WŁODZIMIERZ GAJDA

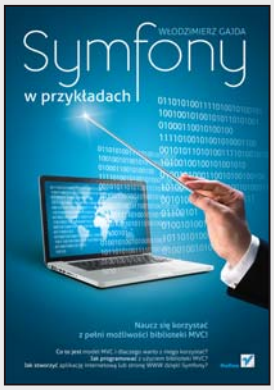
Symfony w przykładach

Naucz się korzystać
z pełni możliwości biblioteki MVC!

Co to jest model MVC i dlaczego warto z niego korzystać?
Jak programować z użyciem biblioteki MVC?
Jak stworzyć aplikację internetową lub stronę WWW dzięki Symfony?

Helion

Symfony w przykładach



Autor: Włodzimierz Gajda
ISBN: 978-83-246-2788-2
Format: 158×235, stron: 384

Naucz się korzystać z pełni możliwości biblioteki MVC!

- Co to jest model MVC i dlaczego warto z niego korzystać?
- Jak programować z użyciem biblioteki MVC?
- Jak stworzyć aplikację internetową lub stronę WWW dzięki Symfony?

Symfony, framework stworzony w języku PHP i mający na celu uproszczenie oraz przyspieszenie tworzenia aplikacji internetowych, znajduje zastosowanie w coraz większej liczbie projektów. Jego wykorzystanie wiąże się ze znacznie efektywniejszym programowaniem, a także pozwala uniknąć wielu błędów i powtarzających się, nużących czynności. Symfony opiera się na modelu MVC i posiada wiele wbudowanych funkcji – między innymi ochronę przed atakami CSRF oraz XSS. Ten framework nie ogranicza się do wykorzystania własnej biblioteki, lecz zapewnia także możliwość integracji z innymi. Jeśli chcesz nauczyć się, jak to działa w praktyce, trzymasz w rękach właściwą pozycję!

Książka „Symfony w przykładach" jest możliwie najbardziej skondensowaną instrukcją obsługi Symfony. Żeby ją zrozumieć, nie musisz dysponować oszałamiającą wiedzą – wystarczą podstawy PHP i XHTML/CSS. Jej autor poprowadzi Cię od najprostszych projektów ("Hello world"), przez nieco bardziej zaawansowane zagadnienia, dotyczące zewnętrznych zasobów, połączenia projektu z bazą danych, publikacji projektu na serwerze hostingowym, aż po tworzenie różnego typu paneli administracyjnych. Krótko mówiąc, na samych konkretnych przykładach przejdziesz drogę do stworzenia własnej, niezawodnie działającej aplikacji internetowej.

- Pierwszy projekt w Symfony i praca w środowisku NetBeans
- Wymiana szablonu XHTML/CSS i dołączanie zewnętrznych zasobów
- Hiperłącza i strona błędu 404
- Publikowanie projektu na serwerze hostingowym
- Dostosowywanie klas generowanych przez Propel
- Wyświetlanie danych rekordu i identyfikacja rekordów na podstawie wartości slug
- Artykuły na temat HTML/CSS
- Umieszczanie w bazie danych plików binarnych
- Pliki do pobrania i komponent menu
- Relacje 1:n oraz n:m i widoki częściowe
- Panele administracyjne i tłumaczenie interfejsu witryny
- Zbiór zadań C++
- Administracja kontami użytkowników i generowanie paneli administracyjnych
- Zabezpieczanie paneli administracyjnych protokołem HTTPS

I Ty możesz ułatwić sobie tworzenie doskonałych aplikacji internetowych!

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

Spis treści

Rozdział 22.

Kontekstowe hiperłącza do edycji i usuwania rekordów

Projekt omawiany w tym rozdziale zademonstruje metodę połączenia aplikacji `frontend` oraz `backend` w taki sposób, by zalogowany administrator mógł przechodzić pomiędzy trybami odczytu i edycji rekordu. W aplikacji `frontend` umieścimy hiperłącza przenoszące użytkownika do trybu edycji rekordów (czyli do aplikacji `backend`). W aplikacji `backend` dodamy hiperłącza przenoszące do trybu odczytu (czyli do aplikacji `frontend`). Te hiperłącza będą dostępne tylko dla zalogowanego administratora; przedstawimy je w postaci ikon. Takie rozwiązanie sprawi, że korzystanie z panelu administracyjnego będzie znacznie wygodniejsze.

Dodatkowymi ułatwieniami będą: automatyczne podpowiadanie domyślnych wartości wybranych pól oraz kontekstowe działanie usuwania rekordów.

W poniżej omawianym przykładzie wystąpią dwie tabele `rozdzial` oraz `zadanie` połączone relacją *1:n*. Automatyczne podpowiadanie wartości będzie dotyczyło tworzenia nowych rekordów. W przypadku dodawania do bazy danych rekordu do tabeli `rozdzial` automatycznie wypełnimy pole `numer`, nadając mu następną dostępną wartość. Gdy zostanie dodane nowe zadanie, generowanymi wartościami będą numer zadania oraz — w niektórych sytuacjach — numer rozdziału. Ponadto zarówno w wypadku rozdziałów, jak i zadań, automatycznie wygenerujemy wartości kolumn `slug`.

Kontekstowość działania przycisku do usuwania rekordów z tabeli `zadanie` będzie polegała na tym, że po usunięciu rekordu wyświetlimy stronę, na której naciśnięto przycisk *usuń*. Przycisk do usuwania zadania znajdziemy na trzech różnych stronach. Będą to:

- lista wszystkich zadań (przycisk *A*),
- lista zadań z wybranego rozdziału (przycisk *B*),
- szczegółowe dane zadania (przycisk *C*).

Jeśli naciśnięto przycisk *A*, to ma nastąpić powrót do listy wszystkich zadań. Jeśli naciśnięto przycisk *B* lub *C*, to ma nastąpić powrót do listy zadań z wybranego rozdziału. Kontekstowość usuwania zaimplementujemy, zapisując odwiedzane adresy URL w sesji użytkownika oraz wykorzystując funkcję `redirect()`, która wykonuje przekierowania HTTP.

Przykład 22. Zbiór zadań C++

Wykonaj aplikację internetową prezentującą w postaci witryny WWW zbiór zadań z programowania w języku C++. Zbiór zadań jest podzielony na rozdziały. Każdy z rozdziałów może zawierać dowolną liczbę zadań. Projekt powinien składać się z dwóch aplikacji: `frontend` oraz `backend`. Aplikacja `frontend` ma udostępniać całą treść zbioru zadań w trybie do odczytu wszystkim odwiedzającym. Aplikacja `backend` ma umożliwiać edycję całego zbioru zadań. Dostęp do aplikacji `backend` zabezpiecz wtyczką `sfGuardAuth`.

Aplikacje `frontend` oraz `backend` wzbogać o ikony ułatwiające przechodzenie pomiędzy trybami edycji i odczytu wszystkich rekordów. W całym projekcie przy każdym wyświetlanym rekordzie — zarówno w akcji `index`, jak i `show` — umieść:

- w aplikacji `frontend` ikonę przechodzącą do edycji rekordu,
- w aplikacji `backend` ikony: *edycja*, *odczyt*, *usuwanie*.

Operację usuwania zaimplementuj w taki sposób, by po usunięciu rekordu następował powrót do strony, na której naciśnięto przycisk *usuń*. Ikony mają być widoczne tylko po zalogowaniu na konto administracyjne.

