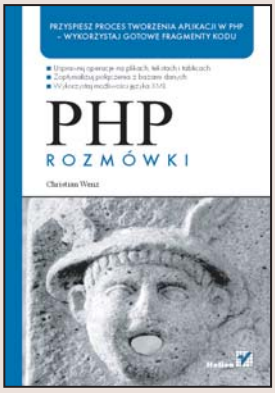
PHP. Rozmówki



Autor: Christian Wenz
Tłumaczenie: Radosław Meryk
ISBN: 83-246-0324-7
Tytuł oryginału: PHP Phrasebook
Format: B5, stron: 360

Język PHP to jedna z najpopularniejszych platform programistycznych służących do tworzenia aplikacji internetowych. Wszędzie tam, gdzie zamierzamy dynamicznie generować treść witryny, gromadzić i przetwarzać dane, identyfikować użytkowników strony lub wysyłać pliki do witryny WWW, wykorzystujemy PHP. Gdy pojawiają się problemy, wertujemy książki, szukając porad i przykładów. Jeśli przydatne wskazówki znajdują się w jednym miejscu, praca szybko posuwa się do przodu.

Książka „PHP. Rozmówki" to zbiór ponad 100 przykładów kodu opatrzonych komentarzami i dokładnie przetestowanych w różnych systemach operacyjnych i przeglądarkach. Autor podpowiada gotowe rozwiązania problemów, z którymi borykają się na co dzień programiści PHP. Przykładowy kod z łatwością można dostosować do własnych potrzeb, przyspieszając w ten sposób pracę nad aplikacją i zwiększając produktywność.

- Operacje na łańcuchach tekstowych
- Stosowanie wyrażeń regularnych
- Przetwarzanie tablic
- Operacje na datach
- Obsługa formularzy WWW
- Uwierzytelnianie użytkowników
- Stosowanie plików cookie i mechanizmów sesji
- Praca z systemem plików na serwerze
- Połączenia z bazami danych
- Przetwarzanie dokumentów XML
- Komunikacja z usługami sieciowymi

Do efektywnej pracy z PHP wystarczy ta książka — zatem po co korzystać z opasłych tomów?

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

5

Zapamiętywanie ustawień użytkowników — pliki cookie i sesje

HTTP (*Hypertext Transfer Protocol*) jest protokołem bezstanowym. Mówiąc prosto, klient (przeglądarka WWW) łączy się z serwerem WWW, przesyła żądanie i uzyskuje odpowiedź. Następnie połączenie jest zamykane. W konsekwencji, jeśli następnym razem ten sam klient prześle żądanie do tego samego serwera WWW, będzie to nowe żądanie, a zatem serwer WWW nie będzie mógł zidentyfikować użytkownika. Takie działanie stanowi oczywisty problem dla aplikacji, które muszą utrzymywać stany (na przykład aplikacji e-commerce z implementacją koszyka na zakupy).

Z ograniczeniem tym można jednak sobie poradzić na kilka sposobów. Podstawowa idea polega na przesłaniu pewnych informacji wraz z odpowiedzią HTTP. Informacje te są przesyłane do serwera we wszystkich kolejnych żądaniach. Istnieją następujące możliwości:

- wysłanie danych za pomocą metody `POST` (tzn. za każdym razem wymagany jest formularz),
- wysłanie danych za pomocą metody `GET` (tzn. poprzez dołączenie informacji do adresu URI żądania),
- wysłanie danych jako części nagłówka HTTP (w postaci pliku cookie).

W praktyce używa się jednej z dwóch metod: sesji (z wykorzystaniem metody `GET` lub plików cookie) oraz plików cookie.

Istota plików cookie

Pliki cookie są przesyłane w nagłówkach HTTP. W uproszczeniu, plik cookie składa się z pary nazwa-wartość. Najważniejszą wadą plików cookie jest możliwość ich blokowania w przeglądarkach WWW (a także filtrowania na serwerach proxy). Niektórzy uważają, że pliki cookie stanowią zagrożenie dla prywatności użytkowników. Po części przyczyną tego poglądu był artykuł Johna Udella z marca 1997 roku, w którym autor stwierdził, że każdy plik cookie można odczytać z każdego serwera WWW, a zatem włączenie obsługi plików cookie oznacza utratę prywatności. Artykuł spowodował zamieszanie. Niestety, sprostowanie opublikowane dwa miesiące później nie wzbudziło już takiego zainteresowania.

Prawdą jest, że pliki cookie mają pewne ograniczenia:

- są powiązane z domenami — zazwyczaj z domeną, z której wysłano plik cookie,
- mogą być powiązane z katalogami na serwerze WWW,
- zawierają wyłącznie informacje tekstowe; maksymalnie 4096 bajtów (włącznie z nazwą pliku cookie i znakami = pomiędzy nazwami a wartościami),
- przeglądarki mogą zaakceptować tylko do 20 plików cookie na domenę i 300 plików cookie w ogóle (choć niektóre przeglądarki akceptują więcej).

UWAGA

Nieoficjalna specyfikacja plików cookie jest związana z przeglądarką Netscape i ciągle jest dostępna pod adresem *http://➥wp.netscape.com/newsref/std/cookie_spec.html*. Były próby stworzenia specjalnego dokumentu RFC dotyczącego plików cookie nowej generacji, ale inicjatywy te nie doczekały się jeszcze obsługi w żadnej z popularnych przeglądarek.

Pliki cookie są przesyłane w nagłówkach HTTP. W przypadku ustawienia pliku cookie w nagłówku HTTP tworzy się zapis `Set-Cookie`. Dalej występuje nazwa i wartość pliku cookie (obie w postaci ciągu znaków) oraz opcjonalnie inne informacje, takie jak data ważności, domena i ścieżka dostępu do pliku cookie. Na przykład, jeśli odwiedzimy witrynę *http://www.php.net*, witryna PHP prześle nagłówek w następującej postaci (oczywiście ustawienia języka i adres IP mogą być inne):

```
Set-Cookie:COUNTRY=DEU%2C84.154.17.84; expires=Thu
19-May-05 15:23:29 GMT; path=/; domain=.php.net
```

Istota plików cookie

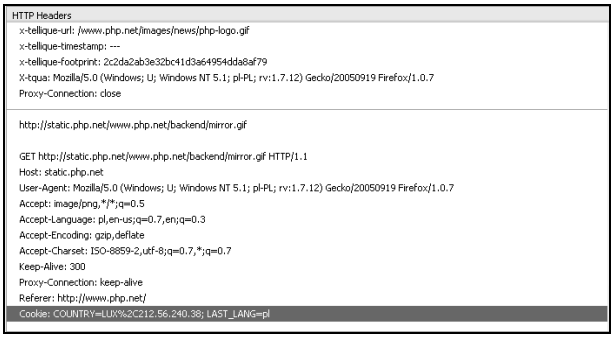
Kiedy przeglądarka (lub użytkownik) zaakceptuje plik cookie, zostanie on przesłany na serwer podany w nagłówku HTTP `Cookie`:

```
Cookie:COUNTRY=DEU%2C84.154.17.84
```

Dla pliku cookie można ustawić datę ważności. Jeśli zostanie ustawiona, plik będzie przechowywany co najwyżej do tej daty. Jest to tzw. *trwały plik cookie* (ang. *persistent cookie*). Po upływie tego okresu przeglądarka automatycznie go usunie — choć może to również zdarzyć się wcześniej; na przykład w przypadku zapisania maksymalnej liczby plików cookie w przeglądarce najstarsze pliki cookie zostaną usunięte. Jeśli data ważności pliku cookie nie zostanie ustawiona, taki plik określa się jako tzw. *plik sesji* lub *plik tymczasowy*. Pliki cookie tego typu są przechowywane tak długo, jak długo działa przeglądarka WWW. Podczas zamykania przeglądarki tymczasowe pliki cookie są usuwane.

