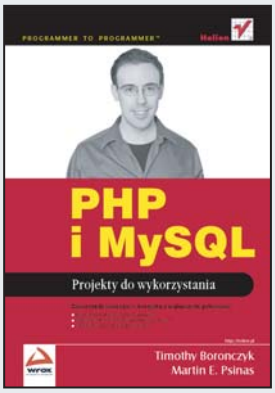
PHP i MySQL. Projekty do wykorzystania



Autorzy: Timothy Boronczyk, Martin E. Psinas
Tłumaczenie: Daniel Kaczmarek
ISBN: 978-83-246-2069-2
Tytuł oryginału: PHP and MySQL: Create - Modify - Reuse
Format: 172×245, stron: 360

Zaoszczędź swój czas – korzystaj z najlepszych gotowców!

- Korzystaj z najlepszych mechanizmów!
- Wzbogać stronę o praktyczne funkcjonalności!
- Szybko twórz profesjonalne serwisy!

Ile czasu zajmuje Ci przygotowanie formularza rejestracyjnego? Czy jest on wykorzystywany tylko raz? Popularność tandemu PHP-MySQL sprawiła, że mnóstwo powszechnie stosowanych mechanizmów ileś razy napisało wielu programistów. A wśród nich są i tacy, którzy te same mechanizmy tworzyli więcej niż raz! Czyż nie jest to klasyczny przykład marnotrawienia czasu?

Dzięki książce „PHP i MySQL. Projekty do wykorzystania" nie zmarnujesz już ani jednej cennej minuty. Stanowi ona zbiór najpopularniejszych mechanizmów, używanych na co dzień przy tworzeniu serwisów WWW. Dzięki niej łatwo (a co najważniejsze – szybko) zaimplementujesz funkcjonalność rejestracji użytkownika, listy dystrybucyjnej czy też wyszukiwarki. Dowiesz się, w jaki sposób stworzyć forum dyskusyjne, osobisty kalendarz, galerię zdjęć czy też menedżer plików, korzystający z technologii AJAX. Po przewertowaniu tego podręcznika nie będzie stanowiła dla Ciebie problemu rejestracja zdarzeń oraz wykonywanie skryptów powłoki. Pozwoli Ci to na szybkie tworzenie nowych serwisów WWW za pomocą sprawdzonych i elastycznych mechanizmów. Jeżeli cenisz swój czas – oto Twoja lektura obowiązkowa!

- Rejestracja użytkowników w serwisie
- Zabezpieczenie przed spamem – mechanizm CAPTCHA
- Implementacja forum dyskusyjnego
- Zastosowanie listy dystrybucyjnej
- Wyszukiwanie informacji w serwisie
- Tworzenie kalendarza
- Zarządzanie plikami – menedżer plików, korzystający z AJAX
- Prezentacja zdjęć – galeria online
- Statystyki serwisu WWW
- Rejestracja zdarzeń
- Wykonywanie skryptów powłoki

Nie trać czasu – korzystaj ze sprawdzonych projektów!

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

Spis treści

1

Rejestracja użytkowników

Umożliwienie rejestracji kont i logowania się przez użytkowników pozwala nadawać witrynom indywidualny charakter i udostępniać zawartość dostosowaną do konkretnych oczekiwań. Tego rodzaju mechanizm uwierzytelnienia jest centralnym punktem wielu witryn społecznościowych i e-commerce. Ze względu na tak dużą wagę mechanizmów uwierzytelniania pierwszą prezentowaną aplikacją jest system rejestracji użytkowników.

Główną funkcją systemu jest umożliwienie użytkownikom tworzenia kont. Członkowie systemu muszą podać adres poczty elektronicznej, który posłuży do weryfikacji poprawności rejestracji. Użytkownicy będą również mogli zmieniać hasła i uaktualniać adresy pocztowe, a także resetować hasła, gdy je zapomną. Są to całkowicie standardowe funkcje, oczekiwane przez użytkowników witryn internetowych.

Jeśli chodzi o architekturę, katalog przechowujący kod źródłowy powinien mieć logiczną strukturę. Na przykład pliki pomocnicze i dołączane powinny znajdować się w innym katalogu niż pliki dostępne publicznie. Ponadto dane na temat użytkowników powinny być przechowywane w bazie danych. Ponieważ na rynku dostępnych jest wiele narzędzi przeznaczonych do przeglądania i przetwarzania danych przechowywanych w relacyjnych bazach danych takich jak MySQL, łatwo jest zapewnić przezroczystość i elastyczność rozwiązania.

Plan struktury katalogów

W pierwszym kroku należy zaplanować strukturę katalogów aplikacji. Zaleca się utworzenie trzech głównych folderów: *public_files* będzie przechowywał wszystkie pliki dostępne publicznie, w *lib* przechowywane będą pliki dołączane, współużytkowane przez dowolną liczbę innych plików źródłowych, w *templates* zaś znajdą się pliki odpowiedzialne za prezentację stron. Choć PHP może się odwoływać do plików położonych w dowolnych lokalizacjach, serwer WWW powinien udostępniać wyłącznie pliki z folderu *public_files*. Przechowywanie plików pomocniczych poza katalogiem udostępnianym publicznie zwiększa poziom bezpieczeństwa witryny.

W folderze *public_files* zostanie utworzony folder *css* przechowujący katalogi stylów, folder *js* dla plików źródłowych JavaScript oraz *img* do przechowywania plików graficznych. Można utworzyć jeszcze dodatkowe foldery, aby zorganizować strukturę katalogów według własnych potrzeb. Dodatkowymi folderami mogą być na przykład *sql* do przechowywania plików serwera MySQL, *doc* dla dokumentacji systemu i dokumentów implementacyjnych oraz *tests* do przechowywania plików dla testów wstępnych i testów jednostkowych.

Plan struktury bazy danych

Oprócz zaplanowania struktury katalogów konieczne jest również pochylenie się nad strukturą bazy danych systemu. Zakres zapisywanych informacji na temat użytkowników będzie zależał od rodzaju usług świadczonych na witrynie. To z kolei będzie wyznaczać wygląd tabel bazy danych. Minimalnym wymaganiem jest, by w bazie danych przechowywać przynajmniej unikatowy identyfikator użytkownika, nazwę użytkownika, zaszyfrowane hasło i adres poczty elektronicznej. Trzeba będzie też zaimplementować funkcje sprawdzające, które konta zostały już zweryfikowane, a które dopiero oczekują na weryfikację.

```
DROP TABLE IF EXISTS HELION_PENDING;
DROP TABLE IF EXISTS HELION_USER;

CREATE TABLE HELION_USER (
    USER_ID    INTEGER UNSIGNED  NOT NULL  AUTO_INCREMENT,
    USERNAME   VARCHAR(20)       NOT NULL,
    PASSWORD   CHAR(40)          NOT NULL,
    EMAIL ADDR VARCHAR(100)      NOT NULL,
    IS_ACTIVE  TINYINT(1)        DEFAULT 0,

    PRIMARY KEY (USER_ID)
)
ENGINE=MyISAM DEFAULT CHARACTER SET latin1
    COLLATE latin1_general_cs AUTO_INCREMENT=0;

CREATE TABLE HELION_PENDING (
    USER ID       INTEGER UNSIGNED  NOT NULL,
    TOKEN         CHAR(10)          NOT NULL,
    CREATED DATE  TIMESTAMP         DEFAULT  CURRENT TIMESTAMP,

    FOREIGN KEY (USER ID)
        REFERENCES HELION_USER(USER_ID)
)
ENGINE=MyISAM DEFAULT CHARACTER SET latin1
    COLLATE latin1_general_cs;
```

W tabeli `HELION_USER` na przechowywanie zaszyfrowanego hasła przewidziano kolumnę o szerokości 40 znaków, ponieważ do szyfrowania haseł używana będzie funkcja `sha1()`, zwracająca właśnie 40-znakowy szesnastkowy ciąg znaków. Nigdy nie powinno się przechowywać w bazie danych haseł w oryginalnej postaci — jest to podstawowa zasada bezpieczeństwa. Zasadą działania zastosowanego rozwiązania jest zaszyfrowanie hasła, gdy zostanie ono podane przez użytkownika po raz pierwszy. To samo hasło wpisywane później też podlega szyfrowaniu, a wynik szyfrowania funkcją `sha1()` jest porównywany z zaszyfrowanym hasłem przechowywanym w bazie danych.

Jako maksymalną długość ciągu znaków przechowującego adres poczty elektronicznej wyznaczono 100 znaków. Z technicznego punktu widzenia obecnie obowiązujące standardy pozwalają na definiowanie adresów pocztowych o maksymalnej długości 320 znaków (64 znaki na nazwę użytkownika, 1 znak na symbol `@` i 255 znaków na nazwę komputera). Trudno jednak znaleźć kogokolwiek, kto używałby tak długiego adresu pocztowego, dlatego w schematach baz danych adresy pocztowe standardowo przechowuje się w kolumnach o szerokości 100 znaków.

W bazie danych można by dodatkowo przechowywać imię i nazwisko użytkownika, jego adres, miasto zamieszkania, województwo, kod pocztowy, numery telefonów i tak dalej.

W tabeli `HELION_PENDING` znajduje się inicjalizowana automatycznie kolumna znacznika czasu. Dzięki temu w dowolnym momencie można usunąć z bazy wszystkie zarejestrowane konta użytkownika, które przez określony czas nie doczekały się weryfikacji. Kolumny tabeli można by połączyć z kolumnami tabeli `HELION_USER`, jednak ze względu na to, że znacznik wskazujący konieczność weryfikacji konta jest używany tylko jeden raz, zdecydowano się na ich wydzielenie do odrębnej tabeli. Dane użytkowników są przechowywane znacznie dłużej, a dzięki zastosowanemu rozwiązaniu tabela `HELION_USER` nie jest zaśmiecana danymi tymczasowymi.