ROZWIĄZANIE

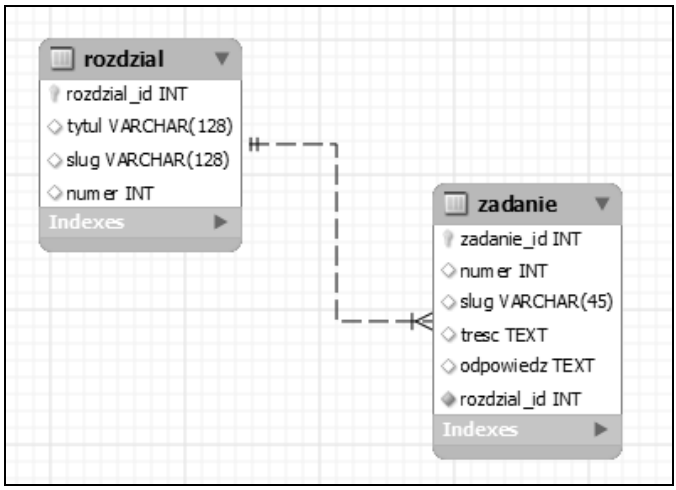
Etap 1. Utwórz nowy projekt i wykonaj aplikację frontend

Krok 1. Utwórz pustą bazę danych

Przygotuj skrypty *tworzenie-pustej-bazy-danych.sql* i *tworzenie-pustej-bazy-danych.bat*, które utworzą pustą bazę danych `cpp` oraz konto `redaktor`. Poprawność tworzenia bazy sprawdź przy użyciu programu phpMyAdmin.

Krok 2. Zaprojektuj bazę danych

Zaprojektuj przedstawioną na rysunku 22.1 bazę danych `cpp`. Ta baza ma zawierać tabele `rozdzial` oraz `zadanie` połączone relacją *1:n*.



Rysunek 22.1.
Baza danych cpp

Krok 3. Utwórz projekt z aplikacją frontend

W folderze *cpp/* utwórz nowy projekt zawierający aplikację `frontend`:

```
symfony generate:project cpp --orm=Propel
symfony generate:app frontend
```

Krok 4. Skonfiguruj dostęp do bazy danych

Wydaj polecenie:

```
symfony configure:database "mysql:host=localhost;dbname=cpp" redaktor tajnehaslo
```

po czym w pliku *config/schema.yml* umieść strukturę bazy danych z rysunku 22.1. Właściwość `primaryString` dodaj:

- w tabeli `rozdzial` dla kolumny `tytul`,
- w tabeli `zadanie` dla kolumny `slug`.

Następnie przy użyciu polecenia:

```
symfony propel:build --all --no-confirmation
```

wygeneruj klasy dostępu i utwórz tabele w bazie danych. Po wydaniu tego polecenia za pomocą programu phpMyAdmin sprawdź, czy na serwerze MySQL w bazie danych `cpp` pojawiły się dwie tabele.

Krok 5. Rozszerz wygenerowane klasy dostępu do bazy danych

W klasach wygenerowanych przez Propel dodaj następujące metody:

- w klasie `Rozdzial` metody `setSlug()`, `getMaxNumerZadania()`,
- w klasie `RozdzialPeer` metody `retrieveBySlug()`, `retrieveByNumer()`, `pierwszyRozdzial()`, `insert()`, `getMaxNumerRozdzialu()`,

- w klasie `Zadanie` metodę `setSlug()`,
- w klasie `ZadaniePeer` metody `retrieveBySlug()`, `insert()`, `doSelect()`.

Metody `retrieveBySlug()`, `insert()`, `doSelect()` przygotuj tak, jak to wielokrotnie omawialiśmy w poprzedniej części. Metoda `retrieveByNumer()` klasy `RozdzialPeer` jest bardzo zbliżona do metody `retrieveBySlug()`: różni się tylko tym, że wyszukiwanie rekordu przeprowadzamy na podstawie kolumny `numer`, a nie `slug`.

Metoda `setSlug()` w klasach `Rozdzial` oraz `Zadanie` ma automatycznie ustalać wartość kolumny `slug`. Metody `getMaxNumerZadania()` oraz `getMaxNumerRozdzialu()` będą służyły do automatycznego wypełniania pól numer rozdziału i numer zadania przy tworzeniu nowych rekordów. Metoda `getMaxNumerZadania()` zwraca największy z numerów zadań wybranego rozdziału, a metoda `getMaxNumerRozdzialu()` zwraca największy numer rozdziału zawarty w bazie danych. Ostatnia z nowych metod, metoda `pierwszyRozdzial()`, zwraca pierwszy rekord z tabeli `rozdzial`. Wykorzystamy ją w akcji `rozdzial/show` do przechodzenia na pierwszą stronę zbioru zadań.

Metoda `setSlug()` klasy `Rozdzial` jest przedstawiona na listingu 22.1.

Listing 22.1. *Metoda setSlug() klasy Rozdzial*

```
public function setSlug($slug) {
    $slug = trim($slug);

    if ($slug == '') {
        $slug = myString::string2slug($this->getTytul());
    } else {
        $slug = myString::string2slug($slug);
    }

    $next_slug = $slug;
    $c = new Criteria();
    $c->add(RozdzialPeer::SLUG, $next_slug);
    $c->add(RozdzialPeer::ROZDZIAL_ID, $this->getRozdzialId(),
    ↪Criteria::NOT_EQUAL);
    $ile = RozdzialPeer::doCount($c);

    $unikatowy = ($ile == 0);

    $min = 2;
    $max = 1000;

    while (!$unikatowy) {

        $next_slug = $slug . '__' . $min;
        $min++;

        if ($min > $max + 1) {
            die("****** ERROR    RozdzialPeer::setSlug({$next_slug})");
        };

        $c->clear();
        $c->add(RozdzialPeer::SLUG, $next_slug);
```

```
            $c->add(RozdzialPeer::ROZDZIAL_ID, $this->getRozdzialId(),
            ↪Criteria::NOT_EQUAL);
            $ile = RozdzialPeer::doCount($c);
            $unikatowy = ($ile == 0);
        }

        parent::setSlug($next_slug);
    }
```

Jest to metoda wirtualna, nadpisująca metodę `setSlug()` wygenerowaną przez Propel. Najpierw przy użyciu funkcji `trim()` usuwamy zbędne białe znaki, po czym sprawdzamy, czy otrzymany parametr jest pusty. Jeśli tak, to w zmiennej `$slug` umieszczamy tytuł rozdziału przekształcony przy użyciu funkcji `string2slug()`. Dzięki takiej procedurze będziemy mieli pełny wpływ na kolumnę `slug`, a jednocześnie będziemy mogli korzystać z automatycznego generatora. Jeśli w wypełnionym formularzu pozostawimy puste pole `slug`, wówczas wartość dla tej kolumny zostanie automatycznie wygenerowana na podstawie tytułu rozdziału. Jeśli zechcemy nadać konkretną wartość kolumnie `slug`, należy ją wprowadzić w formularzu. Spowoduje to wyłączenie automatycznego generowania wartości kolumny `slug`.

Gdy wartość `slug` została wstępnie ustalona, sprawdzamy, czy nie występuje ona w bazie danych. Operację taką powtarzamy w pętli aż do znalezienia wartości, która nie wystąpiła w bazie danych. W kolejnych obrotach pętli na końcu zmiennej `$slug` dołączamy kolejne liczby całkowite:

```
lorem_ipsum
lorem_ipsum2
lorem_ipsum3
lorem_ipsum4
...
```

i badamy, czy otrzymany napis nie występuje w kolumnie `slug` tabeli `rozdzial`. Iterację kończymy, gdy znajdziemy wartość unikalną lub gdy pętla sprawdzająca unikalność obróci się zbyt wiele razy (np. więcej niż 1000).

Znaleziona unikalna wartość `slug` jest przekazywana jako parametr do metody `setSlug()` w klasie bazowej.

Metoda `setSlug()` klasy `Zadanie` jest niemal identyczna.

Metoda `getMaxNumerZadania()` klasy `Rozdzial` jest przedstawiona na listingu 22.2.