WSKAZÓWKA

Aby oglądać nagłówki HTTP, można skorzystać ze specjalnych rozszerzeń standardowych przeglądarek WWW. W przypadku przeglądarek Mozilla (włącznie z przeglądarką Firefox) wielkie możliwości oferuje rozszerzenie *LiveHTTPHeaders* dostępne pod adresem *http://livehttpheaders.mozdev.org/*. Użytkownicy przeglądarki Microsoft Internet Explorer mogą zainstalować pasek narzędzi *ieHTTPHeaders* dostępny pod adresem *http://➥www.blunck.info/iehttpheaders.html*. Na rysunku 5.1 pokazano przykładowe nagłówki dla przeglądarki FireFox podczas dostępu do strony macierzystej PHP.



Rysunek 5.1. Pliki cookie są ustawiane w nagłówkach HTTP

Tworzenie plików cookie

```
<?php
    setcookie('version', phpversion());
?>
Próba wysłania pliku cookie.
```

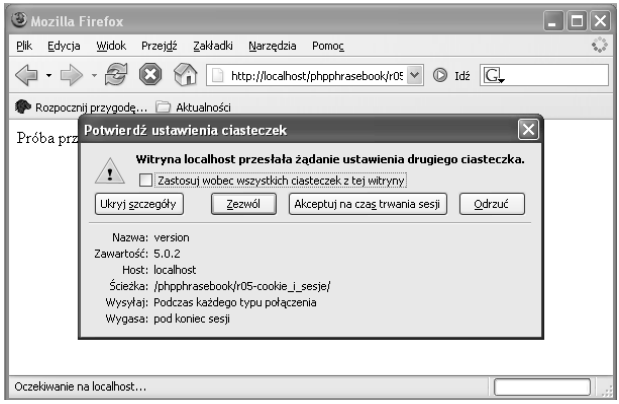
Do tworzenia plików cookie służy funkcja PHP `setcookie()`. Funkcja przyjmuje następujące parametry (tylko pierwszy z nich jest obowiązkowy):

1. Nazwa pliku cookie.
2. Wartość pliku cookie.
3. Data ważności (w formacie uniksowego znacznika czasu).
4. Katalog na serwerze WWW, z którego można uzyskać dostęp do pliku cookie.
5. Domena, z której można uzyskać dostęp do pliku cookie.

6. Informacja, czy plik cookie można przesyłać tylko za pomocą bezpiecznych połączeń HTTPS (SSL).

W kodzie zamieszczonym na początku tego podrozdziału ustawiono prosty plik cookie sesji z wartością w postaci bieżącej wersji PHP.

Na rysunku 5.2 pokazano okno z ostrzeżeniem wyświetlane w przeglądarce Firefox. Bez trudu można odczytać nazwę pliku cookie i jego wartość.



Rysunek 5.2. Firefox otrzymuje plik cookie i wyświetla pytanie do użytkownika o to, co z nim zrobić

Odczytywanie plików cookie

Wszystkie pliki cookie, do których serwer ma dostęp (nie muszą to być wszystkie pliki cookie, które są zapisane w przeglądarce!) są dostępne w tablicy superglobalnej

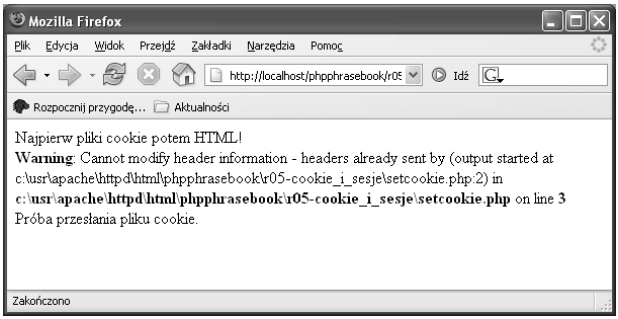
$_COOKIE. W kodzie zaprezentowanym poniżej wszystkie pliki cookie są odczytywane w pętli foreach i wysyłane do klienta w postaci tabeli HTML.

```
<table>
<?php
  foreach ($_COOKIE as $name => $value) {
    printf('<tr><td>%s</td><td>%s</td></tr>',
      htmlspecialchars($name),
      htmlspecialchars($value));
  }
?>
</table>
```

Czytanie plików cookie (getcookie.php)

OSTRZEŻENIE

Ponieważ pliki cookie są przesyłane jako nagłówki HTTP, muszą być utworzone przed wysłaniem wyjścia HTML (chyba że korzystamy z buforowania wyników). W innym przypadku uzyskamy komunikat o błędzie podobny do tego, który pokazano na rysunku 5.3.



Rysunek 5.3. Pliki cookie trzeba przesłać przed wysłaniem zawartości HTML

UWAGA

Zazwyczaj interpreter PHP automatycznie unieszkodliwia znaki specjalne w wartościach plików cookie poprzez poprzedzenie ich znakami lewego ukośnika (szczególnie w przypadku przekazywania plików cookie poprzez URL). Jednak począwszy od PHP 5 można to robić ręcznie, poprzez wysłanie „surowych" danych pliku cookie. Aby to uczynić, należy przekazać do funkcji `setrawcookie()` te same parametry, które były przekazane do funkcji `setcookie()`. Obie funkcje działają podobnie. Różnica polega na tym, że pierwsza z nich nie koduje wartości pliku cookie. Aby ją zakodować, należy ręcznie wywołać funkcję `urlencode()`.

Pozbywanie się „magicznych" cudzysłowów (apostrofów) z plików cookie

Magiczne cudzysłowy (apostrofy), które opisano i wyklęto w poprzednim rozdziale, dotyczą również plików cookie, ponieważ to również są dane pochodzące od klientów. Tak więc, jeśli ustawiono opcję `magic_quotes` na wartość `on`, znaki apostrofów i cudzysłowów będą poprzedzane znakiem lewego ukośnika. Aby pozbyć się ukośników, można wykorzystać poniższy kod. Podobny kod zastosowano w poprzednim rozdziale do usunięcia ukośników z danych formularzy (danych `GET` i `POST`).

Jeśli ustawiono opcję `magic_quotes`, funkcja `stripsla-➥shes()` będzie wywoływana rekurencyjnie dla wszystkich danych w tablicy `$_COOKIE`.

```
<?php
  function stripCookieSlashes($arr) {
    if (!is_array($arr)) {
      return stripslashes($arr);
    } else {
      return array_map('stripCookieSlashes', $arr);
    }
  }

  if (get_magic_quotes_gpc()) {
    $_COOKIE  = stripCookieSlashes($_COOKIE);
  }
?>
```

Usuwanie „magicznych" cudzysłowów (apostrofów) z plików cookie (stripCookieSlashes.inc.php)

Skrypt *stripCookieSlashes.inc.php* należy włączyć do wszystkich skryptów PHP, które odczytują pliki cookie za pomocą następującej instrukcji:

```
require_once 'stripCookieSlashes.inc.php';
```

Ustawianie daty ważności względem innej daty

Data ważności pliku cookie jest przekazywana za pomocą trzeciego parametru funkcji `setcookie()`. Jest to liczba całkowita, dlatego trzeba użyć uniksowego znacznika czasu.