Kod współużytkowany

Kod, który jest współużytkowany przez większą liczbę plików, powinien zostać wyłączony z pliku docelowego i dołączony do niego przy użyciu instrukcji `include` lub `require`. Dzięki temu ten sam kod nie będzie duplikowany i łatwiej będzie utrzymywać aplikację. Tam, gdzie to możliwe, kod potencjalnie przydatny w przyszłych aplikacjach (taki jak funkcje albo klasy) powinien być przechowywany oddzielnie. Dobrym założeniem jest pisanie kodu z myślą o tym, by móc go ponownie wykorzystać w przyszłości. Plik *common.php* zawiera kod współużytkowany, który będzie dołączany do innych skryptów aplikacji, aby ustanowić w ten sposób jednolite środowisko fazy wykonania. Kod tego rodzaju nigdy nie powinien być wywoływany przez użytkowników bezpośrednio, dlatego należy go umieścić w katalogu *lib*.

```
<?php
// true, jeśli środowisko produkcyjne; w przeciwnym razie false
define ('IS_ENV_PRODUCTION', true);

// ustawienie opcji raportowania błędów
error_reporting(E_ALL | E_STRICT);
ini_set('display_errors', !IS_ENV_PRODUCTION);
ini_set('error_log', 'log/phperror.txt');

// ustawienie strefy czasowej, by uniknąć ostrzeżeń
// w przypadku użycia funkcji czasu i daty
date_default_timezone_set('Europe/Warsaw');

// uwzględnienie „magic quotes" w razie konieczności
if (get_magic_quotes_gpc())
{
    function _stripslashes_rcurs($variable, $top = true)
    {
        $clean_data = array();
```

```
        foreach ($variable as $key => $value)
        {
            $key = ($top) ? $key : stripslashes($key);
            $clean_data[$key] = (is_array($value)) ?
                stripslashes rcurs($value, false) : stripslashes($value);
        }
        return $clean data;
    }
    $_GET = _stripslashes_rcurs($_GET);
    $_POST = _stripslashes_rcurs($_POST);
    // $_REQUEST = _stripslashes_rcurs($_REQUEST);
    // $_COOKIE = _stripslashes_rcurs($_COOKIE);
}
?>
```

Nie zawsze ma się kontrolę nad konfiguracją używanego serwera, dlatego dobrze jest zdefiniować kilka podstawowych dyrektyw, dzięki którym przenoszenie aplikacji będzie znacznie łatwiejsze. Na przykład zdefiniowanie opcji raportowania błędów pozwoli na wyświetlanie błędów w środowisku rozwojowym lub przekierowanie ich w środowisku produkcyjnym, tak by wewnętrzne komunikaty o błędach nie były widoczne dla użytkownika.

Magiczne apostrofy (ang. *magic quotes*) to opcja konfiguracyjna, dzięki której PHP może automatycznie poprzedzać znakami ucieczki symbole apostrofu, cudzysłowu i ukośników odwrotnych zawarte w danych wejściowych. Funkcja ta może się wydawać przydatna, jednak nigdy z góry nie powinno się przyjmować założenia, że na danym serwerze jest ona włączona lub nie, ponieważ może to doprowadzić do kłopotów. Lepiej jest najpierw znormalizować dane, a następnie w razie konieczności poprzedzać je znakami ucieczki przy użyciu funkcji `addslashes()` lub `mysql_real_escape_string()` (jeżeli dane mają być przechowywane w bazie danych, zalecane jest zastosowanie drugiej ze wspomnianych funkcji). Zastępowanie magicznych apostrofów zapewni, że znaki ucieczki zostaną zastosowane w danych odpowiednio i we właściwym momencie, bez względu na sposób konfiguracji PHP. Dzięki temu dalsza implementacja będzie prostsza i zmniejszy się zagrożenie popełnieniem błędów.

Ustanawianie połączenia z serwerem bazy danych MySQL jest czynnością wykonywaną powszechnie, dlatego warto przenieść wykonujący ją kod do oddzielnego pliku. Plik o nazwie *db.php* przechowuje stałe konfiguracyjne oraz kod zestawiający połączenie z bazą danych. Również ten plik ma być dołączany do innych plików, a nie należy go wywoływać w sposób bezpośredni, dlatego należy zapisać go w katalogu *lib*.

```
<?php
// stałe bazy danych i schematów
define('DB_HOST', 'localhost');
define('DB_USER', 'użytkownik');
define('DB_PASSWORD', 'hasło');
define('DB_SCHEMA', 'HELION_DATABASE');
define('DB_TBL_PREFIX', 'HELION_');

// ustanowienie połączenia z serwerem bazy danych
if (!$GLOBALS['DB'] = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD))
{
    die('Błąd: Nie udało się nawiązać połączenia z bazą danych.');
}
if (!mysql_select_db(DB_SCHEMA, $GLOBALS['DB']))
{
```

```
    mysql_close($GLOBALS['DB']);
    die('Błąd: Nie udało się wybrać schematu bazy danych.');
}
?>
```

Stałe `DB_HOST`, `DB_USER`, `DB_PASSWORD` i `DB_SCHEMA` reprezentują wartości niezbędne do skutecznego zestawienia połączenia z bazą danych. Jeżeli kod zostanie przeniesiony do środowiska produkcyjnego, w którym serwer bazy danych pracuje na innym komputerze niż PHP i serwer WWW, można wówczas dodatkowo zdefiniować wartość `DB_PORT` i odpowiednio zmodyfikować wywołanie funkcji `mysql_connect()`.

Uchwyt połączenia z bazą danych jest następnie zapisywany w tablicy superglobalnej `$GLOBALS`, aby stał się dostępny w dowolnym zasięgu każdego pliku dołączającego plik *db.php* (albo dołączanego do pliku, który odwołuje się do *db.php*).

Dzięki poprzedzeniu nazw tabel prefiksami można uniknąć konfliktu z tabelami używanymi przez inne aplikacje, przechowywanymi w tym samym schemacie. Ponadto, jeżeli prefiks zostanie zdefiniowany w postaci stałej, łatwiej będzie uaktualnić kod źródłowy później, gdy zajdzie konieczność zmiany prefiksu, ponieważ będzie on definiowany tylko w jednym miejscu.

Wspólne funkcje również można umieszczać w oddzielnym, przeznaczonym dla nich pliku. W projekcie użyta zostanie funkcja `random_text()`, której zadaniem będzie wygenerowanie ciągu znaków CAPTCHA i znacznika weryfikacji. Funkcję `random_text()` można zatem zapisać w pliku *functions.php*.

```
<?php
// zwrócenie ciągu losowych znaków o określonej długości
function random text($count, $rm similar = false)
{
    // utworzenie listy znaków
    $chars = array_flip(array_merge(range(0, 9), range('A', 'Z')));

    // usunięcie podobnie wyglądających znaków, aby uniknąć pomyłek
    if ($rm_similar)
    {
        unset($chars[0], $chars[1], $chars[2], $chars[5], $chars[8],
            $chars['B'], $chars['I'], $chars['O'], $chars['Q'],
            $chars['S'], $chars['U'], $chars['V'], $chars['Z']);
    }

    // wygenerowanie ciągu losowych znaków
    for ($i = 0, $text = ''; $i < $count; $i++)
    {
        $text .= array_rand($chars);
    }

    return $text;
}
?>
```

Bez względu na to, w jakim języku implementuje się kod źródłowy, zawsze trzeba przestrzegać podstawowej zasady, by nigdy nie ufać danym wpisywanym przez użytkowników. Użytkownicy mogą (i będą) wpisywać wszelkiego rodzaju bezsensowne i niezrozumiałe dane. Czasami przez przypadek, niekiedy jednak jest to działanie zamierzone. Za pomocą funkcji

`filter_input()` i `filter_var()` języka PHP można oczyścić dane wejściowe, jednak niektórzy programiści wciąż wolą implementować własne procedury, ponieważ rozszerzenie udostępniające filtry może nie być dostępne w wersjach PHP wcześniejszych niż 5.2.0. Kod źródłowy tego rodzaju własnych procedur również warto umieścić w pliku *functions.php*.