Listing 22.2. *Metoda getMaxNumerZadania() klasy Rozdzial*

```
public function getMaxNumerZadania() {
    $c = new Criteria();
    $c->addDescendingOrderByColumn(ZadaniePeer::NUMER);
    $c->add(ZadaniePeer::ROZDZIAL_ID, $this->getRozdzialId());
    $c->setLimit(1);
    $Zadanie = ZadaniePeer::doSelectOne($c);
    if ($Zadanie) {
        return $Zadanie->getNumer();
    } else {
```

```
            return 0;
        }
    }
```

Zadaniem tej metody jest znalezienie największego numeru zadania w bieżącym rozdziale. Tworzymy kryteria, które zwrócą listę zadań z wybranego rozdziału. Zwracane wyniki sortujemy malejąco względem kolumny `numer` i ograniczamy do jednego rekordu. Wynikiem funkcji jest 0 lub wartość kolumny `numer` otrzymanego rekordu. Statyczna metoda `RozdzialPeer::getMaxNumerRozdzialu()` jest bardzo podobna.

Krok 6. Przygotuj zadanie propel:import-danych

Skopiuj:

- folder z folderu *22-start/dane-zbior-zadan/* do folderu *cpp/data/*,
- pliki z folderu *22-start/lib/* do folderu *cpp/lib/*.

Następnie utwórz zadanie `propel:import-danych`:

```
symfony generate:task propel:import-danych
```

W pliku *lib/task/propelImportdanychTask.class.php* wprowadź kod, który na podstawie plików z folderu *dane-zbior-zadan/* wypełni bazę danych. Zarys kodu zadania `propel:import-danych` jest przedstawiony na listingu 22.3. W tym skrypcie przetwarzamy najpierw plik *rozdzialy.txt*, a następnie wszystkie pliki wyszukane przy wykorzystaniu funkcji `glob()` w podfolderze *txt/*. Rozwiązania niektórych zadań są zawarte w folderze *dane-zbior-zadan/cpp*. Dostępność rozwiązania sprawdzamy przy użyciu funkcji `file_exists()`. Jeśli rozwiązanie jest dostępne, to plik z rozwiązaniem odczytujemy do zmiennej `$dane['odpowiedz']`, pamiętając o tym, że kod C++ może zawierać znaki <, > oraz &. Dlatego wartość zwrócona przez funkcję `file_get_contents()` jest przekształcona przy użyciu funkcji `htmlspecialchars()`.

Listing 22.3. *Fragment pliku propelImportdanychTask.class.php*

```
//rozdzialy
$tmp_rozdzialy = string2HArray(file_get_contents('data/dane-zbior-zadan/
➥rozdzialy.txt'));
foreach ($tmp_rozdzialy['items'] as $tmp_rozdzial) {
    $dane = array(
        'tytul'  => $tmp_rozdzial[0],
        'slug' => string2slug($tmp_rozdzial[0]),
        'numer' => $tmp_rozdzial[1],
    );
    RozdzialPeer::insert($dane);
}

//zadania
$plks = glob('data/dane-zbior-zadan/txt/*.txt');
foreach ($plks as $plk) {
    preg_match('/^(\d+)-(\d+)\.txt$/', basename($plk), $m);
```

```
        $nr_rozdzialu = $m[1];
        $nr_zadania = $m[2];

        $rozdzial = RozdzialPeer::retrieveByNumer($nr_rozdzialu);
        if (!$rozdzial) {
            die('blad ###1');
        }

        $slug =
            uzupelnij_int_zerami($nr_rozdzialu, 2) . '-' .
            uzupelnij_int_zerami($nr_zadania, 2);

        $dane = array(
            'numer' => $nr_zadania,
            'slug' => $slug,
            'tresc'  => file_get_contents($plk),
            'rozdzial_id' => $rozdzial->getRozdzialId()
        );

        $np_cpp = str_replace('txt', 'cpp', $plk);

        if (file_exists($np_cpp)) {
            $dane['odpowiedz'] = htmlspecialchars(file_get_contents($np_cpp));
        }

        ZadaniePeer::insert($dane);
    }
```

Krok 7. Wypełnij bazę danych rekordami

Wydaj komendę:

```
symfony propel:import-danych
```

W bazie danych `cpp` pojawi się 298 rekordów. Sprawdź to za pomocą programu phpMyAdmin.

Krok 8. Wykonaj moduł rozdzial

Wygeneruj moduł CRUD dla tabeli `rozdzial`:

```
symfony propel:generate-module --with-show frontend rozdzial Rozdzial
```

W tym module usuń wszystkie akcje oprócz akcji `show`. Dodaj dwie nowe akcje: `blad404` oraz `rozwiazanie`.

W metodzie `executeShow()` wykorzystaj metodę `pierwszyRozdzial()`. Jeśli parametr `slug` jest zdefiniowany, to akcja `show` ma przekazywać do widoku wybrany rekord. W przeciwnym razie przekaż do widoku pierwszy rozdział. Kod metody `execute➥Show()` jest przedstawiony na listingu 22.4.

Listing 22.4. *Metoda akcji rozdzial/show*

```
public function executeShow(sfWebRequest $request)
{
    if ($request->getParameter('slug')) {
        $this->Rozdzial
        ↪= RozdzialPeer::retrieveBySlug($request->getParameter('slug'));
    } else {
        $this->Rozdzial = RozdzialPeer::pierwszyRozdzial();
    }
    $this->forward404Unless($this->Rozdzial);
}
```

Metoda `executeRozwiazanie()` będzie wykorzystana do wykonania hiperłączy, pozwalających na pobieranie rozwiązań zadań w postaci plików o rozszerzeniu *.cpp*. W treści metody mamy przekazać do widoku obiekt klasy `Zadanie`, którego identyfikator `slug` jest zawarty w zapytaniu HTTP. Obiekt `Zadanie` przekazujemy do widoku wyłącznie wtedy, gdy ma on niepuste rozwiązanie. Trzy warunki:

- istnienie parametru `slug`,
- istnienie zadania o podanej wartości parametru `slug`,
- oraz to, czy zadanie ma niepuste rozwiązanie,

łączymy spójnikami `&&` i przekazujemy do metody `forward404Unless()`. Jeśli którykolwiek warunek nie jest spełniony, to sterowanie zostanie przekazane do obsługi błędu 404. Drugi z warunków zawiera instrukcję przypisania, która spowoduje przekazanie do widoku zmiennej `$Zadanie`. Treść metody `executeRozwiązanie()` jest przedstawiona na listingu 22.5.

Listing 22.5. *Metoda akcji rozdzial/rozwiazanie*

```
public function executeRozwiazanie(sfWebRequest $request)
{
  $this->forward404Unless(
      $request->getParameter('slug') &&
      ($this->Zadanie
      ↪= ZadaniePeer::retrieveBySlug($request->getParameter('slug'))) &&
      $this->Zadanie->getOdpowiedz()
  );
}
```

Krok 9. Przygotuj widoki akcji rozdzial/show, rozdzial/rozwiązanie oraz rozdzial/blad404

Na stronie akcji `rozdzial/show` drukujemy tytuł rozdziału oraz kompletną listę wszystkich zadań z rozdziału. Tytuł rozdziału jest zwracany przez funkcję `__toStrung()`, więc wydruk tytułu przyjmuje postać:

```
<?php echo $Rozdzial ?>
```

Natomiast zadania z rozdziału są zwracane jako wynik metody `getRozdzials()`. Otrzymaną tablicę obiektów klasy `Zadanie` przetwarzamy iteracyjnie. Dla każdego zadania drukujemy nagłówek `h3` zawierający tytuł:

```
<h3  id="zad<?php echo $Zadanie->getSlug() ?>">
    Zadanie <?php echo $Rozdzial->getNumer() ?>.<?php echo $Zadanie->getNumer() ?>
</h3>
```

Zauważ, że ten element jest wzbogacony o identyfikator `id`. Dzięki temu w panelu administracyjnym będziemy mogli korzystać z hiperłączy postaci:

```
/rozdzial/wprowadzenie.html#zad01-04
```

Po numerze zadania drukujemy jego treść oraz sprawdzamy, czy rozwiązanie jest dostępne. Jeśli tak, to umieszczamy je poniżej treści zadania. Obok napisu *rozwiązanie* drukujemy hiperłącze do akcji `rozdzial/rozwiazanie`, które pozwoli na pobranie rozwiązania w postaci pliku *.cpp*. Treść rozwiązania umieszczamy w elemencie `pre` wzbogaconym o atrybut `class="syntax-highlight:cpp"`. W ten sposób osiągniemy kolorowanie składni kodu C++. Treść widoku *showSuccess.php* jest przedstawiona na listingu 22.6.

