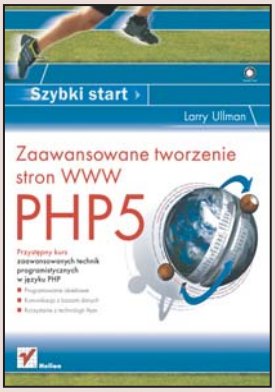
PHP5. Zaawansowane tworzenie stron WWW. Szybki start



Autor: Larry Ullman
Tłumaczenie: Radosław Meryk
ISBN: 78-83-246-1171-3
Tytuł oryginału: PHP 5 Advanced for the World Wide Web: Visual QuickPro Guide (2nd Edition)
Format: B5, stron: około 6002

Kurs zaawansowanych technik programistycznych w języku PHP

- Programowanie obiektowe
- Komunikacja z bazami danych
- Korzystanie z technologii Ajax

PHP to dziś jeden z najpopularniejszych języków programowania stosowanych do tworzenia aplikacji i witryn internetowych. Za jego pomocą powstały setki tysięcy blogów, galerii, portali, sklepów internetowych, serwisów społecznościowych i innych stron WWW. PHP jest prosty, ma czytelną składnię i duże możliwości, a jego najnowsza wersja – PHP5 – umożliwia wykorzystanie wszystkich zalet, jakie płyną z programowania obiektowego. Skrypty PHP łatwo połączyć z bazami danych i stosować razem z innymi technologiami, np. z zyskującym coraz większe uznanie Ajaksem.

Książka „PHP5. Zaawansowane tworzenie stron WWW. Szybki start" to wprowadzenie do rozwiązywania przy użyciu tego języka bardziej złożonych zagadnień programistycznych. Czytając ją, poznasz zasady programowania obiektowego, komunikacji z bazami danych, zabezpieczania aplikacji i poprawy ich wydajności. Dowiesz się, jak projektować złożone aplikacje sieciowe i tworzyć dokumentację projektową. Przeczytasz o interakcji skryptów PHP z serwerem i technikach uruchamiania ich z poziomu wiersza poleceń. Znajdziesz tu także informacje o tworzeniu aplikacji e-commerce. Ostatni rozdział został poświęcony technologii Ajax w skryptach PHP.

- Projektowanie aplikacji i dokumentowanie kodu
- Przechowywanie danych sesji w bazie
- Zabezpieczanie aplikacji WWW
- Tworzenie elementów sklepów internetowych
- Komunikacja z innymi witrynami WWW
- Interakcja z serwerem
- Uruchamianie skryptów PHP z wiersza poleceń
- Korzystanie z repozytorium PEAR
- Technologia Ajax w PHP

Pokonaj kolejny etap w PHP
i zostań ekspertem w dziedzinie programowania

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Projektowanie aplikacji WWW

2

Kariera programisty PHP zazwyczaj rozpoczyna się od tworzenia pojedynczych skryptów, przeznaczonych do wykonania określonego zadania. Na ich bazie zwykle zaczynamy definiować coraz to więcej plików, aż w końcu powstaje aplikacja internetowa. W końcu tworzymy witryny działające na własnym serwerze, a nawet korzystające z wielu serwerów. Niezależnie od stopnia złożoności projektu, nauczenie się nowych i lepszych sposobów tworzenia aplikacji internetowych to istotna część życia programisty języka PHP. Warto o tym pamiętać.

W tym rozdziale skoncentrowano się na tworzeniu aplikacji internetowych wykraczających poza poziom początkującego lub średnio zaawansowanego programisty. Założono, że Czytelnicy potrafią posługiwać się standardowymi mechanizmami stosowanymi w aplikacjach internetowych, takimi jak sesje i szablony. Omówione tematy służą zarówno ugruntowaniu podstaw (które w dużych projektach nabierają większego znaczenia), jak i zmianie sposobu tworzenia stron WWW. Rozdział kończy się omówieniem dwóch rodzajów buforowania usprawniających komunikację pomiędzy częścią kliencką a serwerową aplikacji.

Dokumentowanie kodu

Właściwe dokumentowanie kodu jest tak ważne, że życzyłbym sobie, aby interpreter PHP generował błędy w przypadku napotkania kodu pozbawionego komentarzy. Wiem z mojego wieloletniego doświadczenia w nauczaniu języka PHP i kontaktów z Czytelnikami, że komentarze są bardzo często zaniedbywane. Czasami programiści odkładają wstawianie komentarzy na tzw. „później" (którego zwykle nigdy nie ma). Prawidłowe dokumentowanie kodu jest czymś, co warto robić dla własnego dobra, dla dobra klientów, współpracowników oraz programistów, którym przypadnie w udziale wprowadzanie poprawek do naszej pracy. Trzeba to robić nawet wtedy, a może zwłaszcza wtedy, gdy to my sami będziemy tymi programistami.

Można zaryzykować stwierdzenie, że dokumentacji nigdy za wiele. Warto robić notatki dotyczące funkcji, zmiennych, fragmentów kodu i całych stron. Należy także pamiętać, że dokumentowanie trzeba rozpocząć razem z kodowaniem i kontynuować przez cały czas trwania prac. Powrócenie do kodu w późniejszym czasie w celu wprowadzenia notatek to nie to samo (często się zdarza, że jeśli czegoś nie zrobimy od razu, nie zrobimy tego nigdy). Coś, co wydaje się oczywiste w momencie tworzenia, zaledwie trzy miesiące później okazuje się bardzo zagmatwane.

Komentarze i spacje w przykładach w tej książce

Z powodu ograniczeń wynikających z formatu książkowego skrypty zaprezentowane w tej książce nigdy nie są tak dobrze udokumentowane, jakbym sobie tego życzył (i jak powinienem je udokumentować). Istnieją jednak granice w zapełnianiu cennego miejsca w książce notatkami w stylu:

```
// Opracował: Larry E. Ullman.
```

Zakładam, że Czytelnicy uczą się PHP ode mnie i z tej książki. W związku z tym w kwestii dokumentowania kodu i układu komentarzy zalecam przyjęcie wzoru z tej książki za niezbędne minimum. Objętość komentarzy w tej książce być może nie jest zbyt mała, ale bez szkody dla nikogo mogłaby być dwukrotnie większa.

Aby właściwie udokumentować kod:

1. Na początku skryptu dokładnie opisz wszystkie metawłaściwości.

   Metawłaściwości odnoszą się do informacji na temat pliku: kto go utworzył, kiedy, dlaczego, w ramach jakiej witryny itp. Można tu również umieścić informacje dotyczące autora skryptu, nawet jeśli wydaje się nam, że nikt inny nigdy nie będzie go oglądał.

2. Opisz wszystkie elementy środowiska.

   Jest to dokładniejszy typ dokumentacji od tej, którą stworzyliśmy w punkcie 1. Przez pojęcie „elementy środowiska" rozumiem wszystko to, co musi istnieć lub być wykonane, aby skrypt mógł działać zgodnie z przeznaczeniem. Czy skrypt pobiera informacje z bazy danych? Czy wymaga repozytorium PEAR lub innego kodu zewnętrznego? Czy korzysta z szablonu? Ogólnie rzecz biorąc, należy wymienić wszystkie pliki, z którymi skrypt się komunikuje lub z których korzysta. Jeśli skrypt ma jakieś wymagania dotyczące minimalnej wersji PHP, również to należy zanotować.

3. Dokładnie opisz wszystkie funkcje.

   Funkcje należy opisywać tak samo jak skrypty: ich przeznaczenie, zależności, pobierane wartości, zwracane wyniki itp.

4. Zrób notatki opisujące przeznaczenie zmiennych, jeśli nazwy nie są oczywiste dla początkujących programistów.

   Nazwy zmiennych powinny dobrze opisywać ich przeznaczenie. Nie zawsze jednak można założyć, że będą one czytelne dla wszystkich.

5. Objaśnij przeznaczenie fragmentów kodu.

6. Opisz przeznaczenie instrukcji warunkowych.

   Upewnij się, że powód użycia warunku nie ulega wątpliwości oraz że pożądane wyniki są czytelne. Jeśli w instrukcji warunkowej występuje odwołanie do określonej liczby lub wartości, napisz w komentarzu, do czego służy ta liczba lub wartość.

7. Oznacz zamykające nawiasy klamrowe złożonych funkcji i konstrukcji sterujących (instrukcji warunkowych, pętli itp.).

   W moich skryptach bardzo często występuje zapis następującej postaci:

   ```
   } // Koniec funkcji nazwa_funkcji().
   ```

8. Pamiętaj o zaktualizowaniu komentarzy po wprowadzeniu modyfikacji w kodzie.

   Oto częsty błąd, stwarzający wiele problemów: zaktualizowano kod, ale bez modyfikacji w komentarzach. Kiedy powrócimy do kodu za jakiś czas, przeczytamy, że kod wykonuje operację X, podczas gdy w rzeczywistości wykonuje operację Y. W związku z tym trudno nam będzie stwierdzić, czy działa tak, jak powinien.

Wskazówki

- W modułach wchodzących w skład repozytorium PEAR wykorzystuje się program *phpDocumentor* (*www.phpdoc.org* — rysunek 2.1), który automatycznie generuje dokumentację pakietów. Narzędzie to jest napisane w PHP i przypomina popularny system *JavaDoc* (używany przez programistów Javy). Jego działanie polega na analizowaniu kodu i tworzeniu odpowiednich komentarzy w skryptach. Automatycznie wygenerowanych komentarzy nie należy używać zamiast komentarzy pisanych samodzielnie, ale można je wykorzystać jako standardową dokumentację stron.
- Nieco przestarzały, ale w dalszym ciągu przydatny jest dokument o nazwie *PHP Coding Standard*, opisujący standardy kodowania w PHP. Warto go przeczytać, nawet jeśli nie zawsze będziemy stosować się do opisanych w nim zasad. Niestety, adres URL dokumentu często się zmienia, zatem by go znaleźć, trzeba skorzystać z wyszukiwarki.
- Wiele sugestii dotyczących sposobu wprowadzania komentarzy, ich stylu i struktury można również znaleźć, studiując standardy kodowania dla innych języków.

Styl i struktura kodu

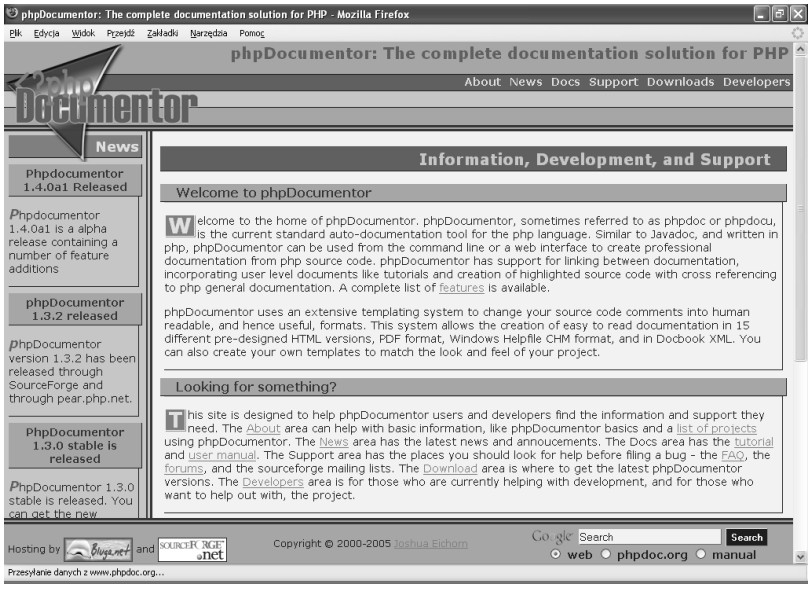
Drugim tematem, który można przypisać do kategorii „warto przestrzegać tych zasad, choć właściwie nie służą niczemu konkretnemu", jest styl kodowania i struktury kodu. Najbardziej podstawowe zasady dotyczące struktury kodu mówią o sposobach stosowania wcięć w kodzie, pozostawiania pustych wierszy, stosowania nawiasów klamrowych itp. Samo stosowanie właściwej składni implikuje określoną strukturę kodu, są jednak dwie „złote" zasady:

- należy zachować spójność kodu,
- należy zadbać o to, by kod był czytelny.