Klasa User

Znakomitą większość kodu utrzymującego konta użytkowników można zawrzeć w ramach jednej struktury danych, którą będzie można później rozszerzać albo ponownie wykorzystywać w kolejnych aplikacjach. Struktura będzie implementować logikę interakcji z bazą danych, a przez to ułatwiać operacje zapisywania i odczytywania danych. Poniżej przedstawiono zawartość pliku *User.php*.

```
<?php
class User
{
    private $uid;      // identyfikator użytkownika
    private $fields;   // inne pola rekordu

    // inicjalizacja obiektu User
    public function   construct()
    {
        $this->uid = null;
        $this->fields = array('username' => '',
                              'password' => '',
                              'emailAddr' => '',
                              'isActive' => false);
    }

    // nadpisanie metody odczytującej właściwości
    public function __get($field)
    {
        if ($field == 'userId')
        {
            return $this->uid;
        }
        else
        {
            return $this->fields[$field];
        }
    }

    // nadpisanie metody ustawiającej właściwości
    public function __set($field, $value)
    {
        if (array_key_exists($field, $this->fields))
        {
            $this->fields[$field] = $value;
        }
    }
```

```
// sprawdzenie, czy nazwa użytkownika ma właściwy format
public static function validateUsername($username)
{
    return preg_match('/^[A-Z0-9]{2,20}$/i', $username);
}

// sprawdzenie, czy adres e-mail ma właściwy format
public static function validateEmailAddr($email)
{
    return filter_var($email, FILTER_VALIDATE_EMAIL);
}

// zwrócenie obiektu wypełnionego na podstawie identyfikatora użytkownika
public static function getById($uid)
{
    $u = new User();
    $query = sprintf('SELECT USERNAME, PASSWORD, EMAIL_ADDR, IS_ACTIVE ' .
        'FROM %sUSER WHERE USER_ID = %d', DB_TBL_PREFIX, $uid);
    $result = mysql_query($query, $GLOBALS['DB']);

    if (mysql_num_rows($result))
    {
        $row = mysql fetch assoc($result);
        $u->username = $row['USERNAME'];
        $u->password = $row['PASSWORD'];
        $u->emailAddr = $row['EMAIL ADDR'];
        $u->isActive = $row['IS ACTIVE'];
        $u->uid = $uid;
    }
    mysql_free_result($result);

    return $u;
}

// zwrócenie obiektu wypełnionego na podstawie nazwy użytkownika
public static function getByUsername($username)
{
    $u = new User();

    $query = sprintf('SELECT USER_ID, PASSWORD, EMAIL_ADDR, IS_ACTIVE ' .
        'FROM %sUSER WHERE USERNAME = "%s"', DB_TBL_PREFIX,
        mysql_real_escape_string($username, $GLOBALS['DB']));
    $result = mysql query($query, $GLOBALS['DB']);

    if (mysql num rows($result))
    {
        $row = mysql_fetch_assoc($result);
        $u->username = $username;
        $u->password = $row['PASSWORD'];
        $u->emailAddr = $row['EMAIL_ADDR'];
        $u->isActive = $row['IS_ACTIVE'];
        $u->uid = $row['USER_ID'];
    }

    mysql free result($result);
    return $u;
}
```

```
    // zapisanie rekordu w bazie danych
    public function save()
    {
        if ($this->uid)
        {
            $query = sprintf('UPDATE %sUSER SET USERNAME = "%s", ' .
                'PASSWORD = "%s", EMAIL_ADDR = "%s", IS_ACTIVE = %d ' .
                'WHERE USER_ID = %d', DB_TBL_PREFIX,
                mysql_real_escape_string($this->username, $GLOBALS['DB']),
                mysql_real_escape_string($this->password, $GLOBALS['DB']),
                mysql_real_escape_string($this->emailAddr, $GLOBALS['DB']),
                $this->isActive, $this->userId);
            mysql query($query, $GLOBALS['DB']);
        }
        else
        {
            $query = sprintf('INSERT INTO %sUSER (USERNAME, PASSWORD, ' .
                'EMAIL_ADDR, IS_ACTIVE) VALUES ("%s", "%s", "%s", %d)',
                DB_TBL_PREFIX,
                mysql_real_escape_string($this->username, $GLOBALS['DB']),
                mysql_real_escape_string($this->password, $GLOBALS['DB']),
                mysql_real_escape_string($this->emailAddr, $GLOBALS['DB']),
                $this->isActive);
            mysql query($query, $GLOBALS['DB']);

            $this->uid = mysql insert id($GLOBALS['DB']);
        }
    }

    // oznaczenie rekordu jako nieaktywnego i zwrócenie znacznika aktywacji
    public function setInactive()
    {
        $this->isActive = false;
        $this->save(); // zapewnienie, że rekord jest zapisany

        $token = random_text(5);
        $query = sprintf('INSERT INTO %sPENDING (USER_ID, TOKEN) ' .
            'VALUES (%d, "%s")', DB_TBL_PREFIX, $this->uid, $token);
        mysql_query($query, $GLOBALS['DB']);

        return $token;
    }

    // wyczyszczenie tymczasowego statusu użytkownika i oznaczenie rekordu jako aktywnego
    public function setActive($token)
    {
        $query = sprintf('SELECT TOKEN FROM %sPENDING WHERE USER ID = %d ' .
            'AND TOKEN = "%s"', DB_TBL_PREFIX, $this->uid,
            mysql real escape string($token, $GLOBALS['DB']));
        $result = mysql query($query, $GLOBALS['DB']);

        if (!mysql num rows($result))
        {
            mysql_free_result($result);
            return false;
        }
        else
        {
```

```
            mysql_free_result($result);
            $query = sprintf('DELETE FROM %sPENDING WHERE USER_ID = %d ' .
                'AND TOKEN = "%s"', DB_TBL_PREFIX, $this->uid,
                mysql_real_escape_string($token, $GLOBALS['DB']));
            mysql_query($query, $GLOBALS['DB']);

            $this->isActive = true;
            $this->save();
            return true;
        }
    }
}
?>
```

W klasie zdefiniowano dwie właściwości prywatne: `$uid`, która odpowiada kolumnie `USER_ID` tabeli `HELION_USER`, oraz tablicę `$fields`, która odpowiada pozostałym kolumnom. Obydwie właściwości są udostępniane intuicyjnie, poprzez nadpisanie metod `__get()` i `__set()`. Jednak właściwość `$uid` jest nadal chroniona przed przypadkową zmianą.

Statyczne metody `getById()` i `getByUsername()` zawierają kod odpowiedzialny za odczytywanie rekordu z bazy danych i wypełnianie obiektu danymi. Metoda `save()` zapisuje rekord w bazie danych i jest na tyle inteligentna, że rozpoznaje, kiedy należy wykonać zapytanie `INSERT`, a kiedy zapytanie `UPDATE`, zależnie od tego, czy ustawiony jest identyfikator użytkownika. W celu utworzenia nowego konta użytkownika wystarczy stworzyć nową instancję obiektu `User`, zdefiniować wartości pól w rekordzie i wywołać metodę `save()`.

```
<?php
$u = new User();
$u->username = 'timothy';
$u->password = sha1('sekret');
$u->emailAddr = 'timothy@helion.pl';
$u->save();
?>
```

W taki sam sposób przebiega czynność zmiany danych konta. Najpierw odczytywane są dane aktualne, następnie w danych wprowadzane są zmiany, po czym następuje ich zapisanie w bazie danych przez metodę `save()`.

```
<?php
$u = User::getByUsername('timothy');
$u->password = sha1('nowe_hasło');
$u->save();
?>
```

Metody `setInactive()` i `setActive()` obsługują proces aktywacji konta. W wyniku wywołania metody `setInactive()` konto zostaje oznaczone jako nieaktywne, następuje wygenerowanie znacznika aktywacji, informacja o tym fakcie zostaje zapisana w bazie danych i znacznik jest zwracany. Gdy użytkownik uaktywni konto, znacznik aktywacji jest przekazywany do metody `setActive()`. Metoda `setActive()` usuwa rekord ze znacznikiem aktywacji i oznacza konto jako aktywne.

CAPTCHA

Wyrażenie CAPTCHA to skrót od angielskich słów *Completely Automated Public Turing Test to Tell Computers and Humans Apart*, co w wolnym tłumaczeniu może oznaczać Całkowicie Zautomatyzowany Publiczny Test Turinga Wskazujący Komputerom i Ludziom, by Trzymali się z Daleka. CAPTCHA, oprócz tego, że jest trudnym do rozszyfrowania akronimem, często bywa używany jako narzędzie powstrzymującego spamerów i innych złośliwych użytkowników przed automatycznym rejestrowaniem kont użytkowników.

W tym celu użytkownikowi stawia się zadanie, które często ma postać obrazka zawierającego litery i cyfry. Użytkownik musi odczytać z niego tekst i przepisać do pola tekstowego. Jeżeli obydwie wartości są identyczne, można założyć, że system ma do czynienia z inteligentną istotą ludzką, a nie z komputerem próbującym automatycznie założyć konto w systemie.

Nie jest to jednak rozwiązanie idealne. CAPTCHA może sprawiać problemy osobom z wadami wzroku, a poza tym niektóre programy potrafią już odczytywać tekst zawarty na obrazkach CAPTCHA (więcej na ten temat pod adresem *www.cs.sfu.ca/~mori/research/gimpy/*). Zadania CAPTCHA stawiane przed użytkownikami mogą mieć także inną postać. Istnieją na przykład zadania CAPTCHA w postaci dźwiękowej — użytkownik musi wówczas wpisywać litery i cyfry usłyszane po odtworzeniu pliku audio. Niektóre zadania mają nawet postać prostych zadań matematycznych.

Zadania CAPTCHA należy traktować jako jedno z tych narzędzi w arsenale administratorów, które służą do odstraszania leniwych złoczyńców, nie powinny natomiast zastępować standardowych metod monitorowania i zabezpieczeń. Niedogodności dla użytkowników wzrastają wraz ze stopniem skomplikowania zadania CAPTCHA, dlatego w tym projekcie ograniczymy się do najprostszego przykładu, polegającego na wykorzystaniu obrazka.

```
<?php
include '../../lib/functions.php';

// należy utworzyć lub kontynuować sesję i zapisać ciąg znaków CAPTCHA
// w $_SESSION, by był dostępny w ramach innych wywołań
if (!isset($_SESSION))
{
    session_start();
    header('Cache-control: private');
}

// utworzenie obrazka o wymiarach 65x20 pikseli
$width = 65;
$height = 20;
$image = imagecreate(65, 20);

// wypełnienie obrazka kolorem tła
$bg_color = imagecolorallocate($image, 0x33, 0x66, 0xFF);
imagefilledrectangle($image, 0, 0, $width, $height, $bg_color);

// pobranie losowego tekstu
$text = random_text(5);
```

```
// ustalenie współrzędnych x i y do wyśrodkowania tekstu
$font = 5;
$x = imagesx($image) / 2 - strlen($text) * imagefontwidth($font) / 2;
$y = imagesy($image) / 2 - imagefontheight($font) / 2;

// wypisanie tekstu na obrazku
$fg color = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
imagestring($image, $font, $x, $y, $text, $fg_color);

// zapisanie ciągu znaków CAPTCHA do późniejszego porównania
$_SESSION['captcha'] = $text;

// zwrócenie obrazka
header('Content-type: image/png');
imagepng($image);

imagedestroy($image);
?>
```

Skrypt najlepiej jest zapisać w folderze *public_files/img* (ponieważ musi on być publicznie dostępny i zwracać obrazek graficzny), w pliku o nazwie *captcha.php*. Skrypt tworzy obrazek PNG o wymiarach 65 na 20 pikseli, z tłem koloru niebieskiego oraz białym, losowym ciągiem pięciu znaków, jak na rysunku 1.1. Ciąg znaków musi być przechowywany w zmiennej `$_SESSION`, aby nieco później można było sprawdzić, czy użytkownik przepisał go prawidłowo. Aby bardziej skomplikować obrazek, można użyć różnych czcionek, kolorów oraz zastosować obrazki w tle.