Listing 22.6. *Widok akcji rozdzial/show*

```
<h1>Rozdział <?php echo $Rozdzial->getNumer() ?>. <?php echo $Rozdzial ?></h1>
<?php foreach ($Rozdzial->getZadanies() as $Zadanie): ?>
  <div class="zadanie">
    <h3  id="zad<?php echo $Zadanie->getSlug() ?>">
      Zadanie <?php echo $Rozdzial->getNumer() ?>.<?php echo $Zadanie->getNumer() ?>
    </h3>
    <?php echo $Zadanie->getTresc() ?>
    <?php if ($Zadanie->getOdpowiedz()): ?>
      <div class="rozwiazanie">
        <h4>
          Rozwiązanie:
            <a href="<?php echo url_for('rozdzial/rozwiazanie?slug=' .
            ➥$Zadanie->getSlug()) ?>">
              <?php echo $Zadanie->getSlug() ?>.cpp
            </a>
        </h4>
        <pre class="syntax-highlight:cpp"><?php echo $Zadanie->getOdpowiedz()
        ➥?></pre>
      </div>
    <?php endif; ?>
  </div>
<?php endforeach; ?>
```

W akcji `rozdzial/rozwiazanie` mamy za zadanie wydrukować wyłącznie treść rozwiązania zadania. Widok *rozwiązanieSuccess.php* jest przedstawiony na listingu 22.7. Rozwiązania nie należy dekorować plikiem *layout.php*. Zatem w pliku konfiguracyjnym widoków modułu, czyli *apps/frontend/modules/rozdzial/config/view.yml*, należy umieścić reguły, które dla widoku `rozwiazanieSuccess` wyłączą dekorację i ustalą nagłówek `text/plain`:

```
rozwiazanieSuccess:
  http_metas:
    content-type: text/plain
  has_layout:     false
```

Listing 22.7. *Widok akcji rozdzial/rozwiazanie*

```
<?php echo html_entity_decode($Zadanie->getOdpowiedz());
```

Ponieważ w listingu 22.6 wykorzystujemy zmienną `$Zadanie`, a nie jej surową postać[1], więc w pliku konfiguracyjnym *settings.yml* aplikacji `frontend` należy wyłączyć zabezpieczanie zmiennych:

```
all:
  .settings:
    escaping_method: ESC_RAW
```

W widoku akcji `rozdzial/blad404` umieść komunikat o błędnym adresie URL.

Krok 10. Przygotuj komponent rozdzial/menu

Menu witryny ma zawierać listę wszystkich rozdziałów. Wykonamy je w postaci komponentu o nazwie `rozdzial/menu`. Utwórz plik *rozdzial/actions/components.class.php* i zdefiniuj w nim przedstawioną na listingu 22.8 klasę `rozdzialComponents`. Metoda `executeMenu()` ma przekazywać do widoku listę wszystkich rekordów z tabeli `rozdzial`.

Listing 22.8. *Klasa rozdzialComponents*

```
class rozdzialComponents extends sfComponents
{
  public function executeMenu(sfWebRequest $request)
  {
    $this->Rozdzials = RozdzialPeer::doSelect(new Criteria());
  }
}
```

W widoku *rozdzial/templates/_menu.php* umieść kod z listingu 22.9. Tablica `$Rozdzials` jest przekształcona w listę `ol` zawierającą hiperłącza do akcji `show` prezentujących szczegółowe informacje poszczególnych rozdziałów.

Listing 22.9. *Widok komponentu rozdzial/menu*

```
<ol id="menu">
  <?php foreach ($Rozdzials as $Rozdzial): ?>
    <li>
      <a href="<?php echo url_for('rozdzial/show?slug='.$Rozdzial->getSlug()) ?>">
        <?php echo $Rozdzial->getNumer() ?>.
        <?php echo $Rozdzial ?>
        <span>&raquo;</span>
      </a>
```

[1] Surowa postać zmiennej `$Zadanie` jest dostępna jako `$sf_data->getRaw('Zadanie')`.

```
    </li>
  <?php endforeach; ?>
</ol>
```

Krok 11. Zmodyfikuj skórkę witryny

Użyj szablonu zawartego w folderze *22-start/xhtml-css-template/*. Pamiętaj o modyfikacjach w pliku *frontend/config/view.yml*. Ustal pliki stylów oraz dołącz skrypty JS kolorujące kod:

```
stylesheets:    [style.css,SyntaxHighlighter.css]
javascripts:    [shCore.js,shBrushCpp.js]
```

Pamiętaj o skopiowaniu plików do folderów */web/css/*, */web/images/* oraz */web/js/*.

Menu główne wstaw do pliku *layout.php*, wywołując funkcję `include_component()`.

Na dole pliku *layout.php*, przed zamykającym znacznikiem `</body>`, wstaw skrypt, który będzie kolorował kod zawarty w elementach `pre` klasy `syntax-highlight:cpp`:

```
  ...
  <script type="text/javascript" >
    dp.SyntaxHighlighter.ClipboardSwf =
      '<?php echo public_path('js/clipboard.swf') ?>';
    dp.SyntaxHighlighter.HighlightAll('code');
  </script>
</body>
```

Krok 12. Ustal tytuły stron

W pliku *layout.php* umieść slot o nazwie `tytul`. Wypełnij ten slot danymi w widokach akcji `rozdzial/show` oraz `rozdzial/blad404`. Pomiędzy znacznikami `<title>` i `</title>` umieść komunikat o błędzie oraz tytuł rozdziału.

Krok 13. Ustal obsługę błędu 404

W pliku *frontend/config/settings.yml* wprowadź reguły definiujące obsługę błędu 404:

```
error_404_module: rozdzial
error_404_action: blad404
```

Krok 14. Ustal przyjazne adresy URL

W pliku *frontend/config/routing.yml* wprowadź reguły przedstawione na listingu 22.10. Reguła o etykiecie `rozwiazanie` ustala, że adres postaci:

```
/cpp/01-04.cpp
```