Trzeba bardzo mocno podkreślić, że zachowanie spójności ma znaczenie podstawowe. Niespójność prowadzi do powstania wielu błędów i wymaga dodatkowego czasu, który trzeba poświęcić na ich usuwanie.

Styl w całości zależy od preferencji programisty. Wszelkie rekomendacje, włącznie z tymi, których ja udzielam, są tylko sugestiami. Trzeba jedynie dbać o zachowanie spójności.

Styl to sprawa znacznie bardziej subiektywna. Obejmuje stosowanie konwencji nazewnictwa (stron, zmiennych, klas i funkcji), miejsc, w których można definiować funkcje, sposobów ich organizacji itp.



Rysunek 2.1. *Macierzysta strona programu phpDocumentor*

Aby kod miał właściwą strukturę i styl:

- Zawsze używaj nawiasów klamrowych, nawet jeśli z punktu widzenia poprawności składniowej są one zbędne.

  Wielu programistów ignoruje to zalecenie, jednak stosowanie nawiasów klamrowych w każdym przypadku umożliwia uniknięcie wielu błędów.

- Stosuj wcięcia dla bloków kodu (np. treści instrukcji warunkowych lub instrukcji wewnątrz funkcji) — najlepiej o szerokości jednej tabulacji lub czterech spacji (z punktu widzenia formalnego należy używać spacji zamiast tabulacji, ale programiści często używają tabulacji, ponieważ są one wygodniejsze).

  Właściwe stosowanie wcięć w kodzie bezpośrednio przekłada się na jego czytelność. Wcięcia umożliwiają oznaczenie czytelnych relacji pomiędzy kodem a strukturami sterującymi (pętlami, instrukcjami warunkowymi itp.), funkcjami, klasami i innymi elementami, których częścią jest określony kod.

- Używaj pustych wierszy do wizualnego oddzielania fragmentów kodu.

- Tam, gdzie pozwala na to składnia, używaj spacji pomiędzy słowami, argumentami funkcji, operatorami itp. (ogólnie rzecz biorąc, spacje wewnątrz kodu w PHP nie mają znaczenia).

- Umieszczaj funkcje na początku skryptu lub w osobnym pliku.

- Zawsze używaj formalnych znaczników PHP.

  Ponieważ krótkie znaczniki (`<? ?>`) mają specjalne znaczenie w języku XML, w przypadku korzystania z XML zawsze należy używać formalnych znaczników PHP (`<?php ?>`). Więcej informacji na ten temat można znaleźć w rozdziale 14., „XML i PHP". Zaleca się używanie formalnych znaczników PHP niezależnie od sytuacji, ponieważ jest to najlepszy sposób zapewnienia zgodności kodu pomiędzy różnymi serwerami.

Struktura serwisu

Podczas tworzenia większych aplikacji internetowych istotne są nie tylko struktura kodu i jego dokumentowanie, ale także — w równym stopniu — struktura serwisu, czyli sposób organizacji plików i ich zapisywania na serwerze. Zachowanie właściwej struktury serwisu ma na celu poprawę bezpieczeństwa i ułatwienie administracji, a także poprawę skalowalności, przenośności i łatwości wprowadzania zmian.

Kluczowe znaczenie dla stworzenia właściwej struktury serwisu ma podział kodu i aplikacji na strony i katalogi zgodnie z ich zastosowaniem, przeznaczeniem i funkcją. Wewnątrz głównego katalogu dokumentów witryny, który nosi nazwę *html*, powinien się znaleźć osobny katalog na grafikę (tak przynajmniej czyni większość programistów), osobny dla klas (jeśli używamy technik obiektowych), dla funkcji itp. Co więcej, ze względów bezpieczeństwa zalecam stosowanie własnych nazw folderów. Im trudniej złośliwemu użytkownikowi odgadnąć nazwy folderów i dokumentów, tym lepiej. Jeśli ktoś zastosuje nazwę *admin* dla administracyjnej części witryny, z punktu widzenia bezpieczeństwa nie wyświadczy sobie przysługi.

- Zaleca się stosowanie rozszerzenia *.php* dla plików, których zawartość ma być interpretowana jako kod PHP (w przypadku skryptów włączanych, takich jak klasy i skrypty konfiguracyjne, można stosować inne rozszerzenia).

  Dozwolone rozszerzenia plików określa konfiguracja serwera WWW, jednak w społeczności programistów PHP istnieje trend stosowania rozszerzenia *.php* jako domyślnego.

Modularyzacja witryny WWW

Wiem z doświadczenia, że większość programistów rozpoczyna swoją „krzywą rozwoju" od pisania jednostronicowych aplikacji, które wykonują proste operacje. Aplikacje te przekształcają się w narzędzia złożone z dwóch stron i na koniec w wielostronicowe witryny, w których stosuje się szablony, sesje i (lub) pliki cookie do powiązania ich ze sobą. W przypadku wielu programistów krzywa rozwoju jest właściwie „krzywą dzwonu". W miarę zdobywania doświadczenia programista PHP zaczyna wykonywać tę samą liczbę operacji za pomocą mniejszej liczby stron — na przykład wykorzystuje ten sam skrypt zarówno do wyświetlania, jak i do obsługi formularza. Inni zaawansowani programiści zaczynają tworzyć pojedyncze skrypty, które wykonują mniej działań, bowiem każdy ze skryptów koncentruje się na osobnym zadaniu. Jest to jedna z przesłanek modularyzacji witryny WWW.

Dla potrzeb tego przykładu utworzę witrynę-atrapę (tzn. taką, która nie będzie wykonywała zbyt wielu działań) i podzielę ją na części. Z pokazanego przykładu Czytelnicy dowiedzą się, w jaki sposób wydziela się poszczególne funkcje, jak należy je zorganizować i połączyć ze sobą. Zamiast używania wielu stron (*contact.php*, *about.php*, *index.php* itp.), cała aplikacja będzie działać za pośrednictwem jednej głównej strony. Strona będzie włączała odpowiedni moduł na podstawie wartości przekazywanych za pośrednictwem adresu URL.

Tworzenie pliku konfiguracyjnego

Każda aplikacja internetowa, jaką tworzę, rozpoczyna się od pliku konfiguracyjnego. Pliki konfiguracyjne spełniają kilka funkcji. Cztery najważniejsze to:

- definicja stałych,
- zdefiniowanie parametrów obowiązujących w całej witrynie,
- utworzenie funkcji użytkownika,
- obsługa błędów.

Ogólnie rzecz biorąc, wszystkie dane, do których jest potrzebny dostęp z poziomu każdej strony w witrynie, powinny być zapisane w pliku konfiguracyjnym (jeśli funkcja nie będzie używana przez większość stron witryny, lepiej umieścić ją w osobnym pliku; w ten sposób unikamy dodatkowych kosztów związanych z definiowaniem funkcji na stronach, na których nie będzie używana).

Aby utworzyć plik konfiguracyjny:

1. Utwórz nowy skrypt PHP w edytorze tekstu lub środowisku IDE (skrypt 2.1):

   <?php # Skrypt 2.1 - config.inc.php

Skrypt 2.1. *Plik konfiguracyjny to skrypt o kluczowym znaczeniu. Definiuje stałe obowiązujące w całym skrypcie oraz decyduje o sposobie obsługi błędów*

Listing

```
1   <?php # Skrypt 2.1 - config.inc.php
2
3   /*
4    *Nazwa pliku:  config.inc.php
5    *Utworzony przez:  Larry E. Ullman z firmy DMC Insights, Inc.
6    *Kontakt:  LarryUllman@DMCInsights.com, http://www.dmcinsights.com
7    *Data ostatniej modyfikacji:  8 listopada 2006
8    *
9    *Plik konfiguracyjny spełnia następujące zadania:
10   * - definiuje wszystkie parametry witryny w jednej lokalizacji,
```

Skrypt 2.1. — *ciąg dalszy*

Listing

```
11      * - zawiera adresy URL i URI w formie stałych,
12      * - określa sposób obsługi błędów.
13      */
14
15     # ******************** #
16     # ***** PARAMETRY ***** #
17
18     // Komunikaty o błędach są wysyłane pocztą elektroniczną.
19     $contact_email = 'adres@przyklad.pl';
20
21     // Sprawdzamy, czy skrypt pracuje na serwerze lokalnym,
22     // czy na serwerze produkcyjnym:
23     if (stristr($_SERVER['HTTP_HOST'], 'local') ||
       (substr($_SERVER['HTTP_HOST'], 0, 7) == '192.168')) {
24     $local = TRUE;
25     } else {
26     $local = FALSE;
27     }
28
29     // Określenie lokalizacji plików i adresu URL witryny:
30     // umożliwienie pracy na innych serwerach.
31     if ($local) {
32
33     // Na serwerze lokalnym skrypt zawsze działa w trybie diagnostycznym:
34     $debug = TRUE;
35
36     // Definicja stałych:
37     define ('BASE_URI', '/ścieżka/do/folderu/html/');
38     define ('BASE_URL','http://localhost/katalog/');
39     define ('DB', '/ścieżka/do/mysql.inc.php');
40
41     } else {
42
43     define ('BASE_URI', '/ścieżka/do/produkcyjnego/folderu/html/');
44     define ('BASE_URL','http://www.przyklad.com/');
45     define ('DB', '/ścieżka/do/produkcyjnego/mysql.inc.php');
46
47     }
48
49     /*
50      *Najważniejsze ustawienia…
51      *Zmienna $debug służy do ustawienia obsługi błędów.
52      *W celu debugowania określonej strony należy dodać poniższy kod do strony index.php:
53
54     if ($p == 'thismodule') $debug = TRUE;
55     require_once('./include/config.inc.php');
56
57      *W celu debugowania całej witryny należy ustawić zmienną $debug
58
59     $debug = TRUE;
60
61      *przed następną instrukcją warunkową.
62      */
```

Skrypt 2.1. — *ciąg dalszy*

Skrypt

```
63
64    // Zakładamy, że debugowanie wyłączono.
65    if (!isset($debug)) {
66    $debug = FALSE;
67    }
68
69    # ***** PARAMETRY ***** #
70    # ******************** #
71
72
73    # **************************** #
74    # ***** OBSŁUGA BŁĘDÓW ***** #
75
76    // Utworzenie procedury obsługi błędów.
77    function my_error_handler ($e_number, $e_message, $e_file, $e_line, $e_vars) {
78
79    global $debug, $contact_email;
80
81    // Stworzenie komunikatu o błędzie.
82    $message = "Wystąpił błąd w skrypcie '$e_file' w wierszu $e_line: \n<br
      />$e_message\n<br />";
83
84    // Umieszczenie daty i godziny.
85    $message .= "Data/godzina: " . date('n-j-Y H:i:s') . "\n<br />";
86
87    // Dołączenie zmiennej $e_vars do komunikatu $message.
88    $message .= "<pre>" . print_r ($e_vars, 1) . "</pre>\n<br />";
89
90    if ($debug) { // Wyświetlenie komunikatu o błędzie.
91
92    echo '<p class="error">' . $message . '</p>';
93
94    } else {
95
96    // Zarejestrowanie błędu:
97    error_log ($message, 1, $contact_email); // Wysłanie wiadomości email.
98
99    // Wyświetlenie komunikatu o błędzie tylko wtedy, gdy nie jest to błąd klasy NOTICE lub STRICT.
100   if ( ($e_number != E_NOTICE) && ($e_number < 2048)) {
101   echo '<p class="error">Wystąpił błąd systemowy. Przepraszamy za niedogodności.</p>';
102   }
103
104   } // Koniec instrukcji IF $debug.
105
106   } // Koniec definicji funkcji my_error_handler().
107
108   // Użycie procedury obsługi błędów:
109   set_error_handler ('my_error_handler');
110
111   # ***** OBSŁUGA BŁĘDÓW ***** #
112   # **************************** #
113
114   ?>
```

2. Dodaj komentarze opisujące naturę i przeznaczenie strony:

```
/*
 *  Nazwa pliku:  config.inc.php
 *  Utworzony przez:  Larry E. Ullman z firmy
➥DMC Insights, Inc.
 *  Kontakt:  LarryUllman@DMCInsights.com,
➥http://www.dmcinsights.com
 *  Data ostatniej modyfikacji:  8 listopada 2006
 *
 *  Plik konfiguracyjny spełnia następujące
➥zadania:
 *  - definiuje wszystkie parametry witryny
➥w jednej lokalizacji;
 *  - zawiera adresy URL i URI w formie stałych;
 *  - określa sposób obsługi błędów.
 */
```

Ponieważ plik konfiguracyjny jest wspólny dla całej witryny, powinien być najlepiej udokumentowanym skryptem w całym serwisie.