Ustawianie daty ważności względem innej daty

W rozdziale 3., „Daty i godziny", zamieszczono sporo informacji dotyczących sposobu wykorzystania tego rodzaju danych.

```
<?php
  setcookie('version', phpversion(), time() +
  21*24*60*60);
?>
Próba wysłania pliku cookie.
```

Ustawianie daty ważności pliku cookie względem innej daty (setcookie-expiry.php)

Zwykle lepszym pomysłem jest ustawienie daty ważności pliku cookie względem innej daty („za trzy tygodnie"), niż podawanie daty konkretnej („koniec maja 2006"). W przypadku użycia dat bezwzględnych skrypt trzeba regularnie modyfikować, ponieważ często data wygaśnięcia ważności nadchodzi bardzo szybko. Ustawianie czasu ważności na bardzo odległa datę w przyszłości, na przykład w roku 2030, uważa się za działanie nieprofesjonalne. Klient, który otrzyma taki plik cookie, najprawdopodobniej w roku 2030 już nie będzie się logował na serwerze.

Z tych względów warto stosować daty względem innych dat. Funkcja PHP `date()` pobiera wartość uniksowego znacznika czasu dla daty bieżącej. Do tej wartości wystarczy dodać liczbę sekund, przez jaką plik cookie ma być ważny. Kod zaprezentowany na poprzednim listingu ustawia plik cookie, który będzie ważny przez trzy tygodnie.

Ustawianie daty ważności specyficznej dla klienta

```
setcookie('version', phpversion(),
    $_GET['time'] + 21*24*60*60)
```

Należy pamiętać, że decyzja dotyczące tego, kiedy plik cookie traci ważność, jest podejmowana po stronie klienta i z wykorzystaniem ustawień daty i godziny klienta. Jeśli zatem data klienta jest ustawiona nieprawidłowo (czego po stronie serwera nie można kontrolować), pliki cookie będą traciły ważność wcześniej lub później niż się spodziewamy. Nie warto zatem ustawiać plików cookie, które będą ważne przez godzinę. Nasze plany mogą zniweczyć tak błahe sytuacje jak zmiana czasu z zimowego na letni.

```
<?php
  if (isset($_GET['time']) && is_int($_GET['time']))
{
    setcookie('version', phpversion(),
      $_GET['time'] + 21*24*60*60);
  } else {
    setcookie('version', phpversion(),
      time() + 21*24*60*60);
  }
?>
Próba wysłania pliku cookie.
```

Ustawianie pliku cookie z określoną datą ważności (setcookie-specific.php)

Wystarczy jednak odrobina kodu w JavaScript, aby uniknąć opisanej pułapki. Kod zamieszczony na poniższym listingu to działający po stronie klienta skrypt JavaScript, który sprawdza bieżącą godzinę wyrażoną jako uniksowy znacznik

czasu i wysyła ją do działającego na serwerze skryptu *setcookie-specific.php*. Najważniejsza różnica pomiędzy funkcją PHP `time()` a metodą `getTime()` języka JavaScript polega na tym, że ta druga zwraca liczbę milisekund, jakie upłynęły od północy 1 stycznia 1970 roku, natomiast pierwsza liczbę sekund. Z tego powodu wartość otrzymaną ze skryptu JavaScript należy najpierw podzielić przez 1000 i zaokrąglić w dół do całości.

Kod skryptu *setcookie-specific.php* zamieszczono na początku tego podrozdziału. Przesyłana wartość jest wykorzystywana jako podstawa obliczenia relatywnej daty ważności pliku cookie.

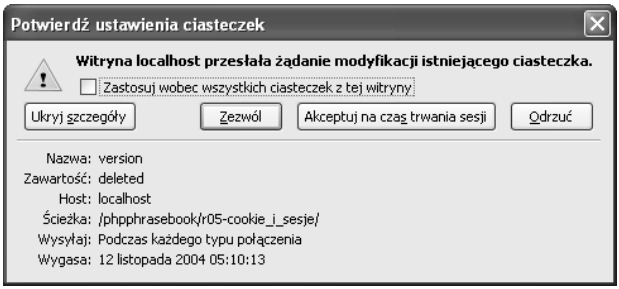
```
<script language = "JavaScript"
    type="text/javascript"><!-
    var epoche = (new Date() - getTime();
    epoche = Math.floor(epoche/1000);
    location.replace("setcookie-specific.php?time="
    + epoche);
//-></script>
```

Usuwanie plików cookie

```
setcookie('version', '', time() - 10*365*24*60*60);
```

Intuicyjnym sposobem usunięcia pliku cookie jest ustawienie jego wartości na pusty ciąg znaków. Ta operacja nie powoduje jednak usunięcia pliku cookie. Plik w dalszym ciągu istnieje, choć już nie ma ustawionej wartości. Lepszym sposobem jest ponowne przesłanie tego samego

pliku cookie z datą ważności w przeszłości. Także w tym przypadku trzeba uwzględnić nieprawidłowe ustawienia czasu lokalnego, a zatem należy wybierać bardzo odległe daty utraty ważności, na przykład 10 lat wcześniej. Sposób ten zaimplementowano poniżej, gdzie usunięto plik cookie ustawiony w kodzie pokazanym na poprzednim listingu. W tej implementacji połączono obie metody — ustawiono wartość na pusty ciąg znaków i datę wygaśnięcia pliku cookie w przeszłości. Wynik działania skryptu pokazano na rysunku 5.4 — przeglądarka próbuje usunąć plik cookie poprzez ustawienie daty jego wygaśnięcia na określony czas (w przeszłości, a zatem plik cookie przestaje istnieć).



Rysunek 5.4. Plik cookie będzie usunięty (jeśli użytkownik go zaakceptuje)

```
<?php
  setcookie('version', '', time() -
10*365*24*60*60);
?>
Próba usunięcia pliku cookie.
```

Usuwanie plików cookie (deletecookie.php)

OSTRZEŻENIE

Jeśli spróbujemy ustawić datę utraty ważności na 0, PHP pominie ten parametr, a zatem ten sposób nie zadziała. Trzeba użyć argumentu o dodatniej wartości — na przykład 1.