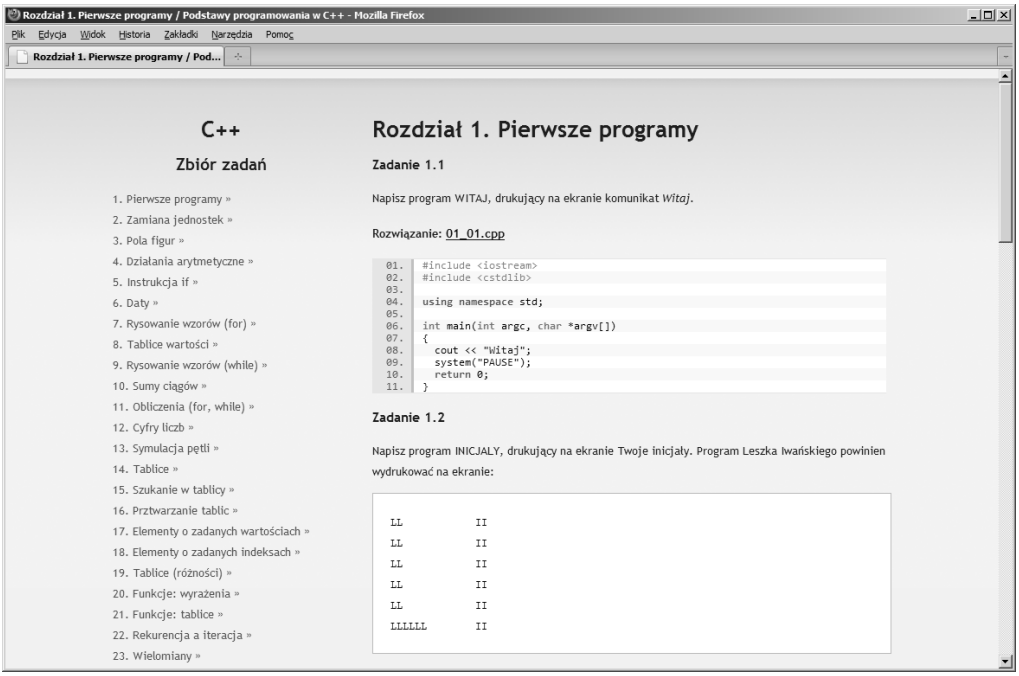
będzie powodował pobranie rozwiązania zadania o numerze 1.4. W ten sposób emulujemy, że w folderze *web/* znajduje się folder *cpp/*, który zawiera statyczne pliki o rozszerzeniu *.cpp*. W rzeczywistości rozwiązania zadań są pobierane z kolumny `odpowiedz` tabeli `zadanie`. Realizuje to akcja `rozdzial/rozwiazanie`.

Listing 22.10. *Reguły translacji adresów w aplikacji frontend*

```
rozwiazanie:
  url:   /cpp/:slug.cpp
  param: { module: rozdzial, action: rozwiazanie }

rozdzial_show:
  url:   /rozdzial/:slug.html
  param: { module: rozdzial, action: show }

index:
  url:   /index.html
  param: { module: rozdzial, action: show }

homepage:
  url:   /
  param: { module: rozdzial, action: show }
```

Krok 15. Przetestuj aplikację frontend

Przetestuj przy użyciu przeglądarki aplikację `frontend`. Powinieneś otrzymać witrynę widoczną na rysunku 22.2. Sprawdź działanie hiperłączy zawartych w menu głównym oraz pozwalających na pobieranie plików *.cpp*.



Rysunek 22.2. *Aplikacja frontend zbioru zadań C++*

Etap 2. Wykonaj aplikację backend

Krok 1. Utwórz aplikację backend

W folderze *cpp/* wydaj kolejno polecenia:

```
symfony generate:app backend
symfony plugin:install sfGuardPlugin
symfony propel:build --all --no-confirmation
symfony propel:import-danych
symfony propel:generate-module backend rozdzial Rozdzial
symfony propel:generate-module backend zadanie Zadanie
symfony guard:create-user admin supertajnehaslo
```

Po wygenerowaniu modułów `rozdzial` oraz `zadanie` zmodyfikuj komunikaty zawarte we wszystkich widokach. W miejsce angielskich terminów, np. *New Zadanie* czy *Edit Rozdzial*, wpisz polskie tłumaczenia.

Na stronie ze szczegółowymi danymi rozdziału wyświetl listę wszystkich zadań z danego rozdziału. Wykonaj to, przygotowując widok częściowy *zadanie/_lista.php*. Ten widok wykorzystaj na stronach: `zadanie/index` oraz `rozdzial/show`.

Krok 2. Zablokuj dostęp do aplikacji backend

W pliku *backend/config/security.yml* wprowadź regułę:

```
default:
  is_secure: true
```

Następnie wyłącz cytowanie zmiennych, włącz moduł `sfGuardAuth`, ustal akcje odpowiedzialne za logowanie i zmień domyślny język aplikacji:

```
//plik backend/config/settings.yml
all:
  .settings:
    escaping_method:  ESC_RAW
    enabled_modules:  [sfGuardAuth]
    login_module:     sfGuardAuth
    login_action:     signin
    secure_module:    sfGuardAuth
    secure_action:    secure
    i18n:             on
    default_culture:  pl_PL
```

W folderze *backend/i18n/* umieść plik *messages.pl.xml*, który przygotowałeś, wykonując przykład z rozdziału 21. Popraw formularz do logowania zawarty w pliku *cpp\sfGuardPlugin\modules\sfGuardAuth\templates\signinSuccess.php*.

Wykonaj zmiany opisane w poprzednim przykładzie lub użyj pliku z przykładu 21.

Zaletą korzystania z rozszerzeń `i18n` jest to, że tłumaczenie wykonujemy jeden raz. W kolejnych projektach będziemy wykorzystywali ten sam plik *messages.pl.xml*, który przygotowaliśmy, wykonując projekt z rozdziału 21.

Ustal adres, na który będziemy przekierowywani po wylogowaniu z aplikacji `backend`:

```
//plik backend/config/app.yml
all:
  sf_guard_plugin_success_signout_url: /cpp/web/
```

Zdefiniuj filtr, który pozwoli na zapamiętanie zalogowanej sesji:

```
//plik backend/config/filters.yml
remember_me:
  class: sfGuardRememberMeFilter
```

oraz ustal adres strony głównej:

```
//plik backend/config/routing.yml
homepage:
  url:   /
  param: { module: rozdzial, action: index }
```

Zabezpieczanie aplikacji zakończ, modyfikując klasę bazową klasy `myUser` zawartej w pliku *backend/lib/myUser.class.php*:

```
class myUser extends sfGuardSecurityUser
{
}
```

Po tej zmianie odwiedź adres, używając przeglądarki:

```
http://localhost/cpp/web/backend.php/
```

Powinieneś ujrzeć panel do logowania. Po zalogowaniu na konto `admin` uzyskasz dostęp do domyślnego panelu CRUD tabeli `rozdzial`.

Krok 3. Zmodyfikuj skórkę witryny

Użyj szablonu *layout.php* z aplikacji `frontend`. Menu główne wykonaj tym razem jako zaszyty na stałe w pliku *layout.php* kod przedstawiony na listingu 22.11.

Listing 22.11. *Menu główne aplikacji backend*

```
<ol id="menu">
    <li><a href="<?php echo public_path('') ?>">Czytaj
<span>&raquo;</span></a></li>
    <?php if ($sf_user->isAuthenticated()): ?>
        <li><a href="<?php echo url_for('sfGuardAuth/signout');
        ↪?>">Wyloguj <span>&raquo;</span></a></li>
        <li><a href="<?php echo url_for('rozdzial/index');
        ↪?>">Rozdziały <span>&raquo;</span></a></li>
        <li><a href="<?php echo url_for('zadanie/index');
        ↪?>">Zadania <span>&raquo;</span></a></li>
    <?php endif; ?>
</ol>
```

Pierwsza z opcji menu ma powodować przejście do aplikacji `frontend`. Druga opcja służy do wylogowania. Opcje *Rozdziały* oraz *Zadania* przechodzą do paneli CRUD dla tabel `rozdzial` i `zadanie`. Ostatnia z opcji automatycznie numeruje rozdziały i zadania

zawarte w zbiorze. Funkcja pomocnicza `public_path()` wygeneruje ścieżkę prowadzącą do folderu *web/*. Pozostałe opcje menu są dostępne tylko wtedy, gdy użytkownik jest zalogowany.

Zmodyfikuj plik *view.yml* konfigurujący widoki aplikacji `backend`. Ustal w nim tytuły wszystkich stron *Edycja zbioru zadań* oraz zmień nazwę pliku CSS na *style.css*.

