końcu nie wykonywaliśmy jawnie żadnych operacji związanych z bazą danych. Field API wykonuje wszystkie niezbędne operacje, począwszy od utworzenia tabeli, przez zapis i odczyt danych, aż do usunięcia tabeli w przypadku deinstalacji. Wywołanie funkcji `field_create_field()` powoduje utworzenie tabeli o nazwie `field_data_<nazwa_pola>`. W naszym przypadku nazwą tabeli będzie ciąg `field_data_annotations`. Szczegółowe informacje na temat Field API znajdziesz w rozdziale 4.

Tworzenie własnej sekcji administracyjnej

Drupal zawiera kilka kategorii ustawień administracyjnych — takich jak zarządzanie treścią czy zarządzanie użytkownikami — które pojawiają się na stronie *Konfiguracja*. Może się zdarzyć, że Twojemu modułowi przydałaby się własna kategoria. Nic prostszego — w tym podrozdziale wykreujemy nową kategorię ustawień o nazwie *Przypisy segmentów*. Na początek musimy utworzyć funkcję obsługi haka `hook_menu`.

```
/**
 * Obsługa haka hook_menu().
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Przypis segmentów',
    'description' => 'Dostosuj opcje przypisów segmentów.',
    'position' => 'right',
    'weight' => -5,
    'page callback' => 'system_admin_menu_block_page',
    'access arguments' => array('administer site configuration'),
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
  );
  $items['admin/config/annotate/settings'] = array(
    'title' => 'Ustawienia przypisów',
    'description' => 'Zmień zachowanie przypisów.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_settings'),
    'access arguments' => array('administer site configuration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
  );

  return $items;
}
```

Na rysunku 2.4 przedstawiamy kategorię na stronie *Konfiguracja*, zawierającą łącze do ustawień naszego modułu.

Rysunek 2.4. *Łącze do ustawień modułu przypisów pojawia się jako odrębna kategoria*

Jeśli kiedykolwiek zdarzy Ci się zmienić kod obsługi haka menu, musisz wyczyścić pamięć podręczną (ang. *cache*) menu. Można to zrobić na dwa sposoby — czyszcząc tabelę *cache_menu* lub wybierając opcję *Utwórz ponownie menu* dostępną w module programistycznym Drupala (ang. *devel.module*). Możesz także skorzystać z opcji *Wyczyść dane pamięci podręcznej*, która znajduje się na stronie *Konfiguracja/Wydajność*.

■ **Wskazówka:** Moduł devel (*http://drupal.org/project/devel*) powstał specjalnie dla programistów rozszerzających możliwości Drupala. Przy jego użyciu można łatwo uzyskać dostęp do takich opcji jak czyszczenie pamięci podręcznej, podgląd zmiennych czy śledzenie zapytań do bazy danych. To obowiązkowe narzędzie każdego programisty Drupala.

Dodanie nowej kategorii wymaga wykonania dwóch kroków. Po pierwsze, dodać trzeba element menu, który opisuje nagłówek kategorii. Element menu ma swoją unikalną ścieżkę (*admin/config/annotate*). Zostanie on umiejscowiony w prawej kolumnie, z wagą o wartości 5, dzięki czemu znajdzie się tuż przed kategorią *Usługi web*, co stanowi najwygodniejsze rozwiązanie.

Drugi krok sprowadza się do zagnieżdżenia hiperłącza prowadzącego do ustawień przypisów wewnątrz kategorii Przypisy segmentów. Uda się to osiągnąć po ustawieniu ścieżki oryginalnego menu na wartość *admin/config/annotate/settings*. W momencie ponownego tworzenia menu Drupal analizuje ścieżki w celu określenia związku między nadrzędnymi i podrzędnymi elementami. Efektem będzie wyświetlenie elementu o ścieżce *admin/config/annotate/settings* jako dziecka elementu o ścieżce *admin/config/annotate*.