Rysunek 1.1

Szablony

Dzięki szablonom programistom łatwiej jest utrzymywać spójny wygląd i układ poszczególnych stron witryny. Szablony nadają organizację kodu oraz przenoszą logikę prezentacji poza właściwy kod źródłowy, dzięki czemu pliki PHP **oraz** HTML stają się bardziej czytelne. Na rynku dostępnych jest wiele rozwiązań do obsługi szablonów — niektóre rozbudowane (na przykład Smarty: *http://smarty.php.net*), inne niepozorne (TinyButStrong: *www.tinybut*
➥*strong.com*). Każde z tych rozwiązań ma własne wady i zalety, bez względu na to, czy jest to produkt komercyjny o otwartym dostępie do kodu źródłowego, czy tworzony na użytek domowy. W wielu przypadkach ostateczna decyzja o tym, którego z tych rozwiązań użyć, jest podejmowana na podstawie własnych upodobań programisty.

Jeśli chodzi o osobiste upodobania, można całym sercem popierać sam pomysł wykorzystania szablonów, a jednocześnie nie przepadać za większością implementacji tej idei. Pomimo niewątpliwych zalet obecnie dostępne rozwiązania do obsługi szablonów wiele rzeczy komplikują. W niektórych stosowana jest ich własna, specjalna składnia, której trzeba się nauczyć; poza tym praktycznie wszystkie wydłużają proces przetwarzania kodu. Prawdę mówiąc, w większości projektów wykorzystanie oddzielnego mechanizmu obsługi szablonów nie jest w ogóle potrzebne, ponieważ PHP sam może być uważany za moduł obsługi szablonów,

zdolny obsługiwać nawet witryny o średniej wielkości, tworzone przez większą liczbę programistów. Wystarczy zastosować w takich przypadkach odpowiednie planowanie i organizację prac.

Rozwiązanie, które wydaje się najlepsze, polega na wydzieleniu najważniejszych elementów prezentacji w plikach HTML w folderze *templates*. Folder ten zwykle umieszczany jest poza folderem dostępnym publicznie (choć pliki CSS, JavaScript i graficzne przywoływane w kodzie HTML muszą już być publicznie dostępne), aby uniknąć sytuacji, w której użytkownik lub wyszukiwarka znajduje pliki w praktyce pozbawione treści.

Poniżej przedstawiono podstawowy szablon, spełniający wymagania naszego projektu.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2" />
  <title>
<?php
if (!empty($GLOBALS['TEMPLATE']['title']))
{
    echo $GLOBALS['TEMPLATE']['title'];
}
?>
</title>
  <link rel="stylesheet" type="text/css" href="css/styles.css"/>
<?php
if (!empty($GLOBALS['TEMPLATE']['extra_head']))
{
    echo $GLOBALS['TEMPLATE']['extra_head'];
}
?>
 </head>
 <body>
  <div id="header">
<?php
if (!empty($GLOBALS['TEMPLATE']['title']))
{
    echo $GLOBALS['TEMPLATE']['title'];
}
?>
  </div>
  <div id="content">
<?php
if (!empty($GLOBALS['TEMPLATE']['content']))
{
    echo $GLOBALS['TEMPLATE']['content'];
}
?>
  </div>
  <div id="footer">Copyright &#169;<?php echo date('Y'); ?></div>
  </div>
 </body>
</html>
```

Zgodnie z powszechnie przyjętymi zasadami należy ustalić pewne konwencje. Treść będzie przechowywana w tablicy `$GLOBALS` w skrypcie wywołującym, dzięki czemu będzie również dostępna w dowolnym zasięgu wewnątrz dołączonego pliku szablonu. Zwykle używane są następujące klucze:

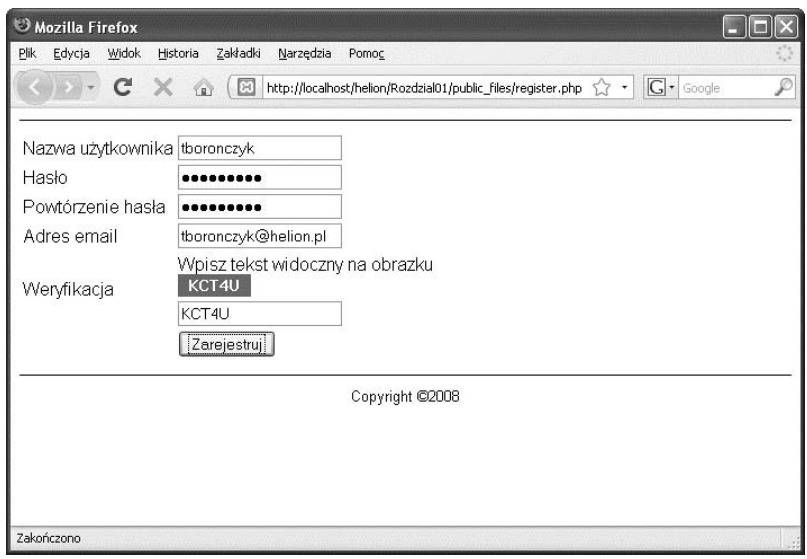
- `title` — Tytuł strony.
- `description` — Opis strony.
- `keywords` — Słowa kluczowe dla strony (tytuł strony, opis i słowa kluczowe mogą być przechowywane w bazie danych).
- `extra_head` — Klucz do wstawiania dodatkowych nagłówków HTML lub kodu JavaScript do kodu strony.
- `content` — Główna treść strony.

Czasami używa się również kluczy dla menu lub ramek, zależnie od układu, jaki zaplanowano dla strony. Za każdym razem jednak konkretne nazwy zmiennych będą zależeć od szablonu. Jeśli tylko zdefiniuje się i zapisze standardowe konwencje, które potem będą konsekwentnie stosowane, zespół implementacyjny dowolnej wielkości może z powodzeniem wykorzystywać tak opracowany mechanizm obsługi szablonów.

Rejestracja nowego użytkownika

Zdefiniowano już strukturę katalogów, zaimplementowano także odpowiednią część kodu pomocniczego, można się więc teraz skupić na rejestrowaniu nowego użytkownika. Kod źródłowy przedstawiony poniżej można zapisać w folderze *public_files*, w pliku o nazwie *register.php*. Na rysunku 1.2 przedstawiono tę samą stronę wyświetloną w przeglądarce.

Rysunek 1.2



```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';
include '../lib/db.php';
include '../lib/functions.php';
include '../lib/User.php';

// rozpoczęcie lub kontynuacja sesji, by udostępnić
// test CAPTCHA przechowywany w zmiennej $_SESSION
session_start();
header('Cache-control: private');

// przygotowanie formularza HTML do rejestracji
ob_start();
?>
<form method="post"
 action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
 <table>
  <tr>
   <td><label for="username">Nazwa użytkownika</label></td>
   <td><input type="text" name="username" id="username"
    value="<?php if (isset($ POST['username']))
    echo htmlspecialchars($_POST['username']); ?>"/></td>
  </tr><tr>
   <td><label for="password1">Hasło</label></td>
   <td><input type="password" name="password1" id="password1"
    value=""/></td>
  </tr><tr>
   <td><label for="password2">Powtórzenie hasła</label></td>
   <td><input type="password" name="password2" id="password2"
    value=""/></td>
  </tr><tr>
   <td><label for="email">Adres email</label></td>
   <td><input type="text" name="email" id="email"
    value="<?php if (isset($ POST['email']))
    echo htmlspecialchars($_POST['email']); ?>"/></td>
  </tr><tr>
   <td><label for="captcha">Weryfikacja</label></td>
   <td>Wpisz tekst widoczny na obrazku<br/ >
   <img src="img/captcha.php?nocache=<?php echo time(); ?>" alt=""/><br />
   <input type="text" name="captcha" id="captcha"/></td>
  </tr><tr>
   <td> </td>
   <td><input type="submit" value="Zarejestruj"/></td>
   <td><input type="hidden" name="submitted" value="1"/></td>
  </tr><tr>
 </table>
</form>
<?php
$form = ob_get_clean();

// wyświetlenie formularza, jeśli strona jest wyświetlana po raz pierwszy
if (!isset($_POST['submitted']))
{
    $GLOBALS['TEMPLATE']['content'] = $form;
}
```

```
// w przeciwnym razie przetworzenie danych wejściowych
else
{
    // weryfikacja hasła
    $password1 = (isset($ POST['password1'])) ? $ POST['password1'] : '';
    $password2 = (isset($ POST['password2'])) ? $ POST['password2'] : '';
    $password = ($password1 && $password1 == $password2) ?
        sha1($password1) : '';

    // weryfikacja tekstu CAPTCHA
    $captcha = (isset($_POST['captcha']) &&
        strtoupper($ POST['captcha']) == $ SESSION['captcha']);

    // jeśli wszystkie dane są prawidłowe — dodanie rekordu
    if ($password &&
        $captcha &&
        User::validateUsername($_POST['username']) &&
        User::validateEmailAddr($_POST['email']))
    {
        // sprawdzenie, czy użytkownik już istnieje
        $user = User::getByUsername($_POST['username']);
        if ($user->userId)
        {
            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Przepraszamy, ' .
                'takie konto już istnieje.</strong></p> <p>Prosimy podać ' .
                'inną nazwę użytkownika.</p>';
            $GLOBALS['TEMPLATE']['content'] .= $form;
        }
        else
        {
            // utworzenie nieaktywnego rekordu użytkownika
            $u = new User();
            $u->username = $ POST['username'];
            $u->password = $password;
            $u->emailAddr = $ POST['email'];
            $token = $u->setInactive();