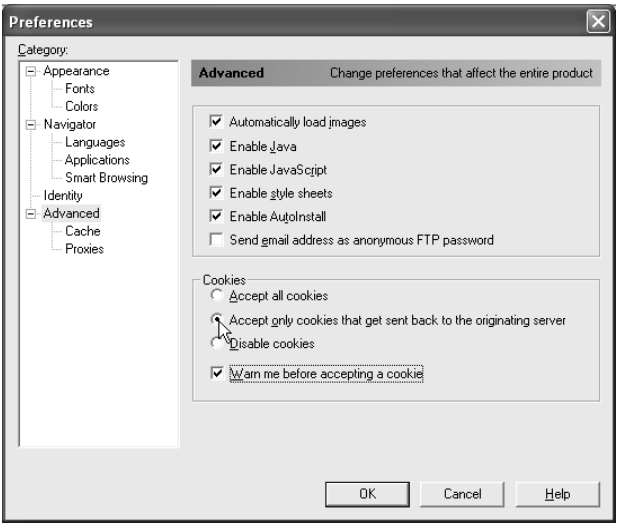
Udostępnianie plików cookie dla wielu domen

```
setcookie('version', phpversion(), 0,
'.przyklad.com');
```

Częścią nagłówka `Set-Cookie` wysyłanego przez serwer jest domena, która ma dostęp do tego pliku cookie. Jeśli wartość ta nie zostanie określona jawnie, domyślnie przyjmuje się domenę, z której wysłano plik cookie. Próba ustawienia tej wartości na całkiem inną domenę — na przykład serwera reklam (tzw. *pliki cookie firm zewnętrznych*, używane w celu wygenerowania profilu użytkownika) — nie zawsze zadziała, ponieważ wiele przeglądarek umożliwia zablokowanie takiej możliwości (na rysunku 5.5 widać, że taką możliwość miały już stare przeglądarki Netscape 4.x).

```
<?php
  setcookie('version', phpversion(), 0,
'.przyklad.com');
?>
Próba przesłania pliku cookie.
```

Ustawianie domeny dla pliku cookie (setcookie-domain.php)



Rysunek 5.5. Nawet w przeglądarce Netscape 4.x można zablokować pliki cookie, które pochodzą z innych domen

W niektórych przypadkach istnieje potrzeba, aby pliki cookie działały w kilku domenach zewnętrznych lub poddomenach, na przykład *www.przyklad.com*, *sklep.przyklad.com* i *ssl.przyklad.com*. Sytuacja dotyczy dużych witryn WWW z wieloma poddomenami, takich jak Amazon czy eBay. Witryny te wymagają obsługi dla wszystkich domen najwyższego poziomu. Aby to było możliwe, trzeba ustawić domenę pliku cookie — czwarty parametr funkcji `set-➥cookie()`. Jest jednak pewien kłopot: nazwy domen są prawidłowe, o ile zawierają dwie kropki. Jeśli zatem ustawimy domenę na *.przyklad.com*, do pliku cookie będą

miały dostęp wszystkie domeny trzeciego poziomu domeny *przyklad.com*. Jest jednak jedno „ale". Strony umieszczone w domenie *http://przyklad.com* nie będą miały dostępu do tego pliku cookie. Można zatem spróbować ustawić domenę na *przyklad.com*, ale to nie jest zgodne ze specyfikacją i może nie zadziałać w niektórych przeglądarkach.

Sprawdzanie, czy klient obsługuje pliki cookie

```
$test_temp = isset($_COOKIE['test_temp']) ?
  'obsługuje' : 'nie obsługuje';
$test_persist = isset($_COOKIE['test_persist']) ?
  'obsługuje' : 'nie obsługuje';
```

Pamiętacie, w jaki sposób są przesyłane pliki cookie? Najpierw serwer przesyła plik cookie do klienta w odpowiedzi HTTP. W następnym żądaniu, jeśli użytkownik zaakceptuje plik cookie, klient przesyła go z powrotem na serwer. Z tego powodu wywołanie funkcji `setcookie()` i późniejsze sprawdzenie zawartości tablicy `$_COOKIE` **nie** zadziała. Trzeba zaczekać na kolejne żądanie HTTP.

```
<?php
  if (isset($_GET['step']) && $_GET['step'] == '2') {
    $test_temp = isset($_COOKIE['test_temp']) ?
      'obsługuje' : 'nie obsługuje';
    $test_persist = isset($_COOKIE['test_persist']) ?
      'obsługuje' : 'nie obsługuje';
    setcookie('test_temp', '', time() - 365*24*60*60);
    setcookie('test_persist', '', time() -
    365*24*60*60);
```

```
    echo "Przeglądarka $test_temp tymczasowe pliki
    cookie.<br />";
    echo "Przeglądarka $test_persist trwałe pliki
    cookie.";
  } else {
    setcookie('test_temp', 'ok');
    setcookie('test_presist', 'ok', time() +
    14*24*60*60);
    header('Location: ' .
    nl2br($_SERVER['PHP_SELF']) . '?step=2');
  }
?>
```

Testowanie konfiguracji plików cookie przeglądarki (cookietest.php)

Można jednak skorzystać z funkcji `header()` i wymusić utworzenie drugiego żądania klienta. W pierwszym żądaniu ustawiamy plik cookie, w drugim sprawdzamy, czy operacja zakończyła się sukcesem.

W zaprezentowanym powyżej kodzie ustawiono dwa pliki cookie — jeden tymczasowy i drugi trwały — ponieważ niektóre przeglądarki można tak skonfigurować, że jeden typ plików cookie będzie akceptowany, a drugi nie. Następnie, aby sprawdzić ustawienie wszystkich plików cookie, wystarczy wykonać przekierowanie za pomocą funkcji `header()` oraz nagłówka HTTP `Location`.

Oczywiście trzeba tak skonfigurować przeglądarkę, aby w momencie przesłania pliku cookie był wyświetlany komunikat — znacznie ułatwia to debugowanie.

Zapisywanie wielu danych w jednym pliku cookie

```
setcookie('cookiedata', serialize($cookiedata)
```

Zazwyczaj w pliku cookie jest ustawiona jedna wartość — ciąg znaków. Z tego powodu, aby zapisać za pomocą plików cookie wiele danych, trzeba użyć wielu plików cookie. Takie działanie stwarza jednak pewne problemy — na przykład z powodu ograniczenia 20 plików cookie na domenę. Z tego względu w niektórych przypadkach ma sens zapisywanie wielu danych w jednym pliku cookie. Do tego przydają się tablice.

Wartościami plików cookie mogą być jednak wyłącznie ciągi znaków. Z tego powodu tablicę trzeba przekształcić na ciąg znaków za pomocą funkcji `serialize()`. Później można ją przekształcić z powrotem na tablicę za pomocą funkcji `unserialize()`.

```
<?php
  require_once 'stripCookieSlashes.inc.php';

  function setCookieData($arr) {
    $cookiedata = getAllCookieData();
    if ($cookiedata == null) {
      $cookiedata = array();
    }
    foreach ($arr as $name => $value) {
      $cookiedata[$name] = $value;
    }
    setcookie('cookiedata',
      serialize($cookiedata),
```

```
      time() + 30*24*60*60);
  }

  function getAllCookieData() {
    if (isset($_COOKIE['cookiedata'])) {
      $formdata = $_COOKIE['cookiedata'];
      if ($formdata != '') {
        return unserialize($formdata);
      } else {
        return array();
      }
    } else {
      return null;
    }
  }

  function getCookieData($name) {
    $cookiedata = getAllCookieData();
    if ($cookiedata != null &&
      isset($cookiedata[$name])) {
        return $cookiedata[$name];
      }
    }
    return '';
  }
?>
```

Biblioteka pomocnicza do zapisywania wielu danych w jednym pliku cookie (getCookieData.inc.php)

Do tego celu można napisać bibliotekę podobną do tej, którą wykorzystywaliśmy w poprzednim rozdziale do zapisywania danych formularza w pliku cookie. Jeden plik cookie o nazwie *cookiedata* zawiera wszystkie wartości zapisane w postaci tablicy asocjacyjnej. Funkcja `getCookie➥Data()` zwraca jedną wartość, natomiast `setCookieDate()` pobiera tablicę i zapisuje jej zawartość do pliku cookie. Kompletny kod źródłowy tej biblioteki pokazano na powyższym listingu.