Drupal ładuje jedynie pliki niezbędne do zrealizowania żądania. Takie podejście sprawia, że udaje się oszczędzić sporo pamięci. Zwróć uwagę, że atrybut `page callback` (wywołanie zwrotne) elementu tablicy `items` odwołuje się do funkcji spoza naszego modułu (`system_admin_menu_block_page()` z modułu `system.module`). W związku z tym, musimy poinformować Drupal o konieczności załadowania pliku *modules/system/system.admin.inc* (domyślnie Drupal próbowałby dołączyć plik *sites/all/modules/custom/annotate/system.admin.inc*). Możemy to osiągnąć, pobierając ścieżkę do modułu *system* i umieszczając ją jako wartość dla atrybutu `file` naszego elementu w tablicy `items`.

Oczywiście, powyższy przykład tworzenia kategorii został zrealizowany głównie w celach dydaktycznych — w praktyce tworzenie kategorii powinno być dobrze uzasadnione — nie ma sensu irytować administratora nadmierną liczbą kategorii.

Wyświetlanie formularza ustawień

W module przypisów administrator może wybrać rodzaje zawartości (segmenty), które mogą być oznaczane przypisami (rysunek 2.1). Przyjrzyjmy się bliżej temu zagadnieniu.

Zmiana ustawień modułu przypisów wiąże się z koniecznością wyświetlenia formularza. Nasz element menu wywoła zwrotnie funkcję `drupal_get_form()`, przekazując jako parametr tablicę `annotate_admin_settings`. Oznacza to, że w momencie nadejścia żądania *http://example.com/?q=admin/config/annotate/settings* zostanie wywołana funkcja `drupal_get_form('annotate_admin_settings')`, co w praktyce spowoduje utworzenie formularza określonego za pomocą funkcji `annotate_admin_settings()`.

Przeanalizujmy funkcję definiującą formularz, która dodaje pole typu checkbox do rodzajów zawartości (rysunek 2.1), a także dwie nowe opcje. Funkcja znajduje się w pliku *sites/all/modules/custom/annotate/annotate.admin.inc*.

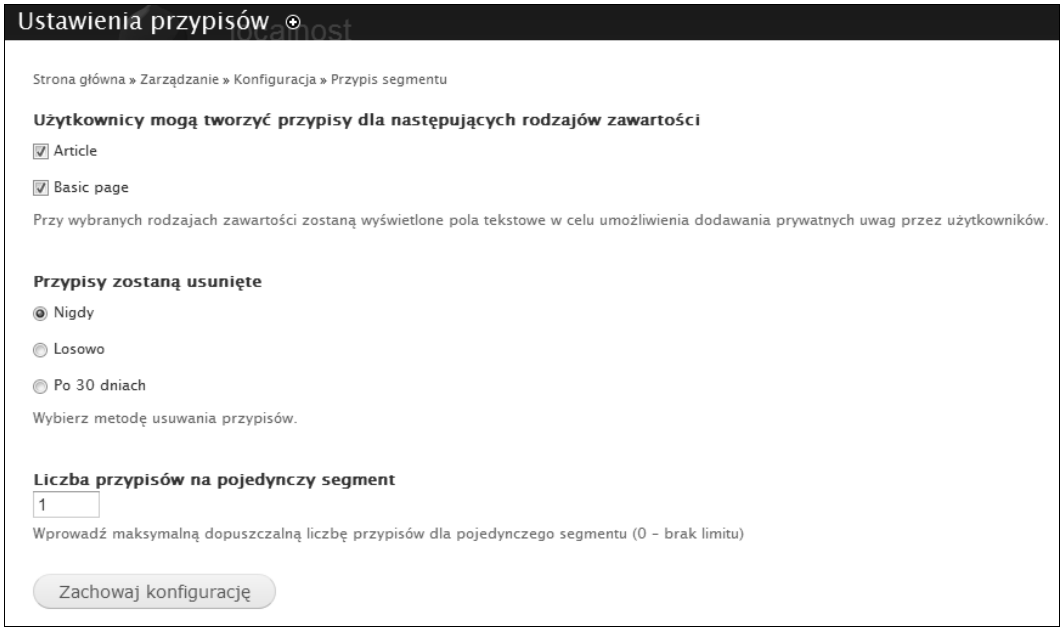
```
/**
 * Tworzenie formularza dla ustawień przypisów.
 *
```

```
 * @ingroup forms
 * @see system_settings_form().
 */
function annotate_admin_settings() {
  // Pobierz tablicę rodzajów segmentów, zawierającą wewnętrzne nazwy jako klucze i przyjazne nazwy jako wartości,
  // np. array('page' => 'Prosta strona', 'article' => 'Artykuły')

  $types = node_type_get_types();
  foreach($types as $node_type) {
     $options[$node_type->type] = $node_type->name;
  }
$form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Użytkownicy mogą tworzyć przypisy dla następujących rodzajów zawartości'),
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('Przy wybranych rodzajach zawartości zostaną wyświetlone pola tekstowe
    ↪w celu umożliwienia dodawania prywatnych uwag przez użytkowników.'),
  );

  $form['annotate_deletion'] = array(
    '#type' => 'radios',
    '#title' => t('Przypisy zostaną usunięte'),
    '#description' => t('Wybierz metodę usuwania przypisów.'),
    '#options' => array(
      t('Nigdy'),
      t('Losowo'),
      t('Po 30 dniach')
    ),
    '#default_value' => variable_get('annotate_deletion', 0) // Domyślnie: Nigdy
  );
  $form['annotate_limit_per_node'] = array(
    '#type' => 'textfield',
    '#title' => t('Liczba przypisów na pojedynczy segment'),
    '#description' => t('Wprowadź maksymalną dopuszczalną liczbę przypisów dla pojedynczego segmentu
    ↪(0 – brak limitu)'),
    '#default_value' => variable_get('annotate_limit_per_node', 1),
    '#size' => 3
  );

  $form['#submit'][] = 'annotate_admin_settings_submit';
  return system_settings_form($form);
}
```