            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Dziękujemy za ' .
                'zarejestrowanie się.</strong></p> <p>Należy pamiętać o ' .
                'zweryfikowaniu konta i kliknąć łącze <a href="verify.php?uid=' .
                $u->userId . '&token=' . $token . '">verify.php?uid=' .
                $u->userId . '&token=' . $token . '</a></p>';
         }
    }
    // dane nieprawidłowe
    else
    {
        $GLOBALS['TEMPLATE']['content'] .= '<p><strong>Podano nieprawidłowe ' .
            'dane.</strong></p> <p>Prosimy prawidłowo wypełnić ' .
            'wszystkie pola, abyśmy mogli zarejestrować konto użytkownika.</p>';
        $GLOBALS['TEMPLATE']['content'] .= $form;
    }
}

// wyświetlenie strony
include '../templates/template-page.php';
?>
```

Najpierw kod z pliku *register.php* importuje pliki z kodem współużytkowanym, który będzie później wykorzystywany. Niektórzy programiści wolą umieszczać wszystkie instrukcje `include` w jednym wspólnym pliku nagłówkowym, a następnie dołączać sam plik nagłówkowy — dzięki temu kod właściwy jest krótszy. W projekcie będziemy jednak oddzielnie dołączać pojedyncze pliki, ponieważ wydaje się, że takie rozwiązanie jest prostsze w utrzymaniu.

Inni programiści wykorzystują funkcję `chdir()` do zmiany katalogu roboczego PHP, dzięki czemu nie trzeba za każdym razem odwracać ścieżek w systemie w celu dołączenia pliku. Również tutaj decydują osobiste upodobania. Jednak w przypadku wyboru takiego rozwiązania trzeba zwrócić szczególną uwagę na starsze instalacje PHP, w których używany jest tryb bezpieczny. Wykonanie funkcji `chdir()` może bowiem się nie udać i nie zwróci żadnego komunikatu o błędzie, jeżeli wskazany katalog będzie niedostępny.

```
<?php
// dołączenie kodu współużytkowanego
chdir('../');
include 'lib/common.php';
include 'lib/db.php';
include 'lib/functions.php';
include 'lib/User.php';
…
?>
```

Po zaimportowaniu plików z kodem współużytkowanym wywoływana jest metoda `session_start()`. Wywołania HTTP są bezstanowe, co oznacza, że serwer WWW zwraca każdą stronę bez śledzenia, co działo się z nią wcześniej, i bez przewidywania, co może się z nią zdarzyć w przyszłości. Mechanizm śledzenia sesji dostępny w PHP umożliwia utrzymywanie w prosty sposób stanów między kolejnym wywołaniami oraz przenoszenie wartości z jednego wywołania do następnego. Wykorzystanie sesji jest niezbędne, aby utrzymać wartość CAPTCHA wygenerowaną w pliku *captcha.php*.

Gdy przygotowywane są większe bloki kodu HTML, na przykład kod formularza do rejestracji, warto jest je buforować dla zwiększenia czytelności kodu źródłowego. Niektórzy programiści preferują natomiast definiowanie zmiennej buforowej i cykliczne doklejanie do niej kolejnych fragmentów HTML, na przykład:

```
<?php
$GLOBALS['TEMPLATE']['content'] = '<form action="'.
    htmlspecialchars(currentFile()) . '" method="post">';
$GLOBALS['TEMPLATE']['content'] .= '<table>';
$GLOBALS['TEMPLATE']['content'] .= '<tr>';
$GLOBALS['TEMPLATE']['content'] .= '<td><label for="username">Nazwa
➥użytkownika</label>' . '</td>';
…
?>
```

Takie podejście wydaje się jednak dość niewygodne i stosunkowo czasochłonne. W przypadku buforowania danych wynikowych wystarczy rozpocząć buforowanie wywołaniem funkcji `ob_start()`, odczytać zawartość bufora funkcją `ob_get_contents()` i zatrzymać buforowanie funkcją `ob_end_clean()`. Funkcja `ob_get_clean()` łączy w sobie wywołanie dwóch funkcji: `ob_get_contents()` i `ob_end_clean()`. Ponadto dla interpretera łatwiej jest włączać i wyłączać tryb PHP, dlatego tak skonstruowany kod obsługi dużych bloków danych wyjściowych powinien działać szybciej niż w metodzie z dopisywaniem danych do bufora.

Gdy użytkownik po raz pierwszy wywołuje stronę, nie powinno być zdefiniowanych żadnych wartości `$_POST`, dlatego kod po prostu zwraca formularz rejestracji. Gdy użytkownik zatwierdzi formularz, ustawiana jest zmienna `$_POST['submitted']`, dzięki czemu wiadomo, że należy rozpocząć przetwarzanie danych wejściowych.

Kod, którego zadaniem jest weryfikacja poprawności nazwy użytkownika i hasła, należy do klasy `User`. Obydwa hasła wpisane w formularzu są ze sobą porównywane, a następnie hasło jest szyfrowane i już w postaci zaszyfrowanej zapisywane w bazie danych do późniejszego użycia. Na koniec wartość wpisana przez użytkownika na podstawie obrazka CAPTCHA jest porównywana z wartością wcześniej zapisaną w sesji przez kod z pliku *captcha.php*. Jeżeli wszystkie dane są poprawne, rekord zostaje zapisany w bazie danych.

Skrypt *verify.php* przywoływany w kodzie HTML odpowiada za odczytanie identyfikatora użytkownika i znacznika aktywacji, sprawdzenie odpowiednich danych w bazie i uaktywnienie konta użytkownika. Skrypt ten również musi zostać zapisany w katalogu dostępnym publicznie.

```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';
include '../lib/db.php';
include '../lib/functions.php';
include '../lib/User.php';

// sprawdzenie, czy otrzymano identyfikator użytkownika i znacznik
if (!isset($_GET['uid']) || !isset($_GET['token']))
{
    $GLOBALS['TEMPLATE']['content'] = '<p><strong>Otrzymane informacje ' .
        'są niepełne.</strong></p> <p>Prosimy spróbować ponownie.</p>';
    include '../templates/template-page.phptemplate_page.php';
    exit();
}

// weryfikacja identyfikatora użytkownika
if (!$user = User::getById($_GET['uid']))
{
    $GLOBALS['TEMPLATE']['content'] = '<p><strong>Podane konto nie
    ↪istnieje.</strong>' .
        '</p> <p>Prosimy spróbować ponownie.</p>';
}
// upewnienie się, że konto jest nieaktywne
else
{
    if ($user->isActive)
    {
        $GLOBALS['TEMPLATE']['content'] = '<p><strong>Konto ' .
            'zostało już zweryfikowane.</strong></p>';
    }
    // uaktywnienie konta
    else
    {
        if ($user->setActive($_GET['token']))
        {
            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Dziękujemy ' .
                'za zweryfikowanie konta.</strong></p> <p>Można się ' .
                'teraz <a href="login.php">zalogować</a>.</p>';
```

```
        }
        else
        {
            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Podano ' .
                'nieprawidłowe dane.</strong></p> <p>Prosimy spróbować
                ➥ponownie.</p>';
        }
    }
}

// wyświetlenie strony
include '../templates/template-page.php';
?>
```

Wysyłanie e-maila z łączem do weryfikacji

Aktualnie skrypt *register.php* wyświetla bezpośrednie łącze służące do weryfikacji konta, natomiast w środowisku produkcyjnym zwykle odpowiednie łącze wysyła się pocztą elektroniczną na adres wpisany przez użytkownika. Wychodzi się przy tym z założenia, że prawdziwy użytkownik poda prawidłowy adres pocztowy i samodzielnie potwierdzi założenie konta, czego nie robi znakomita większość spamerów.

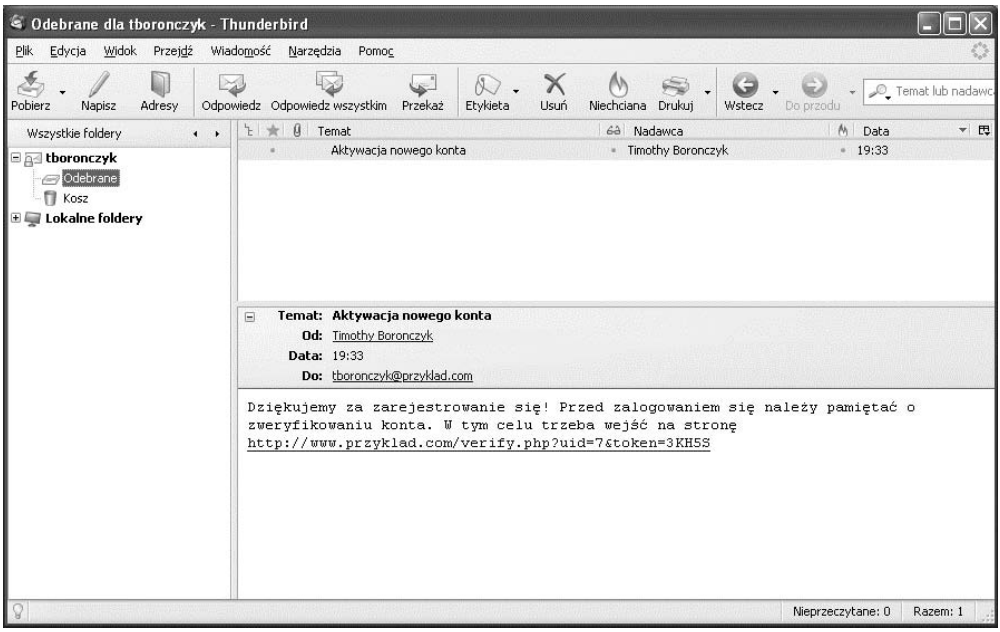
Funkcja `mail()` służy do wysyłania poczty elektronicznej przez PHP. Pierwszym argumentem funkcji jest adres pocztowy użytkownika, drugim jest temat wiadomości pocztowej, trzecim zaś — treść samej wiadomości. Zazwyczaj zaleca się, by nie wstrzymywać wyświetlania ostrzeżeń przy użyciu symbolu `@`, w tym przypadku jednak jest to konieczne, ponieważ w razie niepowodzenia funkcja `mail()` zwróci wartość `false` **oraz** wygeneruje komunikat z ostrzeżeniem.

Kod, który należy umieścić w pliku *register.php*, aby zamiast wyświetlać łącze do weryfikacji konta w przeglądarce, wysyłać je w wiadomości pocztowej, może mieć następującą postać:

```
<?php
…
// utworzenie nieaktywnego rekordu użytkownika
$u = new User();
$u->username = $_POST['username'];
$u->password = $password;
$u->emailAddr = $_POST['email'];
$token = $u->setInactive();

$message = 'Dziękujemy za zarejestrowanie się! Przed zalogowaniem się ' .
           'należy pamiętać o zweryfikowaniu konta. W tym celu trzeba wejść ' .
           'na stronę http://www.przyklad.com/verify.php?uid=' .
           $u->userId . '&token=' . $token . '.';

if (@mail($u->emailAddr, 'Aktywacja nowego konta', $message))
{
    $GLOBALS['TEMPLATE']['content'] = '<p><strong>Dziękujemy za ' .
                'zarejestrowanie się.</strong></p> <p>Wkrótce otrzymasz ' .
                'wiadomość pocztową z instrukcjami na temat sposobu ' .
                'aktywowania konta.</p>';
```

```
}
else
{
    $GLOBALS['TEMPLATE']['content'] = '<p><strong>Wystąpił błąd ' .
                'w trakcie wysyłania łącza aktywacyjnego.</strong></p> ' .
                '<p>Prosimy skontaktować się z administratorem ' .
                'pod adresem <a href="mailto:admin@przyklad.com">' .
                'admin@przyklad.com</a>, aby uzyskać pomoc.</p>';
}
…
?>
```

Na rysunku 1.3 przedstawiono wiadomość pocztową z potwierdzeniem, odczytaną w programie pocztowym.



Rysunek 1.3

Wysyłanie wiadomości pocztowej zawierającej zwykły tekst jest prostym zadaniem, natomiast wysyłanie wiadomości sformatowanej w języku HTML wymaga nieco więcej pracy. Obydwa rodzaje wiadomości mają niewątpliwe zalety: wiadomości tekstowe są bardziej czytelne, a prawdopodobieństwo ich zablokowania przez filtry antyspamowe jest stosunkowo niskie, natomiast wiadomości w języku HTML są bardziej przyjazne dla użytkowników, mniej sterylne, a poza tym mogą zawierać w treści hiperłącza, dzięki którym łatwiej jest skierować użytkownika na stronę odpowiadającą za weryfikację konta.

Wiadomość pocztowa w języku HTML może wyglądać następująco:

```
<html>
<p>Dziękujemy za zarejestrowanie się!</p>
<p>Przed zalogowaniem się należy pamiętać o zweryfikowaniu konta.
```

```
W tym celu trzeba wejść na stronę
<a href="http://www.przyklad.com/verify.php?uid=###&amp;token=xxxxx">
http://www.przyklad.com/verify.php?uid=###&amp;token=xxxxx</a>.</p>
<p>Jeżeli używany program pocztowy nie pozwala na kliknięcie hiperłączy
obecnych w tej wiadomości, należy skopiować hiperłącze i wkleić je w pasku
adresów przeglądarki, aby wyświetlić stronę do weryfikacji konta.</p>
</html>
```

Gdyby jednak powyższą wiadomość wysłać w sposób przedstawiony w poprzednim przykładzie, wówczas i tak dotarłaby ona do adresata w postaci zwykłego tekstu, pomimo że zawiera przecież znaczniki języka HTML. Aby wskazać klientowi pocztowemu prawidłowy sposób wyświetlania wiadomości, konieczne jest również przesłanie odpowiednich nagłówków `MIME` i `Content-Type`. Dodatkowe nagłówki stanowią opcjonalny czwarty parametr funkcji `mail()`.

```
<?php
// treść sformatowanej wiadomości przechowywana w zmiennej $html_message
// sformatowany e-mail wymaga podania nagłówków MIME i Content-Type
$headers = array('MIME-Version: 1.0',
                 'Content-Type: text/html; charset="iso-8859-2"');
// dodatkowe nagłówki są przekazywane jako czwarty argument funkcji mail()
mail($user- > emailAddr, 'Prosimy aktywować nowe konto', $html message,
     join("\n", $headers));
?>
```

Możliwe jest połączenie zalet obu rodzajów wiadomości pocztowych — wystarczy w tym celu wysłać wiadomość w formacie mieszanym. Wiadomość w formacie mieszanym zawiera tak naprawdę dwie wiadomości: tekstową i sformatowaną w języku HTML, a już do decyzji klienta pocztowego pozostaje, która część wiadomości zostanie wyświetlona. Poniżej przedstawiono wiadomość mieszaną:

```
--==A.BC 123 XYZ 678.9
Content-Type: text/plain; charset="iso-8859-2"

Dziękujemy za zarejestrowanie się!

Przed zalogowaniem się należy pamiętać o zweryfikowaniu konta.
W tym celu trzeba wejść na stronę
➥http://www.przyklad.com/verify.php?uid=###&amp;token=xxxxx.

--==A.BC 123 XYZ 678.9
Content-Type: text/plain; charset="iso-8859-2"

<html>
<p>Dziękujemy za zarejestrowanie się!</p>
<p>Przed zalogowaniem się należy pamiętać o zweryfikowaniu konta.
W tym celu trzeba wejść na stronę
<a href="http://www.przyklad.com/verify.php?uid=###&amp;token=xxxxx">
http://www.przyklad.com/verify.php?uid=###&amp;token=xxxxx</a>.</p>
<p>Jeżeli używany program pocztowy nie pozwala na kliknięcie hiperłączy
obecnych w tej wiadomości, należy skopiować hiperłącze i wkleić je w pasku
adresów przeglądarki, aby wyświetlić stronę do weryfikacji konta.</p>
</html>

--==A.BC_123_XYZ_678.9--
```

Aby wysłać tak skonstruowaną wiadomość, trzeba użyć następujących nagłówków:

```
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary="==A.BC_123_XYZ_678.9"
```

Warto zwrócić uwagę, że poszczególne segmenty wiadomości pocztowej są od siebie oddzielone specjalnym ciągiem znaków. Ciąg znaków w postaci `==A.BC_123_XYZ_678.9` nie ma tak naprawdę konkretnego znaczenia; wystarczy, by był to losowy ciąg znaków, który nie pojawi się nigdzie w treści któregoś z elementów wiadomości. Ciąg znaków oddzielający od siebie poszczególne bloki wiadomości jest zawsze poprzedzany dwoma myślnikami, a przed nim występuje pusty wiersz. Myślniki na końcu tego ciągu wskazują jednocześnie koniec całej wiadomości.

Logowanie i wylogowywanie

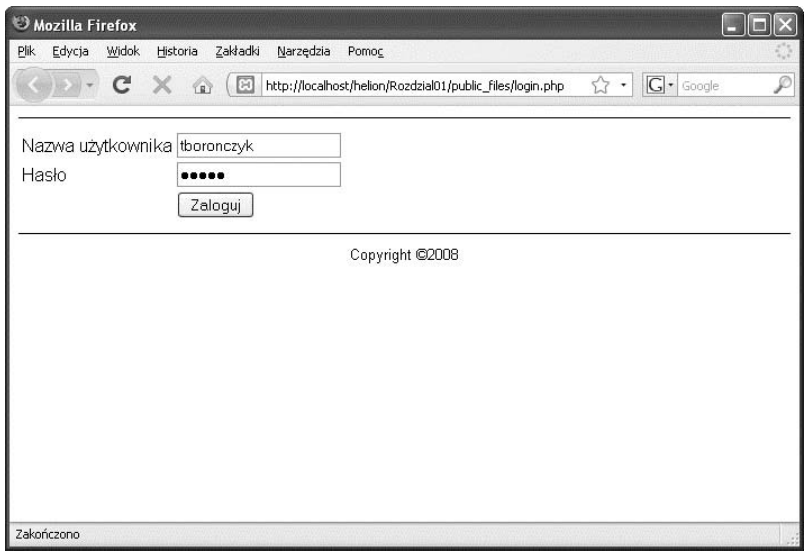
Można już tworzyć nowe konta użytkowników i weryfikować je jako konta założone przez prawdziwych użytkowników dzięki wykorzystaniu podanego adresu poczty elektronicznej. Kolejnym krokiem jest zatem opracowanie mechanizmu, który będzie dostępny dla użytkowników i pozwoli im na logowanie się i wylogowywanie z systemu. Większość uciążliwych zadań związanych ze śledzeniem sesji będzie wykonywana przez PHP, dlatego nam pozostaje tylko zapisywanie danych identyfikacyjnych w zmiennej `$_SESSION`. Poniższy kod należy zapisać w pliku *login.php*.