Z kolei na listingu zamieszczonym poniżej wykorzystano zdefiniowaną bibliotekę do zaimplementowania testu dla plików cookie znanych z poprzedniego podrozdziału.

```
<?php
    require_once 'getCookieData.inc.php';

    if (isset($_GET['step']) && $_GET['step'] == '2')
    {
        $test = (getCookieData('test') == 'ok') ?
          'obsługuje' : 'nie obsługuje';
        echo "Przeglądarka $test pliki cookie.";
      } else {
      setCookieData(array('test' => 'ok'));
      header("Location:
 {$_SERVER['PHP_SELF']}?step=2");
    }
?>
```

Zapisywanie ustawień językowych użytkownika

```
setcookie('lang', $_SERVER['PHP_SELF'], time() +
30*24*60*60, '/')
```

Bardzo często strony WWW obsługują wiele języków. Zazwyczaj są one tak zorganizowane, że każda zlokalizowana część witryny jest zapisana w osobnym katalogu, na przykład w następującej konfiguracji:

- wersja dla języka angielskiego jest zapisana w katalogu *en*,
- wersja dla języka hiszpańskiego jest zapisana w katalogu *es*,

- wersja dla języka francuskiego jest zapisana w katalogu *fr*,
- wersja dla języka polskiego jest zapisana w katalogu *pl*.

Język użytkownika można wykryć na kilka sposobów:

- próbując powiązać adres IP klienta z określonym regionem geograficznym,
- odczytując nagłówek HTTP `Accept-Language`, z którego można się dowiedzieć, jakie języki są preferowane,
- zadając użytkownikowi pytanie.

Chociaż każda z tych metod jest w jakimś stopniu skuteczna, ostatnia (lub kombinacja kilku) jest uważana za najbardziej przyjazną dla użytkownika. Trzeba zatem zdefiniować stronę macierzystą zawierającą łącza do wszystkich dostępnych wersji językowych. Wystarczy prosty język HTML:

```
<a href="en/index.php">English version</a><br />
<a href="es/index.php">Versión espańol</a><br />
<a href="fr/index.php">Versione française</a><br />
<a href="pl/index.php">Wersja polska</a>
```

Strona macierzysta zawiera łącza do różnych wersji językowych (multilingual.php — fragment)

W katalogu dla każdego języka znajduje się skrypt *index.php* napisany w określonej wersji językowej. W tym pliku przechowywany jest kod z listingu zamieszczonego na początku tego podrozdziału. Instrukcja sprawdza, czy już ustawiono plik cookie dla wersji językowej, a jeśli nie — ustawia plik cookie dla bieżącego katalogu (odczytanego za pomocą odwołania do elementu `$_SERVER['PHP_SELF']`).

Zapisywanie ustawień językowych użytkownika

```
<?php
  if (!isset($_COOKIE['lang']) ||
    $_COOKIE['lang'] != $_SERVER['PHP_SELF']) {
    setcookie('lang', $_SERVER['PHP_SELF'],
      time() + 30*24*60*60, '/');
  }
?>
```

Zapisywanie informacji o bieżącym katalogu w pliku cookie (savelanguage.inc.php)

UWAGA

Jest bardzo istotne, aby ustawić ścieżkę pliku cookie na główny katalog serwera WWW. W innym przypadku wartością domyślną ścieżki będzie katalog bieżący i plik będzie mógł być odczytany tylko w katalogu bieżącym i jego podkatalogach. Nie będzie go można go odczytać ze strony macierzystej.

Na koniec, na stronie macierzystej trzeba sprawdzić, czy ustawiono plik cookie, a jeśli tak, przekierować użytkownika na stronę zawierającą odpowiednią wersję językową. W tym celu na początku skryptu *multilingual.php* trzeba wprowadzić następujący kod:

```
<?php
    if (isset($_COOKIE['lang']) && $_COOKIE['lang']
    != '') {
        header("Location: {$_COOKIE['lang']}");
    }
?>
```

Sesje

Sesja to nic innego jak wizyta użytkownika na stronie WWW. Użytkownik klika jakieś łącza, przegląda niektóre strony i wychodzi. Działania użytkownika w witrynie WWW definiują sesję. Jeśli użytkownik nie zażąda danych z witryny WWW przez pewien czas, na przykład 20 minut, sesja kończy się.

Protokół HTTP nie zawiera żadnego mechanizmu obsługi sesji, ponieważ jest bezstanowy. PHP zawiera jednak wbudowaną obsługę sesji, dzięki czemu skorzystanie z nich jest stosunkowo proste.

Po utworzeniu sesji PHP generuje jej identyfikator w postaci długiego ciągu znaków. Następnie tworzy zapis w pliku lub bazie danych dotyczący określonej sesji. Po wykonaniu tych operacji aplikacja PHP może zapisywać dane w sesji. Dane te trafiają do pliku sesji bądź do bazy danych (rzadziej dane sesji są zapisywane w pamięci współdzielonej).

Zatem jedynym elementem, który musi być przekazywany pomiędzy klientem a serwerem jest identyfikator sesji. Wszystkie pozostałe dane związane z sesją są przechowywane na serwerze. Dzięki temu w łączu nie są niepotrzebnie przesyłane istotne dane.

Konfiguracja mechanizmu sesji języka PHP jest w całości zapisana w sekcji `[session]` pliku konfiguracyjnego *php.ini*. Domyślne ustawienia nie pasują do wszystkich zastosowań, dlatego w kolejnych podrozdziałach opisano kilka możliwych konfiguracji.

Gdzie należy zapisywać sesje?

Zazwyczaj dane sesji są zapisywane w plikach. Lokalizację tych plików ustawia się za pomocą dyrektywy pliku php.ini `session.save_path`. Oczywiście katalog ustawiony za pomocą tej dyrektywy musi po pierwsze istnieć, a po drugie proces PHP (zwykle proces serwera WWW) musi mieć w nim uprawnienia odczytu i zapisu. W innym przypadku danych sesji nie można zapisać.

Jednak w przypadku dużej liczby użytkowników, a co za tym idzie dużej liczby sesji, interpreter PHP nie powinien umieszczać wszystkich plików sesji w jednym katalogu, ponieważ może to wpłynąć na znaczne obniżenie wydajności. Poniższa instrukcja umożliwia przenoszenie danych sesji do wielu podkatalogów:

```
session.save_path = "n;/tmp"
```

Skorzystanie z tej dyrektywy spowoduje utworzenie podkatalogów w katalogu */tmp* do 5 poziomów w głąb. Aby mechanizm sesji PHP mógł zapisywać dane w podkatalogach, trzeba je wcześniej utworzyć. Do tego celu służy skrypt *mod_files.sh* w katalogu *ext/sessions*.

Oczywiście prawo do odczytywania tego katalogu powinien posiadać tylko serwer WWW. W innym przypadku inni użytkownicy systemu mogliby odczytywać informacje o sesji potencjalnie zawierające wrażliwe dane.

W jaki sposób zachować stan sesji?

Identyfikator sesji musi być przesłany do przeglądarki w każdej odpowiedzi, a co ważniejsze — musi być przesłany z powrotem na serwer wraz z każdym żądaniem.