Pole `radio` (pole jednokrotnego wyboru) posłuży nam do wyboru metody usuwania przypisów, a z kolei zwykłe pole tekstowe pozwoli na ograniczenie maksymalnej liczby przypisów dla pojedynczego segmentu (implementację tych funkcji pozostawiamy jako ćwiczenie). Zamiast samodzielnie przetwarzać nasz formularz, korzystamy z funkcji `system_settings_form()`, która doda niezbędne przyciski, przeprowadzi walidację i obsłuży wysłany formularz. Na rysunku 2.5 przedstawiamy aktualny widok formularza.

Walidacja ustawień wysłanych przez użytkownika

Skoro funkcja `system_settings_form()` zajmuje się przechowaniem wartości formularza automatycznie, skąd mamy wiedzieć, że wartość wprowadzona w pole *Liczba przypisów* na pojedynczy segment jest istotnie liczbą? Musimy dodać stosowny kod, tworząc funkcję walidacji (`annotate_admin_settings_validate($form, $form_state)`) w pliku *sites/all/modules/custom/annotate/annotate.admin.inc*. W razie jakichkolwiek problemów z przesłaną wartością funkcja włączy komunikat o błędzie:



Rysunek 2.5. *Rozszerzony formularz ustawień, zawierający pole typu checkbox, pole typu radio i pole tekstowe*

```
/**
 * Walidacja formularza ustawień przypisów.
 */
function annotate_admin_settings_validate($form, &$form_state) {
  $limit = $form_state['values']['annotate_limit_per_node'];
  if (!is_numeric($limit)) {
    form_set_error('annotate_limit_per_node', t('Proszę wprowadzić liczbę.'));
  }
}
```

Powyższa funkcja zostanie wywołana w momencie przetwarzania formularza przez Drupal. Po wystąpieniu problemów użytkownik otrzyma stosowny komunikat o błędzie. Dodatkowo pole zawierające niepoprawną zawartość zostanie odpowiednio podświetlone.