```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';
include '../lib/db.php';
include '../lib/functions.php';
include '../lib/User.php';

// rozpoczęcie lub dołączenie do sesji
session_start();
header('Cache-control: private');

// logowanie, jeśli ustawiono zmienną login
if (isset($_GET['login']))
{
    if (isset($_POST['username']) && isset($_POST['password']))
    {
        // odczytanie rekordu użytkownika
        $user = (User::validateUsername($_POST['username'])) ?
            User::getByUsername($_POST['username']) : new User();

        if ($user->userId && $user->password == sha1($_POST['password']))
        {
            // zapisanie wartości w sesji, aby móc śledzić użytkownika
            // i przekierować go do strony głównej
            $_SESSION['access'] = TRUE;
            $_SESSION['userId'] = $user->userId;
            $_SESSION['username'] = $user->username;
            header('Location: main.php');
        }
```

```
        else
        {
            // nieprawidłowy użytkownik i (lub) hasło
            $_SESSION['access'] = FALSE;
            $_SESSION['username'] = null;
            header('Location: 401.php');
        }
    }
    // brak danych uwierzytelniających
    else
    {
        $_SESSION['access'] = FALSE;
        $_SESSION['username'] = null;
        header('Location: 401.php');
    }
    exit();
}

// wylogowanie, jeśli ustawiono zmienną logout
// (wyczyszczenie danych sesji prowadzi do wylogowania użytkownika)
else if (isset($ GET['logout']))
{
    if (isset($_COOKIE[session_name()]))
    {
        setcookie(session_name(), '', time() - 42000, '/');
    }

    $_SESSION = array();
    session_unset();
    session destroy();
}

// wygenerowanie formularza logowania
ob_start();
?>
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>?login"
 method="post">
 <table>
  <tr>
   <td><label for="username">Nazwa użytkownika</label></td>
   <td><input type="text" name="username" id="username"/></td>
  </tr><tr>
   <td><label for="password">Hasło</label></td>
   <td><input type="password" name="password" id="password"/></td>
  </tr><tr>
   <td> </td>
   <td><input type="submit" value="Zaloguj"/></td>
  </tr>
 </table>
</form>
<?php
$GLOBALS['TEMPLATE']['content'] = ob_get_clean();

// wyświetlenie strony
include '../templates/template-page.php';
?>
```

Przedstawiony kod źródłowy odpowiada zarówno za logowanie się, jak i wylogowywanie z witryny. W tym celu w adresie strony przekazywany jest odpowiedni parametr. Wysłanie zawartości formularza do strony *login.php?login* spowoduje zalogowanie użytkownika, natomiast wywołanie strony *login.php?logout* doprowadzi do wyczyszczenia wszystkich danych sesji, co będzie jednoznaczne z wylogowaniem bieżącego użytkownika. Formularz logowania przedstawiono na rysunku 1.4.

Rysunek 1.4



Aby zalogować użytkownika, skrypt przyjmuje nazwę użytkownika i hasło. Nazwa użytkownika, z którym wywołano skrypt, jest przekazywana do `metody getByUsername()`, aby na tej podstawie odczytać rekord z bazy danych. Z kolei hasło jest szyfrowane, aby jego zaszyfrowaną wersję można było porównać z zaszyfrowanym hasłem przechowywanym w bazie. Jeżeli dane uwierzytelniające będą identyczne, będzie to oznaczać, że użytkownik wpisał prawidłową nazwę i hasło i zostanie zalogowany, to znaczy informacje na jego temat zostaną zapisane w sesji i nastąpi przekierowanie do strony głównej. Jeżeli natomiast uwierzytelnianie nie powiedzie się, dane sesji zostaną wyczyszczone i użytkownik zostanie przekierowany do strony z informacją o błędzie (*404.php*).

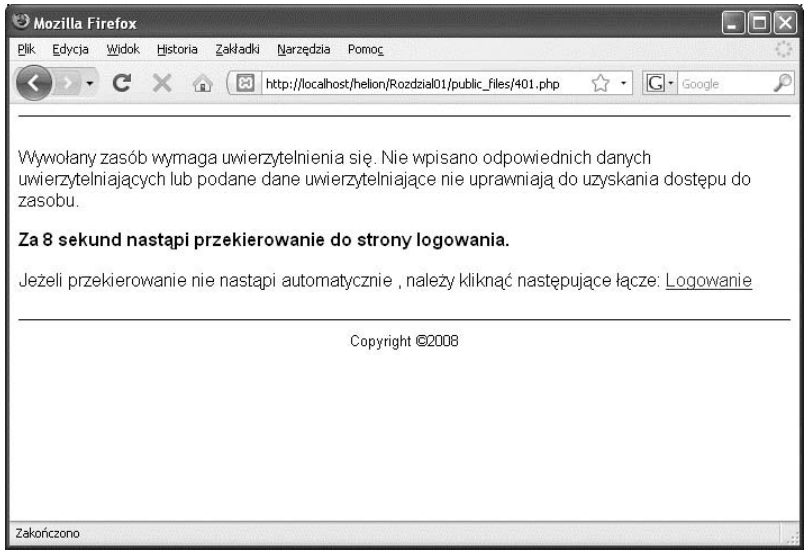
Jeżeli skrypt zostanie wywołany bez żadnych parametrów, jego danymi wynikowymi będzie kod HTML formularza logowania. Jest to wygodne rozwiązanie w sytuacji, gdy formularz ma być wywoływany przez inną stronę albo gdy trzeba przekierować do niego ze strony z komunikatem o błędzie. Formularz ten nie jest jednak jedynym narzędziem umożliwiającym zalogowanie. Jako że instrukcja `exit` została celowo umieszczona po kodzie realizującym logowanie, skrypt może posłużyć do przetwarzania dowolnego formularza logowania bez względu na to, czy znajduje się on na stronie zgodnej z szablonem, czy gdziekolwiek indziej. Wystarczy tylko pamiętać, aby w adresie skryptu przekazać parametr logowania.

Jeżeli logowanie zakończy się niepowodzeniem, użytkownik zostanie przekierowany do strony *401.php*.

```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';

// rozpoczęcie lub dołączenie do sesji
session start();
header('Cache-control: private');

// zwrócenie błędu 401, jeśli użytkownik się nie uwierzytelnił
if (!isset($_SESSION['access']) || $_SESSION['access'] != TRUE)
{
    header('HTTP/1.0 401 Authorization Error');
    ob start();
?>
<script type="text/javascript">
window.seconds = 10;
window.onload = function()
{
    if (window.seconds != 0)
    {
        document.getElementById('secondsDisplay').innerHTML = '' +
            window.seconds + ' sekund' + ((window.seconds > 4) ? '' : 'y');
        window.seconds--;
        setTimeout(window.onload, 1000);
    }
    else
    {
        window.location = 'login.php';
    }
}
</script>
<?php
    $GLOBALS['TEMPLATE']['extra_head'] = ob_get_contents();
    ob_clean();

?>
<p>Wywołany zasób wymaga uwierzytelnienia się. Nie wpisano
odpowiednich danych uwierzytelniających lub podane dane
uwierzytelniające nie uprawniają do uzyskania dostępu do zasobu.</p>

<p><strong>Za <span id="secondsDisplay">10 sekund</span> nastąpi
przekierowanie do strony logowania.</strong></p>

<p>Jeżeli przekierowanie nie nastąpi automatycznie, należy kliknąć następujące
↪łącze:
<a href="login.php">Logowanie</a></p>
<?php
    $GLOBALS['TEMPLATE']['content'] = ob_get_clean();

    include '../templates/template-page.php';
    exit();
}
?>
```

W przypadku wystąpienia błędu 401 w przeglądarce pojawi się strona widoczna na rysunku 1.5. Najważniejszym zadaniem skryptu jest wysłanie do przeglądarki informacji o błędzie autoryzacji i przekierowanie użytkownika z powrotem do formularza logowania (kodem oznacza-

Rysunek 1.5



jącym błąd autoryzacji HTTP jest 401). Dzięki temu, że wywoływana jest metoda `session_start()` i sprawdzana jest wartość `$_SESSION['access']`, błąd autoryzacji pojawia się jedynie w przypadku, gdy użytkownik nie został wcześniej uwierzytelniony. Aby objąć takim zabezpieczeniem dowolną stronę, należy plik z przedstawionym kodem umieścić na początku strony. Jeżeli użytkownik wcześniej się uwierzytelnił, bez trudu uzyska dostęp do wywoływanego zasobu.

Przekierowanie użytkownika na poziomie klienta można wykonać na kilka sposobów. W naszym projekcie połączono niewielki fragment kodu JavaScript z danymi wynikowymi skryptu, aby odliczyć 10 sekund (10 000 milisekund). Ten czas powinien wystarczyć, aby użytkownik zauważył, że odmówiono mu dostępu. Ten sam kod na bieżąco wyświetla czas pozostały do przekierowania użytkownika, a po jego upływie dokonuje przekierowania przez odpowiednie zdefiniowanie wartości właściwości `window.location`. Innym sposobem przekierowania klienta jest zwrócenie elementu meta języka HTML:

```
<meta http-equiv="refresh"
  content="10;URL=http://www.przyklad.com/login.php" />
```

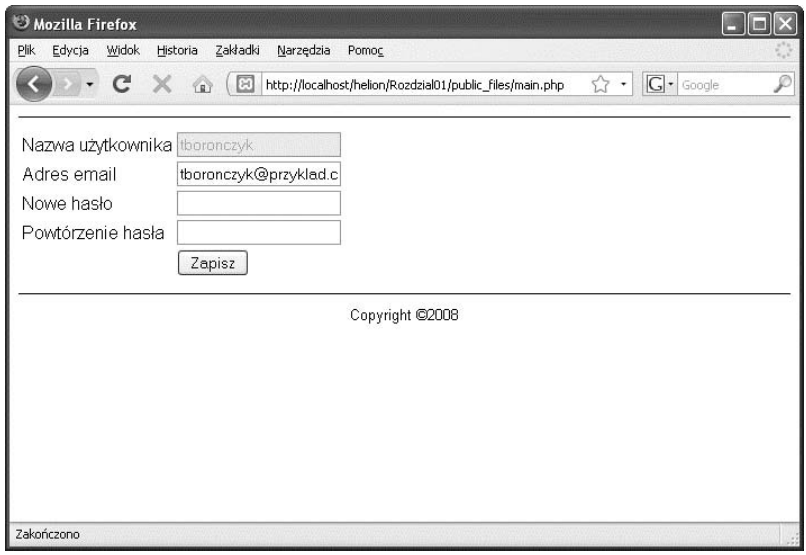
Bez względu na to, jaką metodę wybierze się do wykonania przekierowania, zawsze należy udostępniać łącze, na wypadek gdyby przeglądarka nie potrafiła prawidłowo przekierować użytkownika.

Zmiana danych

Użytkownicy mogą zechcieć zmienić imiona, nazwiska i adresy poczty elektronicznej, dlatego warto taką funkcję zawrzeć w aplikacji. Już wcześniej, przy okazji opisywania klasy `User`, pokazano sposób zmiany rekordu dotyczącego użytkownika. Zmiana danych będzie przebiegać w taki sam sposób: najpierw właściwościom obiektu nadane zostaną nowe wartości, a następnie nastąpi wywołanie metody `save()`.

Odpowiedni kod został umieszczony w pliku *main.php* z tego prostego powodu, że po zalogowaniu użytkownika skrypt *login.php* przekierowuje go właśnie do strony *main.php*. W innych aplikacjach można ten sam skrypt nazwać inaczej, na przykład *editmember.php*, a na stronie *main.php* prezentować inne, ciekawe informacje. Formularz przedstawiono na rysunku 1.6.

Rysunek 1.6