3. Ustaw adres e-mail, pod który będą przesyłane zgłoszenia błędów:

```
$contact_email = 'adres@przyklad.com';
```

W przypadku „żywych" (tzw. produkcyjnych) witryn preferuję sposób powiadamiania o błędach za pomocą poczty elektronicznej. W związku z tym zadeklarowałem zmienną, która definiuje adres docelowy dla wiadomości. Może to być adres programisty w fazie testowej witryny lub adres administratora po przekazaniu jej do eksploatacji.

4. Sprawdź, czy skrypt działa na serwerze „produkcyjnym" czy testowym.

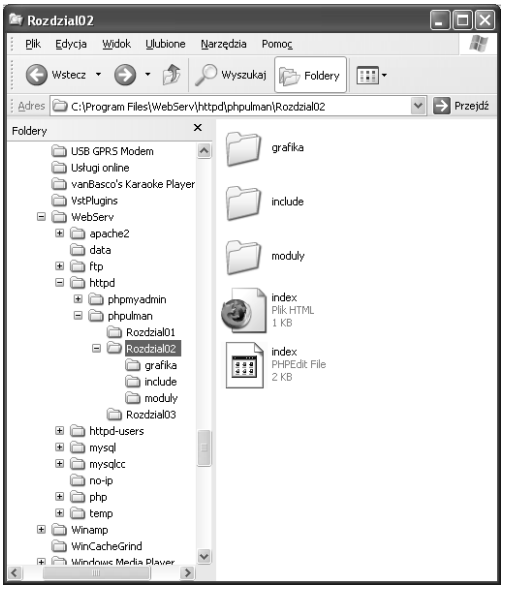
```
if (stristr($_SERVER['HTTP_HOST'],
➥'local') ||
➥(substr($_SERVER['HTTP_HOST'], 0, 7) ==
➥'192.168')) {
    $local = TRUE;
} else {
    $local = FALSE;
}
```

Ja zawsze najpierw tworzę skrypty na serwerze lokalnym, a następnie kopiuję ukończoną stronę na serwer docelowy. Te dwa środowiska charakteryzują się innymi ustawieniami, zatem plik konfiguracyjny powinien potwierdzić, w jakim środowisku pracuje serwis. W celu sprawdzenia, czy witryna pracuje lokalnie, wykorzystałem dwie instrukcje warunkowe sprawdzające parametr `$_SERVER['HTTP_HOST']`. Jeśli zmienna zawiera słowo *local* (na przykład *http://localhost* lub *http://powerbook.local*) albo jeśli adres IP witryny zaczyna się od *192.168* (co wskazuje na sieć lokalną), wówczas ustawiamy zmienną `$local` na wartość `TRUE`. W przeciwnym przypadku ustawiamy ją na `FALSE`.

5. Ustaw stałe specyficzne dla serwera.

```
if ($local) {
    $debug = TRUE;
    define ('BASE_URI',
➥'/ścieżka/do/folderu/html/'); define
➥('BASE_URL','http://localhost/
➥katalog/');
    define ('DB',
➥'/ścieżka/do/mysql.inc.php');
} else {
    define ('BASE_URI', '/ścieżka/
➥do/produkcyjnego/folderu/html/');
    define('BASE_URL',
➥'http://www.przyklad.com/');
    define ('DB', '/ścieżka/
➥do/produkcyjnego/mysql.inc.php');
}
```

W aplikacjach internetowych, które tworzę, zawsze używam tych trzech stałych. Stała `BASE_URI` zawiera bezwzględną ścieżkę do głównego folderu witryny na serwerze (rysunek 2.2). Dzięki tej stałej można z łatwością korzystać z bezwzględnych adresów URL w przypadku, gdy skrypt włącza plik. Stała `BASE_URL` definiuje nazwę hosta i katalog (jeśli serwis działa w podkatalogu). Na serwerze testowym może to być na przykład *http://localhost/r02/*. Na koniec ustawiliśmy zmienną `DB` zawierającą bezwzględną ścieżkę do pliku zawierającego parametry połączeń z bazą danych. Ze względów bezpieczeństwa plik ten należy zapisać poza głównym katalogiem dokumentów witryny. Zwróćmy uwagę na to, że dla każdej stałej istnieją dwie reprezentacje: jedna dla serwera testowego i druga dla produkcyjnego. W przypadku serwera testowego (zmienna `$local` ma wartość `TRUE`) dodatkowo włączyłem debugowanie. Wkrótce opiszę to dokładniej.



Rysunek 2.2. *Główna strona witryny jest zapisana w głównym folderze serwisu. Wewnątrz tego folderu mogą być zdefiniowane podfoldery, na przykład grafika, include lub moduły*

6. Ustawiamy poziom debugowania.

```
if (!isset($debug)) {
    $debug = FALSE;
}
```

Do określenia sposobu obsługi błędów użyłem zmiennej `$debug`. Jeśli witryna działa lokalnie, zmienna `$debug` ma wartość `TRUE`. Aby można było debugować skrypt w serwisie przekazanym do eksploatacji, trzeba użyć następującego wiersza:

```
$debug = TRUE;
```

przed włączeniem pliku konfiguracyjnego. We wszystkich pozostałych przypadkach debugowanie jest wyłączone.

7. Rozpocznij funkcję obsługi błędów.

```
function my_error_handler ($e_number,
➥$e_message, $e_file, $e_line,
➥$e_vars) {
    global $debug, $contact_email;
```

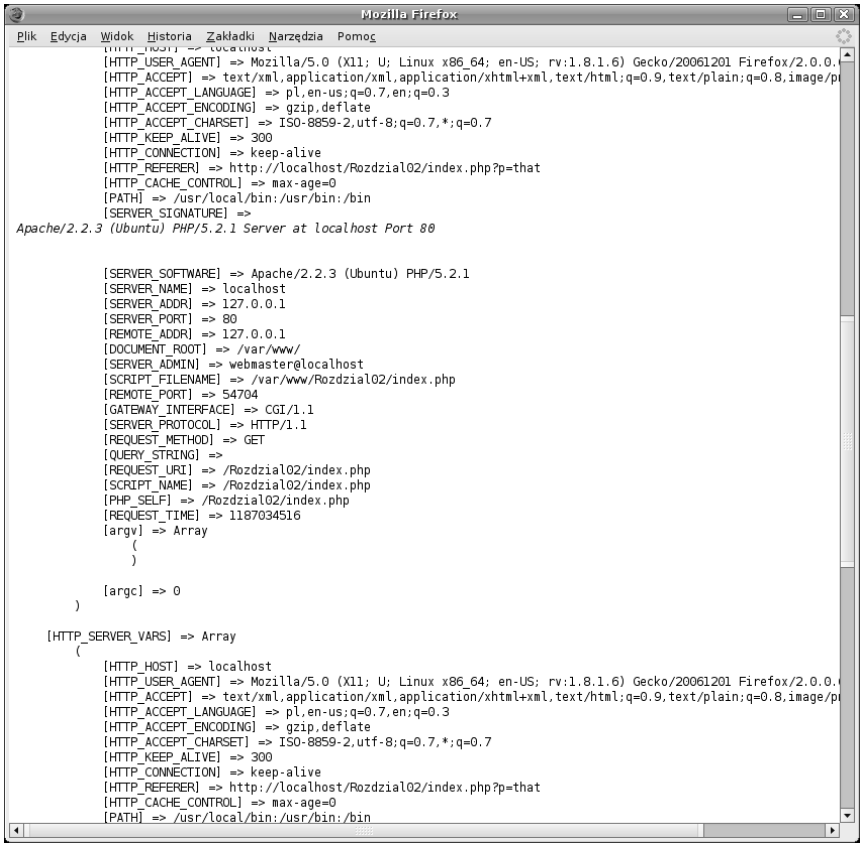
PHP umożliwia zdefiniowanie własnej funkcji obsługi błędów, którą można wykorzystywać zamiast funkcji wbudowanej. Więcej informacji na temat tego procesu, a także na temat składni funkcji można znaleźć w podręczniku PHP online lub w mojej książce *PHP i MySQL. Dynamiczne strony WWW. Szybki start* (Helion 2006).

W funkcji wykorzystywane będą dwie zmienne globalne.

8. Skonstruuj komunikat o błędzie:

```
$message = "Wystąpił błąd w skrypcie
➥'$e_file' w wierszu $e_line: \n<br
➥/>$e_message\n<br />";
$message .= "Data/godzina: " .
➥date('n- j-Y H:i:s') . "\n<br />";
$message .= "<pre>" . print_r ($e_vars, 1) .
➥"</pre>\n<br />";
```

W celu ułatwienia diagnozowania błędów komunikat o błędzie powinien być opisowy. Powinien zawierać co najmniej nazwę pliku, w którym wystąpił błąd, oraz numer wiersza. Następnie w komunikacie powinna znaleźć się data i godzina wystąpienia błędu oraz wszystkie występujące zmienne. To sporo danych (rysunek 2.3), ale są one potrzebne, by można było szybko zlokalizować i usunąć błąd.



Rysunek 2.3. *Oto niektóre informacje zgłaszane w przypadku wystąpienia błędu*

9. Jeśli włączono debugowanie, wyświetlamy komunikat o błędzie:

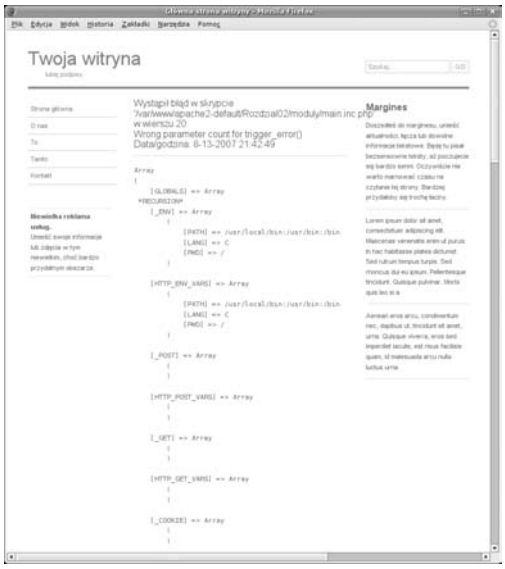
```
    if ($debug) {
        echo '<p class="error">' . $message .
➥'</p>';
```

Jeśli włączono debugowanie, w przeglądarce WWW zostanie wyświetlony pełny komunikat o błędzie (rysunek 2.4). Jest to doskonały mechanizm w fazie projektowania witryny i olbrzymia luka w systemie bezpieczeństwa dla serwisu w eksploatacji. Kod należy odpowiednio zmodyfikować, tak aby pasował do środowiska wybranego serwisu.