Skąd Drupal „wie", że ma wywołać właśnie tę funkcję? Skorzystaliśmy ze specjalnej nazwy, składającej się z nazwy funkcji definiującej formularz (`annotate_admin_settings`) z dołączonym członem `_validate`. Więcej informacji na temat funkcji walidacji znajdziesz w rozdziale 11.

Przechowywanie ustawień

W poprzednim przykładzie dwie operacje, czyli wprowadzenie nowych ustawień i kliknięcie przycisku *Zachowaj konfigurację*, zadziałały bez problemu. Teraz zastanowimy się nad tym, co właściwie stało się po wykonaniu powyższych operacji.

Tabela variables w akcji

Najpierw musimy przyjrzeć się polu *Liczba przypisów na pojedynczy segment*. Wartość domyślna tego pola (`#default_value`) jest uzyskiwana za pomocą wywołania `variable_get('annotate_limit_per_node', 1)`.

Drupal korzysta z tabeli `variables` znajdującej się w bazie danych. Pary klucz-wartość mogą być do niej zapisywane za pomocą metody `variable_set($klucz, $wartosc)`, a następnie odczytywane przy użyciu wywołania `variable_get($klucz, $wartosc_domyslna)`. Powyższe wywołanie metody `variable_get()` dla przypisów możemy więc zinterpretować następująco: ustaw wartość domyślną pola *Liczba przypisów na pojedynczy segment* na wartość przechowywaną w tabeli `variables` dla klucza `annotate_limit_per_node`, a jeśli takiej wartości nie ma, skorzystaj z wartości 1.

■ **Ostrzeżenie:** Przechowywanie wartości w tabeli `variables` wiąże się z możliwością powstania konfliktów nazw. Z tego względu zawsze nadawaj elementom formularza i kluczom tej tabeli takie same nazwy (np. `annotate_limit_per_node`). Nazwy najlepiej tworzyć według pewnego schematu, np. nazwa modułu + czytelny opis zmiennej. Tak utworzone nazwy stosuj zarówno dla elementów formularzy, jak i kluczy wartości z tabeli `variables`.

Pole *Przypisy zostaną usunięte* jest nieco bardziej skomplikowane, ponieważ jest to pole typu radio. Obiekt `#options` dla tego pola ma następującą postać:

```
'#options' => array(
  t('Nigdy'),
  t('Losowo'),
  t('Po 30 dniach')
)
```

Gdy PHP dostaje tablicę bez kluczy, kolejne wartości otrzymują automatycznie klucze liczbowe, dlatego w rzeczywistości powyższa tablica wygląda następująco:

```
'#options' => array(
  [0] => t('Nigdy'),
  [1] => t('Losowo'),
  [2] => t('Po 30 dniach')
)
```

Ustawiając wartość domyślną, korzystamy z poniższej konstrukcji. Domyślna wartość jest określona kluczem 0, co w praktyce przekłada się na użycie wartości `t('Nigdy')`.

```
'#default_value' => variable_get('annotate_deletion', 0) // Domyślnie: Nigdy
```

Wczytywanie wartości za pomocą funkcji variable_get()

Do odczytania zapisanych wcześniej ustawień modułu należy posłużyć się funkcją `variable_get()`:

```
// Pobierz maksymalną dozwoloną liczbę przypisów na pojedynczy segment.
$max = variable_get('annotate_limit_per_node', 1);
```

Warto zwrócić uwagę na podanie wartości domyślnej w wywołaniu funkcji `variable_get()`. Zostanie ona wykorzystana np. w sytuacji, gdy administrator ani razu nie odwiedził strony z ustawieniami.

Co dalej?