```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';
include '../lib/db.php';
include '../lib/functions.php';
include '../lib/User.php';

// dołączenie pliku 401.php — użytkownik może oglądać stronę tylko po zalogowaniu
include '401.php';

// wygenerowanie formularza informacji o użytkowniku
$user = User::getById($ SESSION['userId']);

ob_start();
?>
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>"
 method="post">
 <table>
  <tr>
   <td><label>Nazwa użytkownika</label></td>
   <td><input type="text" name="username"  disabled="disabled"
    readonly="readonly" value="<?php echo $user->username; ?>"/></td>
  </tr><tr>
   <td><label for="email">Adres email</label></td>
   <td><input type="text" name="email" id="email"
    value="<?php echo (isset($ POST['email']))? htmlspecialchars(
$_POST['email']) : $user->emailAddr; ?>"/></td>
  </tr><tr>
   <td><label for="password">Nowe hasło</label></td>
   <td><input type="password" name="password1" id="password1"/></td>
```

```
  </tr><tr>
   <td><label for="password2">Powtórzenie hasła</label></td>
   <td><input type="password" name="password2" id="password2"/></td>
  </tr><tr>
  <td> </td>
   <td><input type="submit" value="Zapisz"/></td>
   <td><input type="hidden" name="submitted" value="1"/></td>
  </tr><tr>
 </table>
</form>
<?php
$form = ob_get_clean();

// wyświetlenie formularza, jeśli strona jest wyświetlana po raz pierwszy
if (!isset($_POST['submitted']))
{
    $GLOBALS['TEMPLATE']['content'] = $form;
}
// w przeciwnym razie przetworzenie danych wejściowych
else
{
    // sprawdzenie poprawności hasła
    $password1 = (isset($ POST['password1']) && $ POST['password1']) ?
        sha1($ POST['password1']) : $user->password;
    $password2 = (isset($ POST['password2']) && $ POST['password2']) ?
        sha1($ POST['password2']) : $user->password;
    $password = ($password1 == $password2) ? $password1 : '';

    // uaktualnienie rekordu, jeżeli dane wejściowe są poprawne
    if (User::validateEmailAddr($ POST['email']) && $password)
    {
        $user->emailAddr = $ POST['email'];
        $user->password = $password;
        $user->save();

        $GLOBALS['TEMPLATE']['content'] = '<p><strong>Informacje ' .
            'w bazie danych zostały uaktualnione.</strong></p>';
    }
    // dane nieprawidłowe
    else
    {
        $GLOBALS['TEMPLATE']['content'] .= '<p><strong>Podano nieprawidłowe ' .
            'dane.</strong></p>';
        $GLOBALS['TEMPLATE']['content'] .= $form;
    }
}

// wyświetlenie strony
include '../templates/template-page.php';
?>
```

Przedstawiony kod można zmienić na przykład tak, aby przed zapisaniem jakichkolwiek zmian w danych użytkownika weryfikować jego hasło. Powszechną praktyką jest również oznaczanie konta jako nieaktywnego i ponowne weryfikowanie adresu pocztowego, jeżeli uległ on zmianie.

Zapomniane hasła

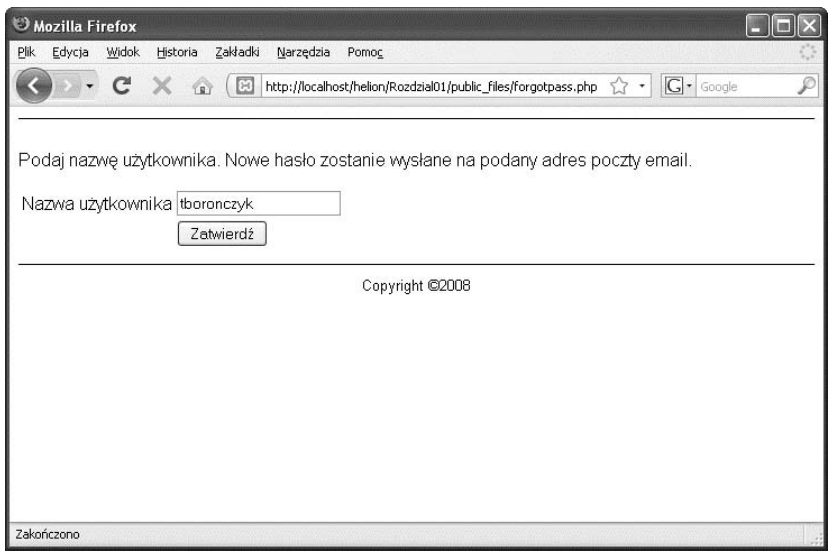
Czasami zdarza się, że użytkownicy zapominają hasło i nie mogą się zalogować. Ponieważ hasła w oryginalnej formie nigdzie nie są przechowywane, nie można ich w jakikolwiek sposób odzyskać. Zamiast tego trzeba wygenerować nowe hasło i przesłać je do użytkownika na podany przez niego adres poczty elektronicznej. Kod wykonujący taką czynność należy umieścić w pliku *forgotpass.php*.

```
<?php
// dołączenie kodu współużytkowanego
include '../lib/common.php';
include '../lib/db.php';
include '../lib/functions.php';
include '../lib/User.php';

// formularz HTML z żądaniem hasła
ob_start();
?>
<form action="<?php echo htmlspecialchars($_SEVER['PHP_SELF']); ?>"
 method="post">
<p>Podaj nazwę użytkownika. Nowe hasło zostanie wysłane
na podany adres poczty email.</p>
<table>
<tr>
 <td><label for="username">Nazwa użytkownika</label></td>
 <td><input type="text" name="username" id="username"
  value="<?php if (isset($_POST['username']))
  echo htmlspecialchars($_POST['username']); ?>"/></td>
</tr><tr>
 <td> </td>
 <td><input type="submit" value="Zatwierdź"/></td>
 <td><input type="hidden" name="submitted" value="1"/></td>
</tr><tr>
</table>
</form>
<?php
$form = ob_get_clean();

// wyświetlenie formularza, jeśli strona jest przeglądana po raz pierwszy
if (!isset($ POST['submitted']))
{
    $GLOBALS['TEMPLATE']['content'] = $form;
}
// w przeciwnym razie — przetworzenie danych wejściowych
else
{
    // sprawdzenie poprawności nazwy użytkownika
    if (User::validateUsername($_POST['username']))
    {
        $user = User::getByUsername($_POST['username']);
        if (!$user->userId)
        {
            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Przepraszamy, ' .
                'podane konto nie istnieje.</strong></p> <p>Prosimy podać ' .
                'inną nazwę użytkownika.</p>';
```

```
            $GLOBALS['TEMPLATE']['content'] .= $form;
        }
        else
        {
            // wygenerowanie nowego hasła
            $password = random_text(8);

            // wysłanie nowego hasła na adres pocztowy
            $message = 'Nowe hasło to: ' . $password;
            mail($user->emailAddr, 'New password', $message);

            $GLOBALS['TEMPLATE']['content'] = '<p><strong>Nowe hasło ' .
                'wysłano na podany adres pocztowy.</strong></p>';

            // zapisanie nowego hasła
            $user->password = $password;
            $user->save();
        }
    }
    // dane błędne
    else
    {
        $GLOBALS['TEMPLATE']['content'] .= '<p><strong>Nie podano ' .
            'prawidłowej nazwy użytkownika.</strong></p> <p>Prosimy ' .
            'spróbować ponownie.</p>';
        $GLOBALS['TEMPLATE']['content'] .= $form;
    }
}

// wyświetlenie strony
include '../templates/template-page.php';
?>
```

Na rysunku 1.7 przedstawiono powyższą stronę tak, jak będzie się ona prezentować w przeglądarce.

Rysunek 1.7



Podsumowanie

I gotowe! Utworzyliśmy podstawowy system rejestracji użytkowników, który można rozszerzać na dowolne sposoby. Można na przykład poszerzyć zakres danych na temat użytkowników o numery telefonów komórkowych (aby móc wysyłać do nich SMS-y), adresy pocztowe, a nawet ich nazwy użytkowników w komunikatorach internetowych.

W tym rozdziale zaprezentowano także, jak należy odpowiednio zaprojektować strukturę katalogów aplikacji oraz przedstawiono niektóre zalety wynikające z implementowania kodu gotowego do wielokrotnego wykorzystania. Opisaną strukturę katalogów oraz większość plików pomocniczych można wykorzystać również dla celów innych projektów prezentowanych w niniejszej książce.

W następnym rozdziale wykorzystamy efekty naszej dotychczasowej pracy i na ich podstawie utworzymy przykładowy społecznościowy biuletyn informacyjny, częściej określany mianem **forum**.