10. Jeśli debugowanie wyłączono, przesyłamy komunikat o błędzie pocztą elektroniczną i wyświetlamy domyślny komunikat:

```
    } else {
        error_log ($message, 1,
➥$contact_email);
        if ( ($e_number != E_NOTICE) &&
➥($e_number < 2048)) {
            echo '<p class="error">Wystąpił
➥błąd systemowy. Przepraszamy za
➥niedogodności.</p>';
        }
    } // Koniec instrukcji IF $debug.
```

W przypadku witryny przekazanej już do eksploatacji nie należy wyświetlać szczegółowego komunikatu o błędzie (chyba że dla strony czasowo włączono tryb diagnostyczny). Można to zrobić za pomocą funkcji `error_log()`, jeśli przekaże się do niej liczbę 1 jako drugi argument. Użytkownik powinien jednak wiedzieć, że coś nie działa tak jak powinno, dlatego wyświetlamy domyślny komunikat (rysunek 2.5). Jeśli błąd jest klasy `NOTICE` lub `STRICT` (o numerze 2048), nie należy wyświetlać komunikatu o błędzie, ponieważ błąd prawdopodobnie nie przeszkadza w działaniu strony.



Rysunek 2.4. *Wyświetlanie komunikatów o błędach podczas diagnostycznego uruchamiania strony*

Rysunek 2.5. *W serwisach przekazanych do eksploatacji błędy należy obsługiwać bardziej dyskretnie (i bezpieczniej)*

11. Zakończ funkcję, zainicjuj używanie zdefiniowanej procedury obsługi błędów i zakończ stronę:

```
} // Koniec definicji funkcji my_error_handler().
set_error_handler('my_err_handler');
?>
```

12. Zapisz plik jako *config.inc.php* i umieść na serwerze WWW (w podfolderze *include*).

Układ katalogów dla przykładowego serwisu pokazano na rysunku 2.2.

Tworzenie pliku konfiguracji bazy danych

W tym przykładzie nie utworzyłem pliku konfiguracji bazy danych, ponieważ witryna nie korzysta z baz danych. Gdyby była potrzebna baza danych, trzeba by napisać plik *mysql.inc.php* (lub *postgresql.inc.php*, *oracle.inc.php* czy jakiś inny), którego zadaniem byłoby nawiązanie połączenia z bazą danych. W takim pliku definiuje się również funkcje wykonujące operacje na bazie danych aplikacji.

Plik można zapisać w katalogu *include*, choć najlepiej zapisać go poza katalogiem dokumentów serwisu WWW. W pliku *config.inc.php* zdefiniowano stałą `DB`, która określa bezwzględną ścieżkę do tego pliku na serwerze.

Jeśli na jakiejś stronie jest potrzebne połączenie z bazą danych, można włączyć potrzebny plik za pomocą następującej instrukcji:

```
require_once(DB);
```

Ponieważ stała `DB` reprezentuje bezwzględną ścieżkę do pliku, nie ma znaczenia, czy włączany skrypt znajduje się w głównym folderze, czy w podkatalogu.

Tworzenie szablonu HTML

Niemal we wszystkich większych aplikacjach stosuje się szablony HTML. Można w tym celu skorzystać z technologii Smarty (*http://smarty.php.net*) lub wielu innych systemów obsługi szablonów. Ja jednak preferuję użycie dwóch prostych plików: nagłówka, który zawiera wszystko, co ma się znaleźć na stronie do miejsca, w którym występuje specyficzna treść, oraz stopki zawierającej pozostałą część strony (rysunek 2.6).

Dla potrzeb tego szablonu skorzystałem z projektu, który znalazłem w witrynie Open Source Web Design (*www.oswd.org*) — doskonałego źródła materiałów do projektowania aplikacji internetowych. Autorem tego projektu (rysunek 2.7) jest Anthony Yeung (*www.smallpark.org*). Użyłem go grzecznościowo, dzięki uprzejmości autora.

Aby stworzyć strony korzystające z szablonów:

1. Zaprojektuj stronę HTML w edytorze tekstowym lub WYSIWYG.

   Aby przystąpić do tworzenia szablonu witryny WWW, należy zaprojektować standardową stronę HTML niezależną od kodu PHP. Dla potrzeb tego przykładu, jak już wspominałem, skorzystam z projektu *Leaves CSS* (rysunek 2.7).

   Uwaga! W celu zaoszczędzenia miejsca plik CSS dla tego przykładu (decydujący o układzie strony) nie został dołączony do książki. Można go pobrać ze strony WWW towarzyszącej książce (*www.DMCInsights.com/phpvqp2/* — patrz strona *Extras*). Można też uruchomić przykład bez arkusza CSS (szablon będzie działał, tylko będzie mniej estetyczny).

2. Skopiuj do nowego dokumentu treść szablonu od pierwszego wiersza do miejsca, w którym zaczyna się treść specyficzna dla strony (skrypt 2.2). Tak więc, skopiuj tekst począwszy od:

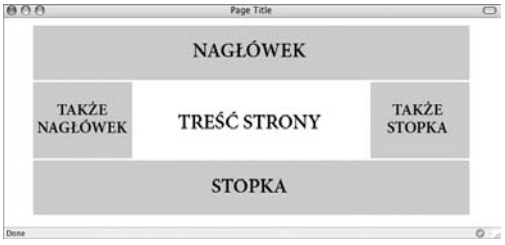
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
➥Transitional//EN" "http://www.w3.org/
➥TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/
➥1999/xhtml">
<head>
```

aż do:

```
<div id="content">
<!-- Koniec nagłówka. -->
```

W pierwszym pliku są początkowe znaczniki HTML (począwszy od znacznika `DOCTYPE`, przez nagłówek, aż do początku treści strony). Jest w nim również kod tworzący kolumnę łączy po lewej stronie okna przeglądarki oraz margines po prawej (rysunek 2.7). W tym kroku pominąłem duży fragment kodu HTML. Kompletny kod zamieszczono w skrypcie 2.2. Można go również pobrać ze strony WWW towarzyszącej książce.



Rysunek 2.6. *Bardzo prosty przykład szablonu witryny: obszar treści specyficznej dla strony został otoczony dwoma plikami włączanymi*

Rysunek 2.7. *Szablon używany dla wszystkich stron w tej witrynie*

Skrypt 2.2. *Plik nagłówka rozpoczyna szablon HTML Zawiera również plik CSS oraz wykorzystuje zmienną PHP w celu zdefiniowania tytułu okna przeglądarki*

Skrypt

```
<?php # Skrypt 2.2 - header.html

// Ta strona rozpoczyna nagłówek HTML witryny.

// Sprawdzenie wartości zmiennej $page_title:
if (!isset($page_title)) $page_title = 'Domyślny tytuł strony';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//PL"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250" />
<title><?php echo $page_title; ?></title>
<link href="./include/style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container">
    <div id="header">
      <h1>Twoja witryna</h1>
      <p>lubię podpisy</p>
          <form name="form1" id="form1" method="get" action="index.php">
          <input type="text" name="terms" value="Szukaj..." />
          <input type="hidden" name="p" value="search" />
          <input class=" button" type="submit" name="Wyślij" value="GO" />
      </form>
    </div>

    <div id="navigation">
        <ul id="navlist">
            <li><a href="index.php">Strona główna</a></li>
            <li><a href="index.php?p=about">O nas</a></li>
            <li><a href="index.php?p=this">To</a></li>
            <li><a href="index.php?p=that">Tamto</a></li>
            <li><a href="index.php?p=contact">Kontakt</a></li>
<li><p><strong>Niewielka reklama usług.</strong><br/>Umieść swoje informacje lub zdjęcia
w tym niewielkim, choć bardzo przydatnym obszarze. </p></li>
        </ul>

  </div>
      <div id="sidebar">
        <h2>Margines</h2>
        <p>Doszedłeś do marginesu, umieść aktualności, łącza lub dowolne informacje
tekstowe. Będę tu pisał bezsensowne teksty, aż poczujecie się bardzo senni. Oczywiście nie
warto marnować czasu na czytanie tej strony. Bardziej przydałoby się trochę łaciny.
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas venenatis
enim ut purus. In hac habitasse platea dictumst. Sed rutrum tempus turpis. Sed rhoncus dui
eu ipsum. Pellentesque tincidunt. Quisque pulvinar. Morbi quis leo sit amet neque tempo
fringilla.Pellentesque faucibus metus vitae erat. Quisque a urna ut sapien accumsan ornare
Nulla porta pretium eros. Fusce velit erat, accumsan pellentesque, porttitor eu, commodo
quis augue. <a href="#">Fusce convallis ipsum eget felis</a>. </p>
```

3. Zmodyfikuj wiersz zawierający tytuł strony w taki sposób, by przyjął następującą postać:

```
<title><?php echo $page_title; ?></title>
```

Chciałbym, aby tytuł strony (wyświetlający się na początku strony w przeglądarce WWW — na rysunku 2.7 wyświetla się jako *Tytuł strony*) można było definiować osobno dla każdej ze stron. W tym celu ustawię zmienną, której wartość wyświetli interpreter PHP.

4. Przed kodem HTML strony utwórz sekcję kodu PHP, która sprawdza wartość zmiennej `$page_title`.

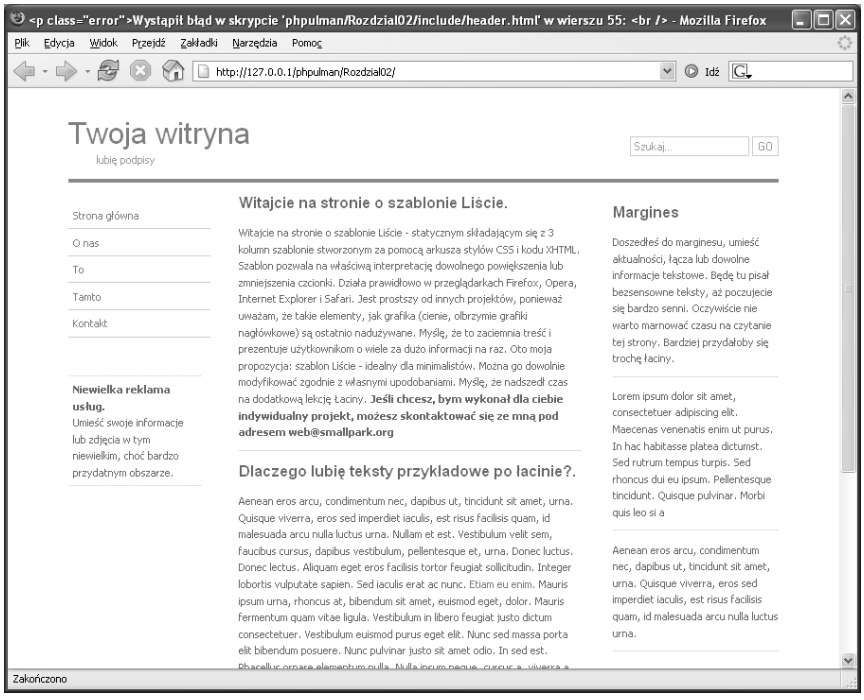
```
<?php # Script 2.2 - header.html
if (!isset($page_title)) $page_title =
➥'Domyślny tytuł strony';
?>
```

Na wypadek, gdyby skrypt PHP włączył plik nagłówkowy bez ustawienia zmiennej `$page_title`, w tym fragmencie kodu PHP zadeklarowano domyślny tytuł strony (oczywiście można go zmienić na bardziej opisowy). Gdyby nie było tego fragmentu, a opcja zgłaszania błędów byłaby włączona, tytuł strony w przeglądarce mógłby wyświetlić się w postaci pokazanej na rysunku 2.8.

Skrypt 2.2. *— ciąg dalszy*

Skrypt

```
42          <p>Aenean eros arcu,
        condimentum nec, dapibus ut,
        tincidunt sit amet, urna. Quisque
        viverra, eros sed imperdiet iaculis,
        est risus facilisis quam, id
        malesuada arcu nulla luctus urna.
        </p></div>
43
44              <div id="content">
45      <!-- Koniec nagłówka. -->
```



Rysunek 2.8. *Należy sprawdzić, czy ustawiono zmienną $page_title. Jeśli nie ustawiono wartości tej zmiennej, system zgłaszania błędów zamiast tytułu wyświetli szczegółowy komunikat o błędzie*

Skrypt 2.3. *Plik stopki uzupełnia szablon HTML*

Skrypt

```
1   <!-- # Skrypt 2.3 - footer.html -->
2     </div>
3
4       <div id="footer">
5       <p><a
    href="http://validator.w3.org/
    check?uri=referer"
    target="_blank">Prawidłowy kod
6     XHTML</a> | <a
    href="http://jigsaw.w3.org/
    css-validator/check/referer"
    target="_blank">Prawidłowy kod
7     CSS</a> | Copyright &copy; Tu
    umieść swoje nazwisko |
    Zaprojektowała firma <a href="http://
    smallpark.org">SmallPark</a></p>
8     </div>
9     </div>
10  </body>
11  </html>
```

5. Zapisz plik jako *header.html*.

Dla plików włączanych można używać dowolnych rozszerzeń. Niektórzy programiści stosują rozszerzenie *.inc*, które wskazuje, że plik jest włączany. W tym przypadku można także użyć rozszerzenia *.inc.html*, które wskazuje, że plik jest włączany i zawiera kod HTML (dla odróżnienia od plików złożonych wyłącznie z kodu PHP).

6. Skopiuj z pliku szablonu całą treść, począwszy od końca zawartości specyficznej dla strony do końca strony, i wklej ją do nowego pliku (skrypt 2.3).