Najprościej można zaimplementować ten mechanizm za pomocą plików cookie. PHP przesyła do klienta plik cookie o nazwie `PHPSESSID` (nazwę można zmienić za pomocą dyrektywy pliku *php.ini* `session.name`). Aby było to możliwe, należy ustawić następującą dyrektywę pliku *php.ini*:

```
session.use_cookies = 1
```

Co jednak zrobić, jeśli użytkownik wyłączy obsługę plików cookie? W takim przypadku można skorzystać z innego mechanizmu. W tym celu należy ustawić następującą dyrektywę:

```
session.use_trans_sid = 0
```

W tym przypadku PHP automatycznie ustawia tryb, w którym identyfikator sesji jest dołączany do wszystkich adresów URL. Takie rozwiązanie może powodować pewne zagrożenie (na przykład modyfikowanie sesji — ang. *session fixation*, lub przechwytywanie sesji — ang. *session hijacking*), ale jest dość praktyczne. Wykorzystują je wszystkie znane witryny e-commerce, na przykład Amazon. Jeśli odwiedzimy ich witrynę WWW i załadujemy stronę, identyfikator sesji zostanie automatycznie dołączony na końcu adresu URL.

Aby można było skorzystać z ustawienia `session.user_trans_sid`, PHP musi być skompilowane z przełącznikiem `-enable-trans-sid`. Opcja ta jest automatycznie włączona dla binarnych dystrybucji przeznaczonych dla systemów Windows i Mac OS X.

Można również zezwolić na przekazywanie w adresach URL tylko plików cookie, bez identyfikatorów sesji. Do tego służy następująca dyrektywa pliku *php.ini*:

```
session.use_only_cookies = 1
```

UWAGA

Przekazywanie identyfikatorów sesji w adresach URL w zasadzie jest złe, ponieważ odwiedzający mogą zapisywać je w postaci zakładek, niektóre wyszukiwarki nie uwzględniają witryn, w których stosuje się tę technikę itd. W każdej witrynie e-commerce (a także w większości innych witryn) trzeba jednak wziąć pod uwagę, że niektórzy odwiedzający (potencjalni klienci!) nie włączą obsługi plików cookie. W takim przypadku sesje gwarantują wygodny sposób pokonania tego ograniczenia.

Aktywacja sesji

`session_start()`

Korzystanie z mechanizmów zarządzania sesjami zawsze wymaga zasobów, a w związku z tym wiąże się z obniżeniem wydajności. Dlatego właśnie sesje trzeba aktywować.

Ponieważ operacja ta może wymagać przesłania plików cookie, trzeba to zrobić przed wysłaniem do klienta zawartości HTML. Można to zrobić na dwa sposoby:

- globalna aktywacja sesji za pomocą dyrektywy pliku *php.ini* `session.auto_start = 1`,
- aktywacja sesji na poziomie skryptu, za pomocą funkcji `session_start()`.

Z punktu widzenia wydajności druga opcja jest lepsza.

```
<?php
  session_start();
  echo 'Sesje aktywowano.';
?>
```

Aktywacja sesji (session_start.php)

Czytanie i zapisywanie sesji

```
<?php
    session_start();
    echo 'Sesje uaktywniono.<br />';
    $_SESSION['version'] = phpversion();
    echo 'Dane sesji zapisano.<br />';
    echo "Dane sesji zostały odczytane:
{$_SESSION['version']} .";
?>
```

Dostęp do wszystkich danych sesji w skrypcie PHP można uzyskać za pomocą tablicy `$_SESSION`. Ponieważ dane są zapisane po stronie serwera, można zapisać dane sesji i odczytać je już w następnej instrukcji PHP bez konieczności transmisji, tak jak to było w przypadku plików cookie.

Wystarczy pamiętać o wywołaniu funkcji `session_start()`. Potem można korzystać z tablicy `$_SESSION`. Kod zamieszczony na listingu na początku tego podrozdziału tworzy plik sesji pokazany na rysunku 5.6.

Rysunek 5.6. Zawartość pliku sesji utworzonego przez kod pokazany na poprzednim listingu

Zamykanie sesji

`session_destroy()`

W niektórych przypadkach, na przykład kiedy użytkownik się wyloguje, wszystkie dane sesji należy usunąć, a sesję zamknąć. Oczywiście można przetworzyć tablicę `$_SESSION` w pętli `foreach` i ustawić każdą z wartości na pusty ciąg znaków, ale istnieje szybszy sposób — wystarczy wywołać funkcję `session_destroy()`. Po wykonaniu tej funkcji, tak jak wskazuje nazwa (*destroy* — niszczyć), wszystkie dane w bieżącej sesji zostaną zniszczone.

```
<?php
  session_start();
  echo 'Przed: <pre>';
  print_r($_SESSION);
  echo '</pre>Po: <pre> ';
  session_destroy();
```

```
  print_r($_SESSION);
  echo '</pre>';
?>
```

Usuwanie wszystkich danych sesji (session_destroy.php)

Modyfikacje identyfikatora sesji

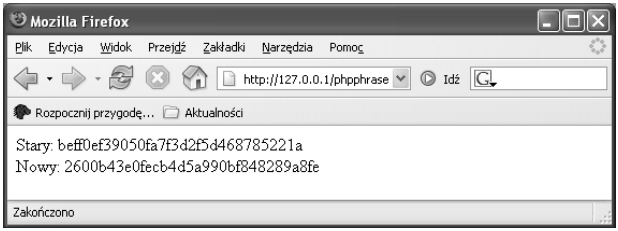
session_regenerate_id()

Jednym ze znanych ataków przeciwko witrynom WWW zabezpieczonych za pomocą sesji jest przechwycenie identyfikatora sesji użytkownika (na przykład w wyniku analizy zapisów HTTP_REFERER w żądaniach HTTP), a następnie wykorzystanie go w celu podszycia się pod niego. Z takimi atakami trudno się walczy, można jednak utrudnić życie napastnikom, zmieniając identyfikator sesji każdorazowo, kiedy coś ważnego się zmienia, na przykład gdy użytkownik się zaloguje. Przykładowo w witrynie Amazon użytkownicy, którzy zostali wcześniej uwierzytelnieni za pomocą pliku cookie, gdy chcą coś zamówić, muszą się ponownie zalogować.

```
<?php
  ob_start();
  session_start();
  echo 'Stary: ' . session_id();
  session_regenerate_id();
  echo '<br />Nowy: ' . session_id();
  ob_end_flush();
?>
```

Modyfikacja identyfikatora sesji (session_regenerate_id.php)

W tym przypadku funkcja `session_regenerate_id()` modyfikuje bieżący identyfikator sesji, ale pozostawia zapisane w niej dane bez zmian. Pokazano to w zamieszczonym wcześniej kodzie, gdzie za pomocą funkcji `session_id()` odczytano stary i bieżący identyfikator sesji. Przykładowy wynik działania tego skryptu pokazano na rysunku 5.7.



Rysunek 5.7. Dwa identyfikatory sesji: stary i nowy

UWAGA

W tym kodzie wykorzystano buforowanie wyników — funkcje `ob_start()` i `ob_end_flush()` — ponieważ funkcję `session_regenerate_id()` trzeba wywołać przed wysłaniem do klienta wyniku HTML.