Krok 4. Zmodyfikuj właściwości prezentacyjne formularzy

Zmień wygląd formularzy edycyjnych dla tabel `rozdzial` oraz `zadanie`. W plikach *lib/form/RozdziaForm.class.php* oraz *lib/form/ZadanieForm.class.php* wprowadź kod przedstawiony na listingach 22.12 oraz 22.13. Metoda `setLabels()` ustala etykiety poszczególnych pól formularza, a metoda `setAttributes()` zmienia rozmiary kontrolek edycyjnych.

Listing 22.12. *Dostosowanie formularza edycyjnego rekordów z tabeli rozdzial*

```
class RozdzialForm extends BaseRozdzialForm {
    public function configure() {
        $this->widgetSchema->setLabels(array(
            'tytul' => 'Tytuł'
        ));
        $this->widgetSchema['tytul']->setAttributes(array('size' => 80));
        $this->widgetSchema['slug']->setAttributes(array('size' => 80));
        $this->widgetSchema['numer']->setAttributes(array('size' => 80));
    }
}
```

Listing 22.13. *Dostosowanie formularza edycyjnego rekordów z tabeli zadanie*

```
class ZadanieForm extends BaseZadanieForm {
    public function configure() {
        $this->widgetSchema->setLabels(array(
            'tresc' => 'Treść',
            'odpowiedz' => 'Odpowiedź',
            'rozdzial_id' => 'Rozdział'
        ));
        $this->widgetSchema['numer']->setAttributes(array('size' => 80));
        $this->widgetSchema['slug']->setAttributes(array('size' => 80));
        $this->widgetSchema['odpowiedz']->setAttributes(array('rows' => 12,
        ➥'cols' => 60));
        $this->widgetSchema['tresc']->setAttributes(array('rows' => 12,
        ➥'cols' => 60));
    }
}
```

Krok 5. Przetestuj aplikację backend

Wyczyść pamięć podręczną i odwiedź aplikację `backend`. Po zalogowaniu powinieneś uzyskać dostęp do częściowo zmodyfikowanego panelu administracyjnego. Pozwoli Ci on na dodawanie, usuwanie i uaktualnianie rekordów.

Etap 3. Połącz aplikacje frontend i backend

Krok 1. Zmodyfikuj menu główne aplikacji frontend

W menu głównym aplikacji `frontend` dodaj opcje umożliwiające zalogowanie. Przed pętlą `foreach` z listingu 22.9 dodaj instrukcje `if` widoczne na listingu 22.14.

Listing 22.14. *Zmodyfikowany widok _menu.php z aplikacji frontend*

```
<ol id="menu">
    <?php if ($sf_user->isAuthenticated()): ?>
        <li><a href="<?php echo public_path('backend.php')
        ➥?>">Edytuj <span>&raquo;</span></a></li>
        <li><a href="<?php echo public_path('backend.php/sfGuardAuth/signout');
        ➥?>">Wyloguj <span>&raquo;</span></a></li>
    <?php else: ?>
        <li><a href="<?php echo public_path('backend.php')
        ➥?>">Zaloguj <span>&raquo;</span></a></li>
    <?php endif; ?>

    <?php foreach ($Rozdzials as $Rozdzial): ?>
        <li><a href="<?php echo url_for('rozdzial/show?slug='.$Rozdzial->getSlug())
        ➥?>">
                <?php echo $Rozdzial->getNumer() ?>.
                <?php echo $Rozdzial ?>
                <span>&raquo;</span></a></li>
    <?php endforeach; ?>
</ol>
```

Menu główne aplikacji `backend`, które jest przedstawione na listingu 22.11, zawiera już opcję *Czytaj*, która powoduje przejście do aplikacji `frontend`. Pierwszy etap łączenia aplikacji `frontend` i `backend` jest zakończony. Przejście od jednej aplikacji do drugiej wykonasz za pośrednictwem opcji *Edytuj* i *Czytaj*.

> Bez względu na to, czy odwiedzasz aplikację `frontend` czy `backend`, metoda `isAu➥thenticated()` będzie zwracała poprawną informację o tym, czy jesteś zalogowany.

Krok 2. W aplikacji backend dodaj ikony edit, view, delete

W folderze */web/images/* umieść ikony *edit.png*, *tick.png* oraz *delete.png*. Ikony te znajdziesz w folderze[2] *22-start/ikony/*. Następnie zmodyfikuj widok akcji `rozdzial/index` w aplikacji `backend`. Ten widok jest przedstawiony na listingu 22.15.

Listing 22.15. *Widok akcji rozdzial/index w aplikacji backend*

```
<h1>Lista wszystkich rozdziałów</h1>
<p>
<a href="<?php echo url_for('rozdzial/new') ?>">Utwórz nowy rozdział</a>
```

[2] Te ikony są zawarte w pakiecie Symfony. Znajdziesz je w folderze *C:\php\data\symfony\web\sf\sf_admin\images*.

```
</p>

<table>
  <thead>
    <tr>
      <th>Numer</th>
      <th>Tytul</th>
      <th>Edytuj / Czytaj / Usuń</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($Rozdzials as $Rozdzial): ?>
    <tr>
      <td class="r"><?php echo $Rozdzial->getNumer() ?>.</td>
      <td><?php echo $Rozdzial ?></td>
      <td class="c">
          <?php echo link_to(image_tag('edit.png', array('alt' => '')),
          ➥'rozdzial/edit?rozdzial_id='.$Rozdzial->getRozdzialId()) ?>
          <?php echo link_to(image_tag('tick.png', array('alt' => '')),
          ➥public_path('rozdzial/' . $Rozdzial->getSlug() . '.html')) ?>
          <?php echo link_to(image_tag('delete.png', array('alt' => '')),
          ➥'rozdzial/delete?rozdzial_id='.$Rozdzial->getRozdzialId(),
          ➥array('method' => 'delete', 'confirm' => 'Czy na pewno usunąć rozdział?
          ➥Wszystkie zadania z rozdziału zostaną usunięte!')) ?>
      </td>
    </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

Ikona ułatwiająca edycję rekordu jest drukowana przy użyciu instrukcji:

```
<?php echo link_to(image_tag('edit.png', array('alt' => '')),
'rozdzial/edit?rozdzial_id='.$Rozdzial->getRozdzialId()) ?>
```

Przejście do aplikacji `frontend` realizuje druga ikona drukowana za pomocą instrukcji:

```
<?php echo link_to(image_tag('tick.png', array('alt' => '')),
public_path('rozdzial/' . $Rozdzial->getSlug() . '.html')) ?>
```

Instrukcja ta zakłada, że adres strony z rozdziałem w aplikacji `frontend` ma postać:

```
/rozdzial/slug-tytulu-rozdzialu.html
```

Trzecia z ikon, ikona do usuwania rekordu, wskazuje adres w aplikacji `backend`, zatem korzystamy z adresu wewnętrznego `rozdzial/delete?rozdzial_id=`. Ponieważ jednak jest to link do akcji `delete`, która jest zabezpieczona przed atakami CSRF, więc do funkcji pomocniczej `link_to()` przekazujemy dodatkowe parametry:

```
<?php echo link_to(image_tag('delete.png', array('alt' => '')),
➥'rozdzial/delete?rozdzial_id='.$Rozdzial->getRozdzialId(), array('method' =>
➥'delete', 'confirm' => 'Czy na pewno usunąć rozdział? Wszystkie zadania z
➥rozdziału zostaną usunięte!')) ?>
```

Podobne ikony dodaj w pozostałych widokach w aplikacji `backend`. Pamiętaj, że zadania nie są dostępne na osobnych stronach. Wszystkie zadania z rozdziału drukujemy na

stronie akcji `show` tegoż rozdziału. Hiperłącze zadania będzie więc zawierało identyfikator `#zad-xx-xx`. Łącze tego typu możesz wydrukować np. w akcji `zadanie/edit` w następujący sposób:

```
<h1>
    Edycja zadania
    <a href="<?php echo public_path('/rozdzial/' .
    ➥$Zadanie->getRozdzial()->getSlug() . '.html#zad' .  $Zadanie->getSlug()) ?>">
        <?php echo image_tag('tick.png', array('alt' => '')) ?>
    </a>
</h1>
```

Krok 3. W aplikacji frontend dodaj ikony edit

Pracę nad ikonami zakończ, modyfikując widok akcji `rozdzial/show` w aplikacji `fron ➥tend`. Dodaj ikony prowadzące do akcji `edit` w odpowiednim module aplikacji `backend`. Wszystkie ikony zabezpiecz sprawdzeniem, czy użytkownik jest zalogowany. W ten sposób ikony będą widoczne wyłącznie po zalogowaniu na konto administracyjne. Adresy do akcji `rozdzial/edit` oraz `zadanie/edit` w aplikacji `backend` drukujemy przy użyciu funkcji `public_path()`. Widok akcji `backend/rozdzial/show` jest przedstawiony na listingu 22.16.

Listing 22.16. *Widok akcji rozdzial/show aplikacji frontend po dodaniu ikon edit.png*

```
<h1>
    Rozdział <?php echo $Rozdzial->getNumer() ?>. <?php echo $Rozdzial ?>
    <?php if ($sf_user->isAuthenticated()): ?>
        <a href="<?php echo public_path('backend.php/rozdzial/edit/rozdzial_id/' .
        ➥$Rozdzial->getRozdzialId() ) ?>">
            <?php echo image_tag('edit.png', array('alt' => '')) ?>
        </a>
    <?php endif; ?>
</h1>
<?php foreach ($Rozdzial->getZadanies() as $Zadanie): ?>
    <div class="zadanie">
        <h3  id="zad<?php echo $Zadanie->getSlug() ?>">
            Zadanie <?php echo $Rozdzial->getNumer() ?>.<?php echo
            ➥$Zadanie->getNumer() ?>
            <?php if ($sf_user->isAuthenticated()): ?>
                <a href="<?php echo
                ➥public_path('backend.php/zadanie/edit/zadanie_id/' .
                ➥$Zadanie->getZadanieId() ) ?>">
                    <?php echo image_tag('edit.png', array('alt' => '')) ?>
                </a>
            <?php endif; ?>
        </h3>
    <?php echo $Zadanie->getTresc() ?>
    <?php if ($Zadanie->getOdpowiedz()): ?>
        <div class="rozwiazanie">
            <h4>Rozwiązanie: <a href="<?php echo url_for('rozdzial/
            ➥rozwiazanie?slug=' . $Zadanie->getSlug()) ?>"><?php echo
            ➥$Zadanie->getSlug() ?>.cpp</a></h4>
            <pre class="syntax-highlight:cpp"><?php echo $Zadanie->
            ➥getOdpowiedz() ?></pre>
        </div>
```

```
    <?php endif; ?>
    </div>
<?php endforeach; ?>
```

Etap 4. Kontekstowość usuwania rekordów

Kontekstowość operacji usuwania ma dotyczyć rekordów w tabeli `zadanie`. Mamy trzy przypadki usuwania rekordów:

- jeśli usunięcie zadania następuje na stronie akcji `zadanie/index`, to po usunięciu wracamy na tę samą stronę,
- jeśli usunięcie zadania następuje na stronie akcji `zadanie/show`, to po usunięciu również wracamy na stronę akcji `zadanie/index`,
- jeśli zaś rekord `zadanie` zostanie usunięty na stronie akcji `rozdzial/show`, to po usunięciu wracamy na tę samą stronę, czyli `rozdzial/show`.

W celu zapamiętania adresu URL, do którego należy powrócić, trzeba w sesji użytkownika zapamiętać kolejno odwiedzane strony. W metodzie `executeEdit()` modułu `rozdzial` oraz w metodzie `executeIndex()` modułu `zadanie` dodajemy instrukcje, które w sesji zapamiętają adres odwiedzonej strony. W metodzie akcji `zadanie/index` umieścimy kod:

```
$this->getUser()->setAttribute('prevUrl', 'zadanie/index');
```

a w metodzie akcji `rozdzial/edit` kod:

```
$this->getUser()->setAttribute('prevUrl', 'rozdzial/edit?rozdzial_id=' .
↪$Rozdzial->getRozdzialId());
```

Powrót do odpowiedniej strony WWW umieszczamy wewnątrz akcji `zadanie/delete`. Po usunięciu rekordu z tabeli `zadanie` odczytujemy wartość zmiennej sesyjnej `prevUrl`, po czym przechodzimy do adresu URL pobranego z sesji:

```
$url = $this->getUser()->getAttribute('prevUrl', 'zadanie/index');
$this->redirect($url);
```

Drugi parametr metody `getAttribute()` jest wartością domyślną. Jeśli zmienna `prevUrl` nie jest zapisana w sesji, wówczas po usunięciu rekordu przejdziemy na stronę z listą wszystkich zadań (tj. na stronę akcji `zadanie/index`).

Etap 5. Ułatwienia w wypełnianiu formularzy

Ostatnim krokiem udoskonalania projektu `cpp` będzie ułatwienie wypełniania formularzy. Formularze edycyjne zadań i rozdziałów już zawierają jedno ułatwienie. Kolumny `slug` w obu formularzach są automatycznie wypełniane na podstawie tytułu rozdziału lub numeru zadania. Automatyczne wypełnianie kolumn `slug` wykonaliśmy w taki sposób, że możemy korzystać zarówno z automatycznego, jak i ręcznego dostępu do kolumn

Instrukcja `setAttribute()` klasy `User` zapamiętuje w sesji zmienną o podanej nazwie i wartości. Po wywołaniu:

```
$this->getUser()->setAttribute('lorem', 'ipsum');
```

w sesji użytkownika będzie dostępna zmienna o nazwie `lorem` i wartości `ipsum`. Dostęp do tej zmiennej uzyskamy, wywołując metodę `getAttribute()`. Po wykonaniu instrukcji:

```
$zm = $this->getUser()->getAttribute('lorem');
```

w zmiennej zostanie zapisana wartość zmiennej `lorem` odczytana z sesji użytkownika.

`slug`. Jeśli wartość `slug` pozostawisz pustą, to jej wartość zostanie ustalona automatycznie. Jeśli natomiast w formularzu wprowadzisz jakiś napis, to zostanie on użyty do wygenerowania wartości `slug`. Poddamy go jedynie przekształceniu `string2slug()`. Taką funkcjonalność dla kolumn `slug` zapewniliśmy, nadpisując metody `setSlug()` w klasach `rozdzial` oraz `zadanie`.

Krok 1. Dodawanie zadań wewnątrz rozdziału

Drugim ułatwieniem będzie umożliwienie dodawania zadań wewnątrz konkretnego rozdziału. Na stronie akcji `rozdzial/edit` dodajemy hiperłącze:

```
<a href="<?php echo url_for('zadanie/new?rozdzial_id=' .
↪$Rozdzial->getRozdzialId()) ?>">Dodaj zadanie</a>
```

To hiperłącze przenosi do akcji `zadanie/new`, przekazując do niej parametr `rozdzial_id`, który zawiera identyfikator edytowanego właśnie rozdziału. W metodzie `executeNew()` modułu `zadanie` musimy sprawdzić, czy parametr `rozdzial_id` jest dostępny. Jeśli tak, to należy zainicjować pole definiujące powiązanie nowo tworzonego zadania z odpowiednim rekordem tabeli `rozdzial`. Metoda `executeNew()` modułu `zadanie` jest przedstawiona na listingu 22.17.

Listing 22.17. *Metoda executeNew() modułu zadanie*