```
<!-- # Skrypt 2.3 - footer.html -->
  </div>
<div id="footer">
    <p><a href="http://validator.w3.org/
➥check?uri=referer"
➥target="_blank">Prawidłowy kod
  XHTML</a> | <a href="http://
➥jigsaw.w3.org/css-validator/check/
➥referer" target="_blank">Prawidłowy kod
  CSS</a> | Copyright &copy; Tu umieść swoje
➥nazwisko | Zaprojektowała firma <a
➥href="http://smallpark.org">SmallPark</
➥a></p>
  </div>
</div>
</body>
</html>
```

W pliku stopki zamieszczono pozostałe elementy formatowania dla treści strony, włącznie z jej stopką, oraz znaczniki zamykające dokument HTML.

7. Zapisz plik jako *footer.html*.

8. Umieść oba pliki w katalogu *include* serwera WWW.

Tworzenie strony głównej

Strona *index.php* to główny skrypt w aplikacji podzielonej na moduły. W rzeczywistości jest to jedyna strona, która obowiązkowo musi być załadowana w przeglądarce WWW. Strona *index.php* spełnia jeden cel: scala odpowiednie fragmenty, które razem tworzą kompletną stronę WWW. Proces ten może obejmować następujące działania:

- włączenie pliku konfiguracyjnego;
- włączenie pliku parametrów połączenia z bazą danych;
- zastosowanie szablonu HTML;
- załadowanie właściwego modułu z zawartością.

Pamiętając o tym, pozostaje jedynie opracować kod i upewnić się, że właściwie go obsłużono. Proces ten objaśnię szczegółowo w poniższej procedurze.

Aby stworzyć główną stronę witryny:

1. Utwórz nowy skrypt PHP w edytorze tekstu lub środowisku IDE (skrypt 2.4):

   ```
   <?php # Skrypt 2.4 - index.php
   ```

2. Dołącz plik konfiguracyjny.

   ```
   require_once('./include/config.inc.php');
   ```

   W pliku konfiguracyjnym znajdują się definicje wielu ważnych ustawień, dlatego należy go dołączyć w pierwszej kolejności.

Skrypt 2.4. *Strona index.php to skrypt, za pośrednictwem którego wykonywane są wszystkie operacje w witrynie. Określa moduły, które należy dołączyć, ładuje plik konfiguracyjny i łączy treść strony z szablonem HTML*

Skrypt

```
1   <?php # Skrypt 2.4 - index.php
2
3   /*
4    *  To jest główna strona.
5    *  Skrypt ten dołącza pliki konfiguracyjne,
6    *  szablony i wszystkie moduły zawierające treść strony.
7    */
8
9   // Załadowanie pliku konfiguracyjnego przed kodem PHP:
10  require_once('./include/config.inc.php');
11
12  // Sprawdzenie, jaką stronę należy wyświetlić:
13  if (isset($_GET['p'])) {
14      $p = $_GET['p'];
15  } elseif (isset($_POST['p'])) { // Formularze
16      $p = $_POST['p'];
17  } else {
18      $p = NULL;
19  }
20
21  // Sprawdzenie, jaką stronę należy wyświetlić:
22  switch ($p) {
23
```

Skrypt 2.4. — *ciąg dalszy*

Skrypt

```
24          case 'about':
25              $page = 'about.inc.php';
26              $page_title = 'Informacje o tej witrynie';
27              break;
28
29          case 'this':
30              $page = 'this.inc.php';
31              $page_title = 'To jest inna strona.';
32              break;
33
34          case 'that':
35              $page = 'that.inc.php';
36              $page_title = 'A to jeszcze inna strona.';
37              break;
38
39          case 'contact':
40              $page = 'contact.inc.php';
41              $page_title = 'Skontaktuj się z nami';
42              break;
43
44          case 'search':
45              $page = 'search.inc.php';
46              $page_title = 'Wyniki wyszukiwania';
47              break;
48
49          // Domyślnie skrypt włącza stronę główną.
50          default:
51              $page = 'main.inc.php';
52              $page_title = 'Główna strona witryny';
53              break;
54
55      } // Koniec głównej instrukcji switch.
56
57      // Sprawdzenie, czy plik istnieje:
58      if (!file_exists('./moduly/' . $page)) {
59          $page = 'main.inc.php';
60              $page_title = 'Główna strona witryny';
61      }
62
63      // Włączenie pliku nagłówkowego:
64      include_once ('./include/header.html');
65
66      // Włączenie modułu z treścią:
67      // zmienna $page jest określona na podstawie powyższej instrukcji switch.
68      include ('./moduly/' . $page);
69
70      // Włączenie pliku stopki w celu uzupełnienia szablonu:
71      include_once ('./include/footer.html');
72
73      ?>
```

3. Sprawdź poprawność wyświetlanej strony.

```
if (isset($_GET['p'])) {
    $p = $_GET['p'];
} elseif (isset($_POST['p'])) { // Formularze
    $p = $_POST['p'];
} else {
    $p = NULL;
}
```

Zawartość, która zostanie wyświetlona na stronie, zależy od wartości parametru przesłanego do strony. W przypadku kliknięcia łącza wartości są przesyłane za pomocą adresu URL. W większości formularzy wartości są przesyłane za pomocą tablicy superglobalnej `$_POST`. Dla pozostałych przypadków należy ustawić zmienną `$p` na wartość `NULL`.

4. Rozpocznij instrukcję `switch`, która określa tytuł strony i plik.

```
switch ($p) {
case 'about':
        $page = 'about.inc.php';
        $page_title = 'Informacje o tej
➥witrynie';
        break;
```

Każdy moduł ma nazwę *nazwa.inc.php*. Użyte rozszerzenie to mój sposób oznaczenia, że jest to zarówno skrypt PHP, jak i plik włączany. Ze względu na sposób interpretowania rozszerzeń w komputerach, znaczenie ma tylko końcowe rozszerzenie, tzn. przy próbie bezpośredniego uruchomienia pliku zostałby on zinterpretowany jako skrypt PHP.

Dla każdego modułu ustawiono również tytuł strony (który zostanie wyświetlony w przeglądarce).

5. Uzupełnij instrukcję `switch`.

```
    case 'this':
        $page = 'this.inc.php';
        $page_title = 'To jest inna strona.';
        break;

    case 'that':
        $page = 'that.inc.php';
        $page_title = 'A to jeszcze
➥inna strona.';
        break;

case 'contact':
        $page = 'contact.inc.php';
        $page_title = 'Skontaktuj się
➥z nami';
        break;

    case 'search':
        $page = 'search.inc.php';
        $page_title = 'Wyniki
➥wyszukiwania';
        break;

    default:
        $page = 'main.inc.php';
        $page_title = 'Główna strona
➥witryny';
        break;
} // Koniec głównej instrukcji switch.
```

Dla każdego dostępnego modułu treści użyto osobnego warunku `case` instrukcji `switch`. Ze względów bezpieczeństwa domyślny warunek `case` ma znaczenie kluczowe. Jeśli zmiennej `$p` nie przypisano wartości lub jeśli jej wartość jest niewłaściwa (tzn. nie istnieje dla niej właściwy przypadek `case`) — skrypt ładuje plik *main.inc.php*. Jest to niezbędna operacja ze względów bezpieczeństwa. Złośliwy użytkownik może zauważyć, że w witrynie są stosowane adresy URL postaci *index.php?p=contact* i spróbować użyć adresu postaci *index.php?p=/sciezka/do/jakiegos/skryptu*. W takim przypadku skrypt włączy główną stronę witryny.

6. Sprawdź, czy istnieje plik modułu.

```
if (!file_exists('./moduly/' . $page)) {
    $page = 'main.inc.php';
    $page_title = 'Główna strona witryny';
}
```

Umieszczenie powyższej instrukcji nie jest absolutnie konieczne, jeśli dla każdego przypadku `case` istnieje właściwy moduł. Użycie tej instrukcji tworzy jednak dodatkową warstwę bezpieczeństwa.

7. Dołącz plik nagłówkowy.

```
include_once ('./include/header.html');
```

W tym pliku znajduje się początek szablonu HTML.

8. Włącz moduł.

```
include ('./moduly/' . $page);
```

Spowoduje to wstawienie odpowiedniej zawartości.

9. Dołącz plik stopki:

```
include_once ('./include/footer.html');
```

Ten plik uzupełnia szablon HTML.

10. Zakończ stronę.

```
?>
```

11. Zapisz plik jako *index.php* i umieść go w katalogu dokumentów na serwerze WWW.

Skryptu nie można przetestować do momentu utworzenia przynajmniej jednego modułu zawartości (co najmniej *main.inc.php*).

Wskazówka

- Instrukcja `switch`, która sprawdza poprawność wartości zmiennej `$p`, jest bardzo ważnym zabezpieczeniem. Inne zabezpieczenie polega na użyciu osobnej zmiennej jako nazwy włączanego pliku (tzn. `$page`). Poniższy kod stwarzałby bardzo duże zagrożenie:

```
include ($_GET['p']);
```

Moja przygoda związana z zabezpieczeniami

Latem 2006 roku usprawniłem witrynę WWW mojej firmy (*www.DMCInsights.com*), wprowadzając organizację modułową. Już pierwszej nocy po opublikowaniu nowej wersji witryny ktoś próbował włamać się na serwer, zmieniając adres URL postaci *about.php?i=phpmysql2* na wartość *about.php?i=http://jakisserwis.com/plik.txt*. Skrypt *plik.txt* na tym serwerze zawierał kod PHP wyświetlający zawartość mojego serwera. Gdyby hakerowi udało się go uruchomić, bezpieczeństwo mojej witryny byłoby zagrożone.

Próba nie powiodła się z dwóch powodów. Po pierwsze, sprawdziłem poprawność elementu `$_GET['i']`, porównując go ze zbiorem dopuszczalnych wartości. Po drugie, zachowałem ostrożność podczas operacji włączania plików. Równie ważny był jednak mechanizm zgłaszania błędów zaimplementowany w witrynie. Ponieważ witryna była przekazana do eksploatacji, użytkownik nie uzyskał żadnych interesujących informacji przy próbie włączenia niewłaściwego pliku, ale ja otrzymałem informację o próbie włamania pocztą elektroniczną.

Utworzenie modułów treści

Teraz, kiedy wykonaliśmy wszystkie czynności wstępne — napisaliśmy plik konfiguracyjny, szablon i skrypt *index.php*, nadszedł czas, by przystąpić do tworzenia modułów treści. W takim systemie zaimplementowanie modułu treści jest niezwykle proste. Pliki z treścią nie muszą włączać plików konfiguracyjnych bądź szablonów, ponieważ zrobiono to już w głównym skrypcie. Ponieważ wszystkie moduły treści są włączane, mogą one zawierać zarówno kod HTML, jak i PHP.

Jest jednak jedna pułapka: moduły nie powinny być ładowane bezpośrednio. Gdyby ktoś spróbował bezpośrednio wywołać skrypt *main.inc.php* (lub dowolny inny moduł) w przeglądarce WWW, zobaczyłby wynik bez szablonu HTML (rysunek 2.9), bez odpowiedniej obsługi błędów oraz potencjalnie bez kodu nawiązującego połączenie z bazą danych. W związku z tym w każdym z modułów powinien znaleźć się kod, który w przypadku próby bezpośredniego wywołania przekieruje użytkownika do właściwej strony.

Aby stworzyć główny moduł treści:

1. Utwórz nowy skrypt PHP w edytorze tekstu lub w środowisku IDE (skrypt 2.5):