Oczywistym krokiem będzie udostępnienie naszego modułu społeczności Drupala, dlatego musimy utworzyć plik *README.txt* — w tym samym katalogu, co pliki *annotate.info*, *annotate.module*, *annotate.admin.inc* i *annotate.install*. Plik *README.txt* zawiera informacje o autorze i instalacji modułu. Informacje na temat licencji nie są konieczne, ponieważ wszystkie moduły wysyłane na stronę *drupal.org* podlegają licencji GPL. Ponadto odpowiedni skrypt na stronie *drupal.org* automatycznie dołączy plik *LICENSE.txt*. Po zakończeniu przygotowania modułu możesz wysłać go do repozytorium kontrybucji na stronie *drupal.org*, a następnie utworzyć stronę projektu, aby mieć kontakt z innymi członkami społeczności.

Podsumowanie

Po przeczytaniu tego rozdziału powinieneś:

- poradzić sobie z utworzeniem modułu Drupala od podstaw,
- rozumieć, jak korzystać z mechanizmu haków,
- wiedzieć, jak należy zapisywać i odczytywać ustawienia charakterystyczne dla danego modułu,
- tworzyć i przetwarzać proste formularze, korzystając z API formularzy Drupala,
- umieć tworzyć nowe kategorie w obrębie głównej strony administracyjnej Drupala,
- tworzyć formularze dla administratorów stron zawierające pola typu checkbox, pola typu radio i pola tekstowe,
- umieć walidować ustawienia i wyświetlać komunikaty o błędach po wystąpieniu jakichkolwiek problemów,
- rozumieć, jak Drupal przechowuje ustawienia z wykorzystaniem systemu trwałych zmiennych.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ż

Drupal 7. Zaawansowane programowanie

Drupal to nie tylko kolejny, nieco lepszy od innych CMS. Ten system zarządzania treścią zrobił ostatnio oszałamiającą karierę jako doskonały framework do tworzenia wszelkiej maści aplikacji internetowych. Dzięki niezwykłej łatwości i szybkości, z jaką Drupal pozwala programistom tworzyć rozbudowane blogi, profesjonalne witryny korporacyjne czy serwisy społecznościowe, zainteresowanie tą technologią zaczęło dynamicznie wzrastać. Co więcej, choć już dziś wydaje się, że sposoby wykorzystania tego CMS-a ogranicza jedynie ludzka wyobraźnia, system jest wciąż intensywnie rozwijany przez liczną społeczność entuzjastów na całym świecie. Możliwość czerpania z ogromnych zasobów wiedzy tysięcy programistów poprawi jakość Twoich nawet najbardziej wyrafinowanych internetowych projektów.

Jeśli chcesz tworzyć nowoczesne, rozbudowane witryny internetowe z wykorzystaniem Drupala 7, właśnie znalazłeś idealny podręcznik dla siebie! Omówiono tu wszystko, co będzie Ci potrzebne, począwszy od podstawowych zagadnień, takich jak architektura i struktura plików systemu, przez sposoby wykorzystania API formularzy czy dodawania własnych modułów w celu rozszerzania możliwości, aż po metody tworzenia bezpiecznego, wydajnego kodu. Dowiesz się więcej na temat pracy z bazą danych, uruchomienia własnej strony wyszukiwarki oraz korzystania z jQuery w Drupalu. Nauczysz się także sprawnie optymalizować Drupal i przeprowadzać skuteczne testy oraz poznasz wiele innych praktycznych zagadnień, które sprawią, że bez trudu zrozumiesz zasady działania tego CMS-a oraz pokonasz wszelkie problemy napotykane w trakcie pracy.

- Jak działa Drupal i jaką ma architekturę
- Jak tworzyć dodatkowe, funkcjonalne moduły
- Jak pracować z bazą danych i API formularzy
- Jak wygląda obsługa użytkowników, segmentów i pól
- Jak zapewnić wyszukiwanie i indeksowanie informacji
- Jak tworzyć czysty, bezpieczny kod
- Jak przechowywać dane w sesjach
- Jak optymalizować działanie Drupala
- Jak przeprowadzić efektywny proces testowania
- Jak przygotowywać testy jednostkowe

Nr katalogowy: 6872

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/nowosci

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

helion.pl
księgarnia internetowa

Cena 79,00 zł
ISBN 978-83-246-3367-8
9 788324 633678

Informatyka w najlepszym wydaniu