```
public function executeNew(sfWebRequest $request)
{
    $this->form = new ZadanieForm();
    if (
        $request->getParameter('rozdzial_id') &&
        ($this->Rozdzial =
        ↪RozdzialPeer::retrieveByPk($request->getParameter('rozdzial_id')))
    ) {
        $this->form->getWidget('rozdzial_id')->setDefault($this->
        ↪Rozdzial->getRozdzialId());
        $numer = $this->Rozdzial->getMaxNumerZadania();
        $this->form->getWidget('numer')->setDefault($numer + 1);
        $this->form->getWidget('slug')->setDefault(myString::slugZadania($this->
        ↪Rozdzial->getNumer(), $numer + 1));
    }
}
```

Najpierw badamy, czy zmienna `rozdzial_id` jest podana oraz czy jej wartość jest poprawna. Jeśli tak, to wewnątrz instrukcji `if` będzie dostępny obiekt `$this->Rozdzial`, który tworzymy wewnątrz warunku. Na podstawie zmiennej `$this->Rozdzial` ustalamy wartość wyświetlaną w liście rozwijanej generowanej dla relacji *1:n*. Służy do tego metoda `setDefault()`:

```
$this->form->getWidget('rozdzial_id')->setDefault($this->Rozdzial->
➥getRozdzialId());
```

Następnie ustalamy domyślną wartość numeru zadania. Wykorzystujemy do tego metodę `getMaxNumerZadania()`, która zwraca największy z numerów zadań w bieżącym rozdziale:

```
$numer = $this->Rozdzial->getMaxNumerZadania();
$this->form->getWidget('numer')->setDefault($numer + 1);
```

Na zakończenie wstawiamy domyślną wartość kolumny `slug`. Przyjmie ona postać `xx-yy`, gdzie `xx` jest numerem rozdziału, a `yy` — numerem zadania:

```
$this->form->getWidget('slug')->setDefault(myString::slugZadania($this->Rozdzial->
➥getNumer(), $numer + 1));
```

Statyczna metoda `slugZadania()` klasy `myString` tworzy napis `xx-yy` na podstawie dwóch liczb.

Krok 2. Wstępne numerowanie nowo tworzonych rozdziałów

Wstępna numeracja nowych rozdziałów jest zaimplementowana w metodzie `execute➥New()` modułu `rozdzial`. Ta metoda jest przedstawiona na listingu 22.18. Po utworzeniu formularza w zmiennej `$numer` ustalamy największy zawarty w bazie danych numer rozdziału. Wykorzystujemy do tego statyczną metodę `getMaxNumerRozdzialu()`. Na podstawie utworzonej wartości przy użyciu metody `setDefault()` ustalamy domyślny numer nowo tworzonego rozdziału.

Listing 22.18. *Automatyczne numerowanie rozdziałów w metodzie executeNew() modułu rozdzial*

```
public function executeNew(sfWebRequest $request)
{
    $this->form = new RozdzialForm();
    $numer = RozdzialPeer::getMaxNumerRozdzialu();
    $this->form->getWidget('numer')->setDefault($numer + 1);
}
```

Krok 3. Automatyczna numeracja rozdziałów

Ostatnim ułatwieniem, jakie wykonamy w projekcie `cpp`, będzie automatyczne numerowanie rozdziałów i zadań. Jeśli w zbiorze zadań usuniesz kilka rekordów z obu tabel, wówczas w numeracji pojawią się przerwy, których ręczne usuwanie będzie dość żmudne. W celu automatycznego ponumerowania zawartości zbioru w aplikacji `backend` dodajmy akcję `rozdzial/automat`. W menu głównym dodaj hiperłącze do tej akcji:

```
<li><a href="<?php echo url_for('rozdzial/automat');
➥?>">Automatyczna numeracja <span>&raquo;</span></a></li>
```

W metodzie akcji `rozdzial/automat` należy pobrać z bazy danych wszystkie rozdziały i iteracyjnie ustalić ich numer. Zatem najpierw wywołamy metodę `doSelect()`, a następnie otrzymane wyniki przetworzymy pętlą `foreach`, zgodnie z listingiem 22.19. Numer rozdziału ustalimy na podstawie zmiennej `$k`, która jest zdefiniowana jako zmienna sterująca pętli `foreach`. Po ustaleniu numeru zadania w analogicznej pętli przetwarzamy zadanie z wybranego rozdziału, numerując je na podstawie zmiennej `$i`. Po zakończonej numeracji przechodzimy do strony, której adres jest zapamiętany w sesji, lub do strony `zadanie/index`.

Listing 22.19. *Automatyczna numeracja rozdziałów i zadań*

```
public function executeAutomat(sfWebRequest $request)
{
    $Rozdzials = RozdzialPeer::doSelect(new Criteria());
    foreach ($Rozdzials as $k => $Rozdzial) {
        $Rozdzial->setNumer($k + 1);
        $Rozdzial->save();
        foreach ($Rozdzial->getZadanies() as $i => $Zadanie) {
            $Zadanie->setNumer($i + 1);
            $Zadanie->setSlug('');
            $Zadanie->save();
        }
    }
    $url = $this->getUser()->getAttribute('prevUrl', 'zadanie/index');
    $this->redirect($url);
}
```

Symfony, framework stworzony w języku PHP i mający na celu uproszczenie oraz przyspieszenie tworzenia aplikacji internetowych, znajduje zastosowanie w coraz większej liczbie projektów. Jego wykorzystanie wiąże się ze znacznie efektywniejszym programowaniem, a także pozwala uniknąć wielu błędów i powtarzających się, nużących czynności. Symfony opiera się na modelu MVC i posiada wiele wbudowanych funkcji — między innymi ochronę przed atakami CSRF oraz XSS. Ten framework nie ogranicza się do wykorzystania własnej biblioteki, lecz zapewnia także możliwość integracji z innymi. Jeśli chcesz nauczyć się, jak to działa w praktyce, trzymasz w rękach właściwą pozycję!

Książka „Symfony w przykładach" jest możliwie najbardziej skondensowaną instrukcją obsługi Symfony. Żeby ją zrozumieć, nie musisz dysponować oszałamiającą wiedzą — wystarczą podstawy PHP i XHTML/CSS. Jej autor poprowadzi Cię od najprostszych projektów („Hello world"), przez nieco bardziej zaawansowane zagadnienia, dotyczące zewnętrznych zasobów, połączenia projektu z bazą danych, publikacji projektu na serwerze hostingowym, aż po tworzenie różnego typu paneli administracyjnych. Krótko mówiąc, na samych konkretnych przykładach przejdziesz drogę do stworzenia własnej, niezawodnie działającej aplikacji internetowej.

- Pierwszy projekt w Symfony i praca w środowisku NetBeans
- Wymiana szablonu XHTML/CSS i dołączanie zewnętrznych zasobów
- Hiperłącza i strona błędu 404
- Publikowanie projektu na serwerze hostingowym
- Dostosowywanie klas generowanych przez Propel
- Wyświetlanie danych rekordu i identyfikacja rekordów na podstawie wartości slug
- Artykuły na temat HTML/CSS
- Umieszczanie w bazie danych plików binarnych
- Pliki do pobrania i komponent menu
- Relacje 1:n oraz n:m i widoki częściowe
- Panele administracyjne i tłumaczenie interfejsu witryny
- Zbiór zadań C++
- Administracja kontami użytkowników i generowanie paneli administracyjnych
- Zabezpieczanie paneli administracyjnych protokołem HTTPS

I Ty możesz ułatwić sobie tworzenie doskonałych aplikacji internetowych!

helion.pl
księgarnia internetowa

Cena: 59,00 zł

ISBN 978-83-246-2788-2

9 788324 627882

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

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

Nr katalogowy: 5818

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Informatyka w najlepszym wydaniu