Tworzenie dynamicznych łączy z obsługą sesji

```
$name = urlencode(session_name());
$id = urlencode(session_id());
echo "<a href=\"page.php?$name=$id\">dynamiczne
łącze</a>";
```

Zastosowanie ustawienia `session.use_trans_sid` w celu automatycznej aktualizacji wszystkich łączy w taki sposób, by zawierały identyfikatory sesji, to dobry pomysł, jednak sposób ten nie zadziała, jeśli łącza będą generowane automatycznie przez PHP. W PHP istnieją jednak dwie funkcje, które dostarczają wszystkich potrzebnych informacji:

- `session_name()` — zwraca nazwę sesji,
- `session_id()` — zwraca bieżący identyfikator sesji.

Tak więc kod zamieszczony na początku tego podrozdziału tworzy dynamiczne łącze zawierające informacje o sesji. W ten sposób programista może tworzyć dynamiczne łącza z obsługą sesji.

Oto przykładowy wynik działania zamieszczonego wyżej kodu:

```
<a href="page.php?PHPSESSID=2600b43e0fecb
4d5a990bf848289a8fe">dynamiczne łącze</a>
```

WSKAZÓWKA

Kiedy korzystamy z formularzy HTML, PHP automatycznie dołącza identyfikator sesji do atrybutu `action` formularza. Jednak, aby wykorzystać formularze dynamiczne, można dodać ukryte pole zawierające informacje o sesji:

```
<?php
    $name = htmlspecialchars(session_name());
    $id = htmlspecialchars(session_id());
    echo "<input type=\"hidden\" name=\"$name\"
    value=\"$id\" />";
?>
```

Implementacja własnego mechanizmu zarządzania sesjami

```
session_set_save_handler(
    'sess_open', 'sess_close', 'sess_read',
'sess_write', 'sess_destroy', 'sess_gc');
```

Istnieją powody, dla których nie powinno się zapisywać danych sesji w plikach. Wydajność to jedno, a bezpieczeństwo to drugie. Alternatywnie można zapisywać dane sesji w bazach danych. Aby można to było zrobić, należy utworzyć tabelę bazy danych *sessiondata* z trzema kolumnami (ich nazwy mogą być różne, jednak należy je konsekwentnie stosować w kodzie zaprezentowanym na listingach zamieszczonych poniżej):

- kolumna id (klucz główny) jest typu **`VARCHAR(32)`** i zawiera identyfikator sesji,
- kolumna `data` jest typu `TEXT` i zawiera dane sesji,
- kolumna `access` jest typu `VARCHAR(14)` i zawiera znacznik czasu ostatniego dostępu do danych sesji.

Nie ma znaczenia, jaką bazę danych wykorzystamy. W tym podrozdziale wykorzystano bazę MySQL i nowe rozszerzenie PHP5 — *mysqli*. Bez trudu można jednak tak zmodyfikować kod, aby działał również z innymi bazami danych (więcej informacji dotyczących wykorzystania różnych baz danych w PHP można znaleźć w rozdziale 7.).

Funkcję PHP `session_set_save_handler()` można wykorzystać do zdefiniowania własnych funkcji dla sześciu operacji wewnętrznie wykonywanych przez PHP:

- otwierania sesji,
- zamykania sesji,
- odczytywania zmiennej sesji,
- zapisywania zmiennej sesji,
- niszczenia sesji,
- porządkowania (tzw. odśmiecania — ang. *garbage collection* — polegającego, na przykład, na usuwaniu starych danych sesji z bazy danych).

Kod dla wszystkich tych sześciu operacji, który odczytuje i zapisuje dane sesji do i z bazy danych MySQL, zamieszczono na poniższym listingu. Aby z niego skorzystać, trzeba zainstalować PHP 5 i odpowiednio ustawić parametry połączenia (serwer, nazwę użytkownika, hasło). Następnie należy włączyć kod zamieszczony na poniższym listingu za pomocą funkcji `require_once`. Po wykonaniu tych operacji można wykorzystywać sesje w standardowy sposób. Od tego momentu PHP będzie zapisywał informacje dotyczące sesji w bazie danych, a nie w plikach.

```
<?php
  $GLOBALS['sess_server'] = 'localhost';
  $GLOBALS['sess_db'] = 'sessions';
  $GLOBALS['sess_username'] = 'user';
  $GLOBALS['sess_password'] = 'pass';

  function sess_open() {
```

Implementacja własnego mechanizmu zarządzania sesjami

```
    $GLOBALS['sess_mysqli'] = mysqli_connect(
      $GLOBALS['sess_server'],
      $GLOBALS['sess_username'],
      $GLOBALS['sess_password']
    );
    mysqli_select_db($GLOBALS['sess_mysqli'],
    $GLOBALS['sess_db']);
  }

  function sess_close() {
    mysqli_close($GLOBALS['sess_mysqli']);
  }

  function sess_read($id) {
    $result = mysqli_query(
      $GLOBALS['sess_mysqli'],
      sprintf('SELECT data FROM sessiondata WHERE id
       = \'%s\'',

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$id))
    );
    if ($row = mysqli_fetch_object($result)) {
      $ret = $row->data;
      mysqli_query(
        $GLOBALS['sess_mysqli'],
        sprintf('UPDATE sessiondata SET
        access=\'%s\' WHERE id=\'%s\'',
          date('YmdHis'),

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$id))
      );
    } else {
      $ret = '';
    }
    return $ret;
  }

  function sess_write($id, $data) {
    mysqli_query(
      $GLOBALS['sess_mysqli'],
```

```
      sprintf('UPDATE sessiondata SET data=\'%s\',
      access=\'%s\' WHERE id=\'%s\'',

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$data),
        date('YmdHis'),

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$id))
    );
    if
(mysqli_affected_rows($GLOBALS['sess_mysqli']) < 1)
{
      mysqli_query(
        $GLOBALS['sess_mysqli'],
        sprintf('INSERT INTO sessiondata (data,
       access, id) VALUES (\'%s\', \'%s\', \'%s\')',

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$data),
          date('YmdHis'),

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$id))
      );
    }
    return true;
  }

  function sess_destroy($id) {
    mysqli_query(
      $GLOBALS['sess_mysqli'],
      sprintf('DELETE FROM sessiondata WHERE
      id=\'%s\'',

mysqli_real_escape_string($GLOBALS['sess_mysqli'],
$id))
    );
    return true;
  }
  function sess_gc($timeout) {
    $timestamp = date('YmdHis', time() - $timeout);
```

```
    mysqli_query(
      $GLOBALS['sess_mysqli'],
      sprintf('DELETE FROM sessiondata WHERE access
      < \'%s\'',
        $timestamp)
    );
  }

  session_set_save_handler(
    'sess_open', 'sess_close', 'sess_read',
    'sess_write', 'sess_destroy', 'sess_gc');
?>
```

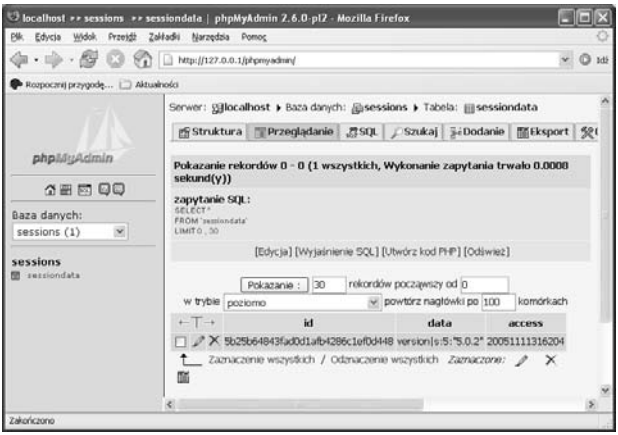
WSKAZÓWKA

W pliku *session.sql* w archiwum z przykładami znajdują się instrukcje SQL służące do utworzenia tabeli bazy danych MySQL. W skrypcie *session_mysqli_readwrite.php* wykorzystano kod z powyższego listingu w celu zapisania pewnych danych z wykorzystaniem sesji.