```
<?php # Skrypt 2.5 - main.inc.php
```


Rysunek 2.9. *Należy tak zaprojektować witrynę, aby nie było możliwości bezpośredniego dostępu do modułów z treścią za pomocą adresu URL. W takim przypadku strona wyświetlałaby się, między innymi, bez szablonu HTML*

Skrypt 2.5. *W pierwszym module treści znajduje się kod HTML strony głównej. Wykorzystano również kod PHP, który przekierowuje użytkownika w przypadku próby bezpośredniego dostępu do skryptu w przeglądarce*

Skrypt

```
<?php # Skrypt 2.5 - main.inc.php

/*
 *  To jest główny moduł treści strony.
 *  Ta strona jest dołączana przez skrypt index.php.
 */

// Przekierowanie w przypadku próby bezpośredniego dostępu:
if (!defined('BASE_URL')) {

    // Potrzebna jest stała BASE_URL zdefiniowana w pliku konfiguracyjnym:
    require_once ('../include/config.inc.php');

    // Przekierowanie do strony głównej:
    $url = BASE_URL . 'index.php';
    header ("Location: $url");
    exit;

} // Koniec instrukcji IF !defined().
?>

      <h2>Witajcie na stronie o szablonie Liście.</h2>
      <p>Witajcie na stronie o szablonie Liście - statycznym składającym się z 3 kolumn
szablonie stworzonym za pomocą arkusza stylów CSS i kodu XHTML. Szablon pozwala na właściwą
interpretację dowolnego powiększenia lub zmniejszenia czcionki. Działa prawidłowo
w przeglądarkach Firefox, Opera, Internet Explorer i Safari. Jest prostszy od innych
projektów, ponieważ uważam, że takie elementy, jak grafika (cienie, olbrzymie grafiki
nagłówkowe), są ostatnio nadużywane. Myślę, że to zaciemnia treść i prezentuje użytkownikom
o wiele za dużo informacji naraz. Oto moja propozycja: szablon Liście - idealny dla
minimalistów. Można go dowolnie modyfikować zgodnie z własnymi upodobaniami. Myślę, że
nadszedł czas na dodatkową lekcję łaciny. <strong>Jeśli chcesz, bym wykonał dla ciebie
indywidualny projekt, możesz skontaktować się ze mną pod adresem web@smallpark.org
</strong></p>
      <h2>Dlaczego lubię teksty przykładowe po łacinie?. </h2>
        <p>Aenean eros arcu, condimentum nec, dapibus ut, tincidunt sit amet, urna. Quisque
viverra, eros sed imperdiet iaculis, est risus facilisis quam, id malesuada arcu nulla
luctus urna. Nullam et est. Vestibulum velit sem, faucibus cursus, dapibus vestibulum,
pellentesque et, urna. Donec luctus. Donec lectus. Aliquam eget eros facilisis tortor
feugiat sollicitudin. Integer lobortis vulputate sapien. Sed iaculis erat ac nunc. <a
href="#">Etiam eu enim.</a> Mauris ipsum urna, rhoncus at, bibendum sit amet, euismod eget,
dolor. Mauris fermentum quam vitae ligula. Vestibulum in libero feugiat justo dictum
consectetuer. Vestibulum euismod purus eget elit. Nunc sed massa porta elit bibendum
posuere. Nunc pulvinar justo sit amet odio. In sed est. Phasellus ornare elementum nulla.
Nulla ipsum neque, cursus a, viverra a, imperdiet at, enim. Quisque facilisis, diam sed
accumsan suscipit, odio arcu hendrerit dolor, quis aliquet massa nulla nec sem. </p>
      <h2>Po prostu je lubię. </h2>
      <p><a href="#">Proin sagittis leo in diam</a>. Vestibulum vestibulum orci vel libero.
Cras molestie pede quis odio. Phasellus tempus dolor eu risus. Aenean tellus tortor,
dignissim sit amet, tempus eu, eleifend porttitor, ipsum. Fusce diam. Suspendisse sed nunc
quis odio aliquet feugiat. Pellentesque sapien. Phasellus sed lorem eu augue luctus
commodo. Nullam interdum convallis nunc. Fusce varius. Ut egestas. Fusce interdum iaculis
pede. Sed vehicula vestibulum odio. <a href="#">Donec id diam. </a></p>
```

2. Sprawdź, czy nie nastąpiła próba bezpośredniego dostępu do skryptu:

```
if (!defined('BASE_URL')) { {
```

Jest wiele elementów, które można sprawdzić w tym celu — na przykład czy ustawiono zmienne $page lub $p. Gdyby jednak włączono parametr `register globals`, a użytkownik spróbowałby wejść na stronę *main.inc.php?p=true*, taki test nie powiódłby się. W związku z tym lepiej sprawdzić, czy zdefiniowano stałą. Tworzy się ją w pliku konfiguracyjnym, który należy załadować w głównym skrypcie w pierwszej kolejności, jeszcze przed załadowaniem modułu treści.

3. Przekieruj użytkownika.

```
require_once ('../include/config.inc.php');
$url = BASE_URL . 'index.php';
header ("Location: $url");
exit ;
```

Użytkownika należy przekierować do strony głównej. Ponieważ pożądane jest przekierowanie z użyciem bezwzględnego adresu URL (co jest najbardziej niezawodne), trzeba włączyć plik konfiguracyjny i odczytać wartość BASE_URL.

4. Zakończ sekcję PHP.

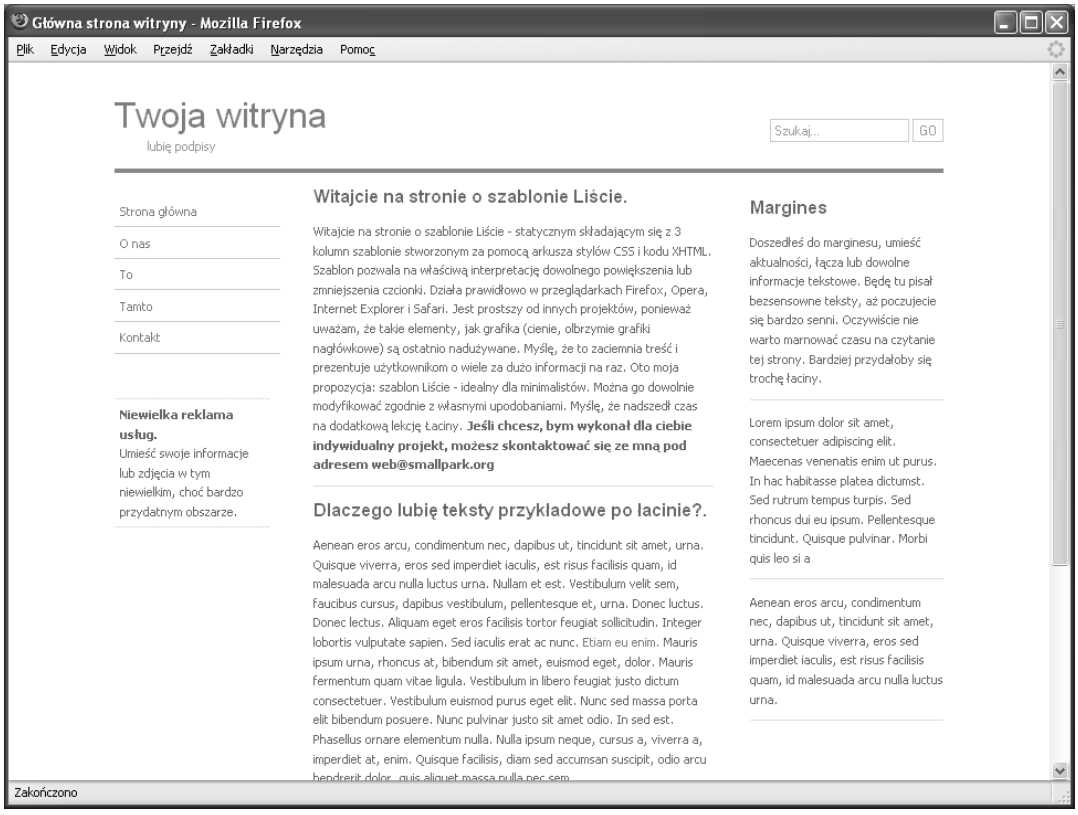
```
} // Koniec instrukcji IF defined()
?>
```

5. Dodaj dowolną treść.

```
      <h2>Witajcie na stronie
➥o szablonie Liście.</h2>
      <p>Witajcie na stronie o szablonie
➥Liście - statycznym składającym się
➥z 3 kolumn szablonie stworzonym za
➥pomocą arkusza stylów CSS i kodu XHTML.
➥Szablon pozwala na właściwą
➥interpretację dowolnego powiększenia
➥lub zmniejszenia czcionki. Działa
➥prawidłowo w przeglądarkach Firefox,
➥Opera, Internet Explorer i Safari. Jest
➥prostszy od innych projektów, ponieważ
➥uważam, że takie elementy, jak grafika
➥(cienie, olbrzymie grafiki nagłówkowe),
➥są ostatnio nadużywane. Myślę, że to
➥zaciemnia treść i prezentuje
➥użytkownikom o wiele za dużo informacji
➥naraz. Oto moja propozycja: szablon
➥Liście - idealny dla minimalistów.
➥Można go dowolnie modyfikować zgodnie
➥z własnymi upodobaniami. Myślę, że
➥nadszedł czas na dodatkową lekcję
➥łaciny. <strong>Jeśli chcesz, bym
➥wykonał dla ciebie indywidualny
➥projekt, możesz skontaktować się ze mną
➥pod adresem web@smallpark.org
➥</strong></p>
```

Można tu wprowadzić dowolną kombinację HTML i PHP, tak jak na dowolnej innej stronie PHP. W tym kroku pominąłem nieco treści, można ją jednak znaleźć w wersji skryptu dostępnej do pobrania.

6. Zapisz plik jako *main.inc.php*, umieść w katalogu dokumentów na serwerze WWW (w folderze *moduły* — rysunek 2.2) i przetestuj przez wywołanie skryptu *index.php* w przeglądarce WWW (rysunek 2.10).



Rysunek 2.10. *Kompletna główna strona serwisu — modularna i zarządzana za pomocą szablonu*

Tworzenie modułu wyszukiwania

W ostatnim przykładzie utworzyłem główny moduł treści. Był zaskakująco prosty, w związku z czym, dla urozmaicenia, pokażę inny przykład. W celu zademonstrowania modułu sterowanego za pomocą PHP, utworzę funkcję wyszukiwania. Pamiętajmy jednak, że bez rzeczywistej treści i zaplecza w postaci bazy danych nie można zaimplementować praktycznie działającej funkcji wyszukiwania. Nie jest to jednak istotne. W tym przykładzie skoncentrujemy się na pokazaniu, w jaki sposób wykorzystać PHP do obsługi formularzy w obrębie struktury modularnej. Pewnie wielu Czytelników zaskoczy fakt, jak nieskomplikowana jest to czynność.

Aby utworzyć moduł wyszukiwania:

1. Utwórz nowy skrypt PHP w edytorze tekstu lub środowisku IDE (skrypt 2.6).

```
<?php # Skrypt 2.6 - search.inc.php
```

Skrypt 2.6. *Moduł wyszukiwania zwraca pewne wyniki. Zaprezentowano go w celu pokazania, jak łatwo można obsługiwać formularze, nawet w przypadku modularnej struktury witryny*

Skrypt

```
<?php # Skrypt 2.6 - search.inc.php

/*
 *  To jest moduł wyszukiwania.
 *  Ta strona jest dołączana przez skrypt index.php.
 *  Należy do niej przekazać zmienną $_GET['terms'].
 */

// Przekierowanie w przypadku próby bezpośredniego dostępu:
if (!defined('BASE_URL')) {

    // Potrzebna jest stała BASE_URL zdefiniowana w pliku konfiguracyjnym:
    require_once ('../include/config.inc.php');

    // Przekierowanie do strony głównej:
    $url = BASE_URL . 'index.php?p=search';

    // Czy przesłano kryteria wyszukiwania?
    if (isset($_GET['terms'])) {
        $url .= '&terms=' . urlencode($_GET['terms']);
    }

    header ("Location: $url");
    exit;

} // Koniec instrukcji IF !defined().

// Wyświetlenie nagłówka:
echo '<h2>Wyniki wyszukiwania</h2>';

// Wyświetlenie wyników wyszukiwania,
// jeśli przesłano formularz.
if (isset($_GET['terms']) && ($_GET['terms'] != 'Szukaj...') ) {

    // Wysłanie zapytania do bazy danych.
    // Pobranie wyników.
    // Wyświetlenie ich:
    for ($i = 1; $i <= 10; $i++) {
        echo <<<EOT
<h4><a href="#">Wyniki wyszukiwania #$i</a></h4>
<p>To jest jakiś opis. To jest jakiś opis. To jest jakiś opis. To jest jakiś opis.</p>\n
EOT;
    }

} else { // Informacja o sposobie użycia formularza wyszukiwania.
    echo '<p class="error">W celu wyszukiwania informacji w tej witrynie, należy użyć
formularza wyszukiwania wyświetlającego się w górnej części strony.</p>';
}
?>
```

2. Przekieruj użytkownika, jeśli podjął próbę bezpośredniego dostępu do strony:

```
if (!defined('BASE_URL')) {
    require_once
➥('../include/config.inc.php');
    $url = BASE_URL . 'index.php?p=search';
    if (isset($_GET['terms'])) {
        $url .= '&terms=' .
➥urlencode($_GET['terms']);
    }
    header ("Location: $url");
    exit ;
}
```

Większa część kodu w tym przykładzie jest identyczna z kodem modułu *main.inc.php*. Są jednak dwie zmiany. Po pierwsze, adres URL przekierowania zmodyfikowano na `BASE_URL` plus *index.php?p=search*. Technikę tę można zastosować dla dowolnego modułu. Dzięki temu za pośrednictwem skryptu *index.php* można przekierować użytkownika na każdą stronę. Po drugie, jeśli z jakiegoś powodu użytkownik dotarłby do tej strony podczas przesyłania formularza, kryteria wyszukiwania wyświetliłyby się w adresie URL. W takiej sytuacji kryteria wyszukiwania zostałyby przekazane do skryptu. Dzięki temu bezpośredni dostęp pod adres URL *www.example.com/moduly/search.inc.php?terms=blah* również jest poprawną operacją wyszukiwania.

3. Wyświetl nagłówek:

```
echo '<h2>Wyniki wyszukiwania</h2>';
```

4. Sprawdź, czy zdefiniowano poprawne kryteria wyszukiwania.

```
if (isset($_GET['terms']) &&
➥($_GET['terms'] != 'Szukaj...') ) {
```

Operacja wyszukiwania w bazie danych nastąpi tylko wtedy, gdy użytkownik przekaże kryteria wyszukiwania w ramach adresu URL. W polu wyszukiwania występuje domyślna wartość *Szukaj…*, zatem należy ją wykluczyć.

Korzystanie ze szkieletów programowych

Szkielety programowe (ang. *framework*) są bibliotekami kodu mającymi na celu ułatwienie projektowania aplikacji. Dla języka PHP dostępnych jest ponad 40 popularnych szkieletów. Firma Zend — twórca interpretera PHP — zainicjowała ostatnio dyskusję na temat własnej wersji szkieletu (*http://framework.zend.com*).

Zazwyczaj szkielety są systemami modularnymi, podobnymi do tego, który opisano w niniejszym rozdziale, ale o znacznie większej skali. Argumenty przemawiające za stosowaniem szkieletów programowych są podobne do tych, jakich używa się, przekonując do korzystania z technik programowania obiektowego lub pakietów PEAR: za ich pomocą można szybko stworzyć aplikacje, które mają więcej funkcji i są bezpieczniejsze aniżeli tworzone w tradycyjny sposób. Argumenty przeciwko używaniu szkieletów również są podobne do tych dotyczących programowania obiektowego i PEAR: trzeba poświęcić czas, by nauczyć się ich stosowania, nie mówiąc już o opanowaniu do perfekcji, a czasami trudno je dostosować do indywidualnych potrzeb aplikacji. Istnieje również prawdopodobieństwo, że witryny zarządzane za pomocą szkieletów będą działały wolniej (ze względu na konieczność dodatkowego przetwarzania).

Ja nie należę do zwolenników szkieletów, ponieważ lubię tworzyć kod od początku. Są jednak tacy, którzy chętnie z nich korzystają. Z analizą i porównaniem często używanych szkieletów aplikacji można się zapoznać pod adresem *www.phpit.net/article/ten-different-php-frameworks/*.

Rysunek 2.11. *Bez przesłania kryteriów wyszukiwania operacja wyszukiwania nie powiedzie się*

Rysunek 2.12. *Wprowadzenie dowolnych kryteriów wyszukiwania spowoduje wyświetlenie przykładowych wyników*

5. Wyświetl wyniki wyszukiwania.

```
    for ($i = 1; $i <= 10; $i++) {
        echo <<<EOT
<h4><a href="#">Wyniki wyszukiwania
➥#$i</a></h4>
<p>To jest jakiś opis. To jest jakiś opis. To
➥jest jakiś opis. To jest jakiś opis.</p>\n
EOT;
 }
```

Ponieważ nie mamy rzeczywistej bazy danych do wyszukiwania, wykorzystałem pętlę `for` w celu wyświetlenia dziesięciu wyników wyszukiwania. W tym przykładzie użyłem składni *heredoc* zgodnie z opisem zamieszczonym w rozdziale 1., „Zaawansowane techniki programowania w PHP".

6. Zakończ stronę.

```
} else {
    echo '<p class="error">W celu
➥wyszukiwania informacji w tej witrynie,
➥należy użyć formularza wyszukiwania
➥wyświetlającego się w górnej części
➥strony.</p>';
}
?>
```

Ta część instrukcji warunkowej zostanie wykonana, jeśli nie wprowadzono prawidłowych kryteriów wyszukiwania (rysunek 2.11).

7. Zapisz plik jako *search.inc.php*, umieść go w katalogu dokumentów serwera WWW (w folderze *moduly*) i przetestuj, próbując przesłać formularz (rysunek 2.12).

Operacje z buforem przeglądarki

Przeglądarki WWW i serwery proxy (mechanizmy, które dostawcy Internetu i inne firmy stosują w celu poprawy wydajności sieci) zwykle buforują strony WWW. Buforowanie strony polega na zapisaniu jej zawartości (lub części, na przykład samej grafiki bądź klipów wideo), a następnie — w przypadku żądania strony — dostarczenia jej z bufora zamiast z serwera.

Dla większości użytkowników nie stanowi to problemu. W większości przypadków nie zdają sobie sprawy, że otrzymują przestarzałą wersję strony lub zapisanej na niej grafiki. Jeśli jednak podczas tworzenia witryny zechcemy zmusić przeglądarkę WWW (spójrzmy prawdzie w oczy: większość użytkowników korzysta z przeglądarki Internet Explorer), by rozpoznała zmiany wprowadzone na stronie, dostrzeżemy ciemną stronę buforowania. W przypadku dynamicznych stron WWW sterowanych za pomocą PHP chcemy być pewni, że użytkownicy otrzymają najbardziej aktualną wersję strony.

Na buforowanie — zarówno w przeglądarkach WWW, jak i na serwerach proxy — można wpływać za pomocą funkcji języka PHP `header()`. Dostępne są cztery typy nagłówków:

- `Last-Modified`,
- `Expires`,
- `Pragma`,
- `Cache-Control`.

Trzy pierwsze typy nagłówków są częścią standardu HTTP 1.0. Dla nagłówka `Last-Modified` wykorzystuje się wartość czasu UTC (ang. *Universal Time Coordinated*). Jeśli system buforowania stwierdzi, że wartość nagłówka `Last-Modified` zawiera datę późniejszą od daty strony w buforze, pobiera nowszą wersję z serwera.

Ustawienie `Expires` wykorzystuje się jako wskaźnik czasu, po którym wersja strony umieszczona w buforze nie powinna być używana (według czasu GMT — ang. *Greenwich Mean Time*). Ustawienie parametru `Expires` na wartość z przeszłości wymusza pobranie strony z serwera za każdym razem:

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
```

Dyrektywa `Pragma` służy jako deklaracja sposobu obsługi strony. W celu zablokowania buforowania strony należy użyć następującej instrukcji:

```
header("Pragma: no-cache");
```

Nagłówek `Cache-Control` dodano w wersji HTTP 1.1. Pozwala on na dokładniejszą kontrolę buforowania (nagłówki HTTP 1.0 są również dostępne). Dostępnych jest wiele ustawień nagłówka `Cache-Control` (tabela 2.1).

Aby zablokować buforowanie strony przez wszystkie systemy, można skorzystać z następujących nagłówków:

```
header("Last-Modified: Thu, 9 Nov 2006 14:26:00
➥GMT"), // W tej chwili
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
➥// Data w przeszłości!
header("Pragma: no-cache");
header("Cache-Control: no-cache");
```

Choć jest to sposób powszechnie stosowany, jest bardzo radykalny. Z całą pewnością, nie dla każdego skryptu PHP trzeba blokować buforowanie. Nawet w najbardziej aktywnych witrynach można buforować niektóre skrypty przez kilka minut (bardzo aktywne witryny w ciągu minuty otrzymują wiele żądań — dzięki wersji w buforze serwer nie będzie zmuszony do obsługiwania ich wszystkich). W celu zastosowania opisanych pojęć spróbujmy zmodyfikować skrypt *view_tasks.php* (skrypt 1.3) z rozdziału 1.

Tabela 2.1. *Dyrektywy nagłówka Cache-Control*

Dyrektywa	Znaczenie
`public`	Buforowanie dozwolone wszędzie
`private`	Buforowanie tylko przez przeglądarki
`no-cache`	Buforowanie zabronione wszędzie
`must-revalidate`	Mechanizm buforowania musi sprawdzić, czy są dostępne nowsze wersje
`proxy-revalidate`	Mechanizm buforowania serwerów proxy musi sprawdzić, czy są dostępne nowsze wersje
`max-age`	Maksymalny czas buforowania treści, w sekundach
`s-maxage`	Przesłania wartość parametru `max_age` w przypadku współużytkowanych buforów

Aby zarządzać buforowaniem:

1. Otwórz skrypt *view_tasks.php* w edytorze tekstu lub środowisku IDE (skrypt 2.7).
2. Przed wysłaniem treści do przeglądarki WWW dodaj otwierający znacznik PHP (skrypt 2.7).

```
<?php # skrypt 2.7 - view_tasks.php
```

Jak większość Czytelników z pewnością wie, funkcję `header()` można wywołać tylko wtedy, gdy jeszcze niczego nie wysłano do przeglądarki. Dotyczy to tekstu, kodu HTML, a nawet spacji.