Na rysunku 5.8 pokazano zawartość tabeli *session* po zapisaniu do niej danych.

Tworzenie zabezpieczonego obszaru z wykorzystaniem sesji

```
session_start();
if (!(isset($_SESSION['authorized']) &&
  $_SESSION['authorized'] != '')) {
  header("Location:
login.php?url={$_SERVER['PHP_SELF']}");
}
```

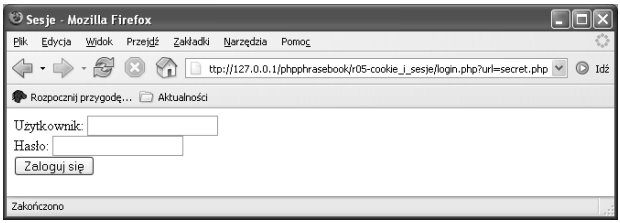


Rysunek 5.8. Dane sesji są teraz zapisane w bazie danych

Sesje to doskonały sposób zabezpieczenia wydzielonych części witryny WWW. Mechanizm jest prosty: po uwierzytelnieniu użytkownika należy zapisać tę informację w zmiennej sesji. Następnie na wszystkich zabezpieczonych stronach wystarczy sprawdzić, czy zdefiniowano tę zmienną.

Najpierw sprawdzimy, czy zdefiniowano zmienną sesji. Kod zamieszczony na początku tego podrozdziału należy włączyć (za pomocą instrukcji `require_once`) na wszystkich stronach, które mają być dostępne tylko dla uprawnionych użytkowników.

Skrypt *login.php*, do którego zaprezentowany wcześniej kod, przekierowuje użytkownika, zawiera formularz HTML (rysunek 5.9) i sprawdza, czy wprowadzone dane są prawidłowe (należy wprowadzić własnych użytkowników i ich



Rysunek 5.9. Formularz logowania — zwróćmy uwagę na stronę wymienioną w adresie URL

hasła). Jak można zobaczyć, adres URL jest podany jako parametr `GET`, a zatem, jeśli go zdefiniowano, kod skryptu logowania przekieruje użytkownika do strony, z której wysłano żądanie:

```
<?php
  if (isset($_POST['user']) && $_POST['user'] ==
  'Damon' &&
      isset($_POST['pass']) && $_POST['pass'] ==
      'secret') {
    session_start();
    $_SESSION['authorized'] = 'ok';
    $url = (isset($_GET['url'])) ?
    nl2br($_GET['url']) : 'index.php';
    header("Location: $url");
  }
?>
```

Sprawdzanie danych identyfikacyjnych użytkownika (login.php — fragment)

To wszystko! Przykładowy skrypt *secret.php* zawiera poufne informacje i jest chroniony przez kod zamieszczony na dwóch poprzednich listingach.

Tworzenie zabezpieczonego obszaru bez korzystania z sesji

```
$_SERVER['PHP_AUTH_USER'] == 'Shelley' &&
$_SERVER['PHP_AUTH_PW'] == 'TopSecret'
```

Jeśli wykorzystanie uwierzytelniania z mechanizmem zarządzania sesji PHP wydaje się zbyt dużym obciążeniem, można skorzystać z dwóch innych możliwości. Po pierwsze, można skonfigurować serwer WWW w taki sposób, aby tylko uwierzytelnieni użytkownicy mieli dostęp do niektórych plików lub katalogów. Na przykład użytkownicy serwera Apache mogą wykorzystać pliki *.htaccess*. Pod adresem *http://apache-server.com/tutorials/ATusing-htaccess.html* można znaleźć sporo informacji na ten temat. Serwer Microsoft IIS jest wyposażony w narzędzie z graficznym interfejsem użytkownika do definiowania praw dostępu, a zatem również dla tego serwera można zdefiniować podobny mechanizm.

```
<?php
if (!(isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']) &&
    $_SERVER['PHP_AUTH_USER'] == 'Shelley' &&
    $_SERVER['PHP_AUTH_PW'] == 'TopSecret')) {
    header('WWW-Authenticate: Basic realm="Secured
    area"');
  header('Status: 401 Unauthorized');
} else {
?>
```

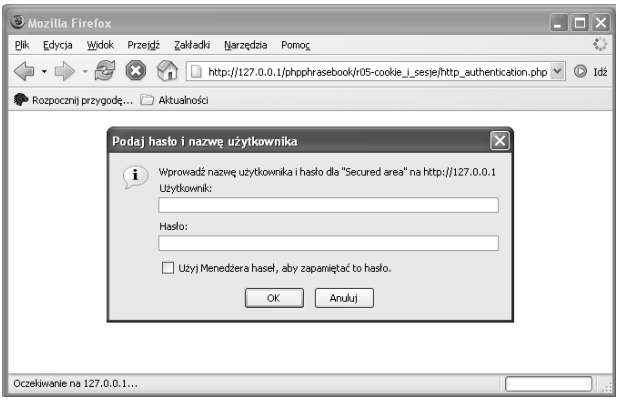
Tworzenie zabezpieczonego obszaru bez korzystania z sesji

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

Wykorzystanie HTTP do zabezpieczania stron PHP (http_authentication.php — fragment)

Jednym z rozwiązań, które w dużym stopniu jest niezależne od platformy, jest wykorzystanie uwierzytelniania HTTP. W przypadku przesłania kodu statusu HTTP 401 (nieuprawniony użytkownik) przeglądarka wyświetli pytanie o nazwę użytkownika i hasło. Informacje te są później dostępne w elementach tablicy superglobalnej `$_SERVER➥['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']`, ale jedynie wtedy, gdy PHP działa jako moduł serwera. Nie są natomiast dostępne w trybie CGI.

Po sprawdzeniu informacji w tablicy superglobalnej `$_➥SERVER` można zdecydować, czy ponownie wysyłać nagłówek 401 czy też wyświetlić zawartość strony. Implementację tego mechanizmu zaprezentowano na powyższym listingu. Okno dialogowe z pytaniem o nazwę użytkownika i hasło pokazano na rysunku 5.10.



Rysunek 5.10. Przeglądarka wyświetla pytanie o nazwę użytkownika i hasło

Co można znaleźć w repozytorium pear?

W repozytorium PEAR znajdują się następujące pakiety dotyczące sesji i uwierzytelniania HTTP:

- pakiet `Auth` zawiera zbiór funkcji służących do uwierzytelniania użytkowników, a tym samym do zabezpieczania stron PHP,
- pakiet `HTTP_Session` bazuje na mechanizmie obsługi sesji języka PHP, ale oferuje obiektowy dostęp do informacji dotyczących sesji.