Skrypt 2.7. *W zmodyfikowanej wersji skryptu view_tasks.php (skrypt 1.3) wykorzystano funkcję header() do sterowania buforowaniem*

Skrypt

```
<?php # Skrypt 2.7 - view_tasks.php

// Nawiązanie połączenia z bazą danych.
$dbc = @mysqli_connect ('localhost', 'uzytkownik', 'haslo', 'test') OR die ('<p>Nie można
połączyć się z bazą danych!</p></body></html>');

// Pobranie najnowszych dat w postaci znaczników czasu:
$q = 'SELECT UNIX_TIMESTAMP(MAX(data_wprowadzenia)), UNIX_TIMESTAMP(MAX(data_ukonczenia))
FROM zadania';
    $r = mysqli_query($dbc, $q);
list($max_a, $max_c) = mysqli_fetch_array($r, MYSQLI_NUM);

// Wyznaczenie znacznika czasu oznaczającego późniejszą datę:
$max = ($max_a > $max_c) ? $max_a : $max_c;

// Utworzenie przedziału buforowania w sekundach:
$interval = 60 * 60 * 6; // 24 godziny

// Wysłanie nagłówka:
header ("Last-Modified: " . gmdate ('r', $max));
header ("Expires: " . gmdate ("r", ($max + $interval)));
header ("Cache-Control: max-age=$interval");
?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">
<head>
    <meta http-equiv="content-type" content="text/html; charset=windows-1250" />
    <title>Przeglądanie zadań</title>
</head>
<body>
<h3>Bieżąca lista zadań</h3>
<?php

/*  Ten skrypt wyświetla wszystkie zdefiniowane zadania.
 *  Do wyświetlenia zadań w postaci zagnieżdżonych list
 *  wykorzystano funkcję rekurencyjną.
 */

// Funkcja wyświetlająca listę.
// Pobiera jeden argument:  tablicę.
function make_list ($parent) {

    // Potrzebna jest globalna tablica $tasks:
```

Skrypt 2.7. *— ciąg dalszy*

Skrypt

```
42        global $tasks;
43
44        // Początek uporządkowanej listy:
45        echo '<ol>';
46
47        // Przetwarzanie w pętli wszystkich tablic podrzędnych:
48        foreach ($parent as $task_id => $todo) {
49
50            // Wyświetlenie elementu:
51            echo "<li>$todo";
52
53            // Sprawdzenie, czy z zadaniem są powiązane zadania podrzędne:
54            if (isset($tasks[$task_id])) {
55
56                // Wywołanie funkcji:
57                make_list($tasks[$task_id]);
58
59            }
60
61            // Wyświetlenie elementu:
62            echo '</li>';
63
64        } // Koniec pętli FOREACH.
65
66        // Koniec uporządkowanej listy:
67        echo '</ol>';
68
69    } // Koniec funkcji make_list().
70
71    // Pobranie wszystkich niezrealizowanych zadań:
72    $q = 'SELECT id_zadania, id_z_nadrzednego, zadanie FROM zadania WHERE data_ukonczenia=
      "0000-00-00 00:00:00" ORDER BY id_z_nadrzednego, data_wprowadzenia ASC';
73    $r = mysqli_query($dbc, $q);
74
75    // Zainicjowanie tablicy:
76    $tasks = array();
77
78    while (list($task_id, $parent_id, $task) = mysqli_fetch_array($r, MYSQLI_NUM)) {
79
80        // Dodanie zadania do tablicy:
81        $tasks[$parent_id][$task_id] =  $task;
82
83    }
84
85    // Instrukcje diagnostyczne:
86    //echo '<pre>' . print_r($tasks,1) . '</pre>';
87
88    // Przesłanie pierwszego elementu tablicy
89    // do funkcji make_list():
90    make_list($tasks[0]);
91
92    ?>
93    </body>
94    </html>
```

3. Nawiąż połączenie z bazą danych.

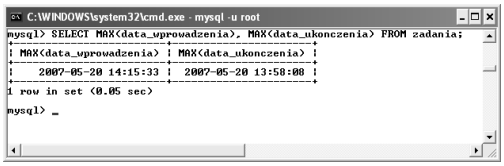
```
$dbc = @mysqli_connect ('localhost',
➥'uzytkownik', 'haslo', 'test') OR die
➥('<p>Nie można połączyć się z bazą
➥danych!</p></body></html>');
```

Aby dokładnie określić datę ostatniej modyfikacji strony, skrypt wykona zapytanie do bazy danych. W tabeli `zadania` są dwie kolumny zawierające daty i godziny: `data_wprowadzenia` i `data_ukonczenia`. Za każdym razem przy aktualizacji zawartości strony te dwie wartości są ustawiane na bieżącą datę i godzinę (nie ma opcji usuwania).

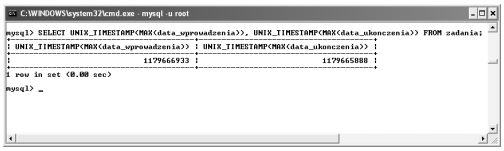
4. Pobierz ostatnie daty aktualizacji z tabeli.

```
$q = 'SELECT
➥UNIX_TIMESTAMP(MAX(data_wprowadzenia)),
➥UNIX_TIMESTAMP(MAX(data_ukonczenia)) FROM
➥zadania';
$r = mysqli_query($dbc, $q);
list($max_a, $max_c) = mysqli_fetch_array($r,
➥MYSQLI_NUM);
$max = ($max_a > $max_c) ? $max_a : $max_c;
```

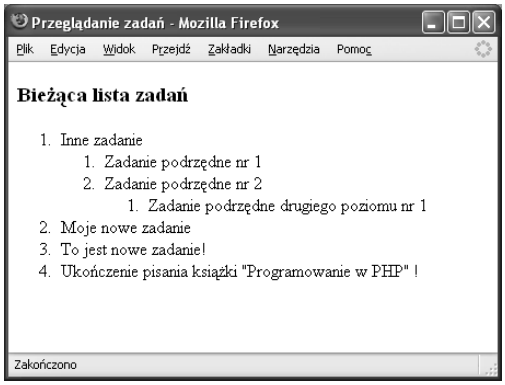
Zapytanie zwraca największe wartości pól `data_wprowadzenia` i `data_ukonczenia`. Ponieważ zapytanie zwróci te wartości w postaci trudnej do przetwarzania (rysunek 2.13), zastosowano funkcję `UNIX_TIMESTAMP` w celu przekształcenia ich na liczby typu `integer` (rysunek 2.14). Następnie użyto operatora trzyelementowego w celu przypisania największej wartości (oznaczającej najbliższą datę) do zmiennej `$max`.



Rysunek 2.13. *Zapytanie zwraca pola typu TIMESTAMP w następującej postaci*



Rysunek 2.14. *Wyniki zapytania w postaci wykorzystanej w skrypcie*



Rysunek 2.15. *Strona WWW z mechanizmem zarządzania buforowaniem*

5. Zdefiniuj sensowny przedział buforowania.

```
$interval = 60 * 60 * 6;
```

Znaczenie słowa „sensowny" zależy od tego, ilu użytkowników odwiedza stronę (tzn. od obciążenia serwera) oraz jak często jest aktualizowana. Dla potrzeb tego przykładu użyłem wartości sześciu godzin (wartość czasu trwania przedziału jest wyrażona w sekundach — 60 sekund×60×6).

6. Wyślij nagłówek `Last-Modified`.

```
header("Last-Modified: " . gmdate
➥('r', $max));
```

Nagłówek ustawia datę modyfikacji skryptu na wartość daty ostatniej aktualizacji bazy danych. Opcja `"r" gmdate()` (oraz `date()`) zwracają datę sformatowaną zgodnie ze specyfikacją HTTP.

7. Ustaw nagłówek `Expires`.

```
header("Expires: " . gmdate ("r", ($max +
➥$interval));
```

Wartość daty ważności jest określona jako bieżąca godzina, do której dodano zdefiniowany przedział.

8. Ustaw nagłówek `Cache-Control`.

```
header("Cache-Control: max-age= $interval");
?>
```

Jest to odpowiednik nagłówka `Expires` według specyfikacji HTTP 1.1. Zamiast przekazywania wartości daty, ustawiamy wartość `max-age` w sekundach.

9. Usuń połączenie z bazą danych, które było w skrypcie w pierwotnej postaci.

Operację tę przeniesiono na początek skryptu — do punktu 3.

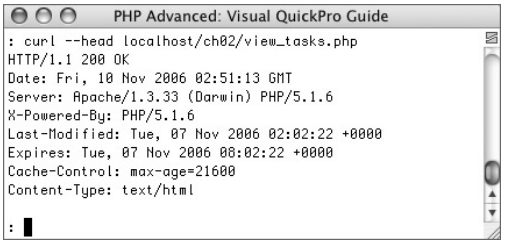
10. Zapisz skrypt w pliku *view_task.php*, umieść go w katalogu dokumentów serwera WWW i przetestuj w przeglądarce (rysunek 2.15).

Wskazówki

- Zwróćmy uwagę, że teoretycznie buforowanie jest bardzo korzystne dla aplikacji — służy do zminimalizowania liczby niepotrzebnych żądań pobrania stron kierowanych do serwera. W przypadku właściwego zarządzania buforami są one doskonałym mechanizmem zarówno dla serwera, jak i klienta.
- Jeśli w systemie zainstalowano bibliotekę cURL, można uruchomić poniższe polecenie w celu wyświetlenia nagłówków strony (rysunek 2.16):

```
curl --head
➥http://www.przyklad.com/strona.php
```

  Bibliotekę cURL omówiono w rozdziale 9., „PHP w sieci".
- Jeśli w aplikacji wykorzystywane są sesje, można dostosować buforowanie sesji za pomocą funkcji `session_cache_limit()`. Więcej informacji można znaleźć w podręczniku online.
- Buforowaniem stron można również sterować za pomocą znaczników `META` umieszczonych w nagłówku HTML dokumentu. Metoda ta w niektórych przeglądarkach może działać nieco mniej niezawodnie w porównaniu z wykorzystaniem funkcji `header()`.
- Wydajność aplikacji klient-serwer można również poprawić — w przypadku bardziej rozbudowanych skryptów — za pomocą mechanizmu kompresji wyniku Zlib lub funkcji `ob_gzhandler()`. Więcej informacji na temat obu technik można znaleźć w podręczniku online PHP.



Rysunek 2.16. *Użycie cURL do przeglądania nagłówków zwracanych przez stronę view_tasks.php*

Buforowanie po stronie serwera

Istnieje alternatywny typ buforowania, który można wykorzystać w celu zarządzania komunikacją pomiędzy klientem a serwerem. Kiedy użytkownik zażąda skryptu PHP, serwer WWW wysyła zapytanie do modułu obsługi PHP o odczytanie i przetworzenie kodu. W buforach działających po stronie serwera są zapisywane przetworzone wersje skryptów, dzięki czemu serwer może je wysłać do użytkownika bez konieczności przetwarzania. Buforowanie po stronie serwera może w znaczny sposób poprawić wydajność aplikacji, ale zazwyczaj wymaga większej kontroli nad serwerem niż posiada przeciętny użytkownik (czytaj: taki, który korzysta ze współdzielonych usług hostingu).

Osoby, które chciałyby przeanalizować i być może także zaimplementować buforowanie po stronie serwera, mają do wyboru wiele opcji. APC (ang. *Alternative PHP Cache*) jest na tyle popularnym mechanizmem, że wchodzi w skład repozytorium PECL (*http://pecl.php.net*). System jest darmowy, ale jego instalacja jest dość złożona. Firma Zend (*www.zend.com*) oferuje darmowy *Zend Optimizer*. Jego instalacja jest dosyć prosta, ale należy pamiętać o jej aktualizacji w przypadku aktualizacji PHP (system jest dedykowany dla wersji).

Spośród opcji niewymagających instalacji oprogramowania na serwerze wymienić można jeszcze pakiety PEAR `Cache` oraz `Cache_Lite`. Można również pokusić się o napisanie własnego systemu buforowania, ale nie jest to najbardziej ekonomiczne rozwiązanie