

Tomasz "Zyx" Jędrzejewski

Własny mechanizm sesji w PHP

Wersja 1.0 (3.06.2006)



Szczegółowe informacje o licencji znajdują się pod artykułem.

www.zyxist.com

Własny mechanizm sesji w PHP

Choć PHP oferuje swym programistom implementację sesji, nie jest ona doskonała. Aby doprowadzić ją do stanu, w którym można ją bezpiecznie użyć w systemach autoryzacji, trzeba poświęcić trochę czasu. Zamiast tego możemy pokusić się o napisanie systemu sesji całkowicie od zera, wykorzystując programowanie obiektowe do pomocy. Na taki wariant decydują się twórcy wielu aplikacji PHP, lecz początkujący programiści nie zawsze wiedzą, jak się do tego zabrać. Mam nadzieję, iż ten artykuł wyjaśni wszystkie kwestie.

Czym jest sesja?

Protokół HTTP jest w swej naturze bezstanowy, tj. serwery rozpatrują żądania zupełnie niezależnie, nie szukając między nimi powiązań w rodzaju wysłania ich przez tego samego użytkownika. Nastręcza to pewnych trudności przy tworzeniu systemów autoryzacji, w których PHP musi mieć rozeznanie w tym, kto się aktualnie z jakiego komputera zalogował i śledzić jego ruch w serwisie. Pojedynczą wizytę internauty na naszej witrynie, podczas której przejrzy on kilka podstron, nazywać będziemy sesją. Dla serwera WWW żądania te będą niezależne. Dla nas jednak występuje między nimi jeden wspólny element: zostały wysłane przez tę samą osobę, a my musimy znaleźć sposób, aby PHP to zauważył. W tym właśnie celu napiszemy system sesji oraz powiązany z nim prosty mechanizm autoryzacji i logowania użytkowników.

Sprawy techniczne

Do stworzenia systemu sesji postanowiłem wykorzystać PHP 5.1 oraz bazę danych MySQL. Komunikacja z nią odbywać się będzie poprzez bibliotekę *PHP Data Objects* dostępną standardowo w PHP. Ponadto skorzystamy z dobrodziejstw programowania obiektowego. Znajomość tych elementów jest zatem wymagana do zrozumienia artykułu i zakładam, że czytelnik posiada stosowne podstawy. Jeżeli nie, polecam zapoznanie się uprzednio z odpowiednimi artykułami w serwisie WebCity.pl.

Każda sesja będzie identyfikowana przez unikalny, 40-znakowy ciąg znaków zwany dalej ID. Ponadto notować będziemy adres IP komputera, z którego została ona utworzona oraz przeglądarkę, za pomocą której tego dokonano. Bez tych dodatkowych zabezpieczeń zdobycie ID sesji umożliwiłoby podszycie się pod innego użytkownika naszej witryny. Do sesji trafi także czas ostatniego żądania przetworzonego z jej użyciem (czyli czas ostatniej aktywności) oraz ID zalogowanego użytkownika lub 0, jeśli to osoba anonimowa. Wszystko to trzymane będzie w bazie, a między stronami wędrować będzie ID sesji w postaci ciastka. To właśnie w ten sposób powiązemy wszystkie żądania wysłane przez konkretnego użytkownika.

Sesja będzie tworzona przy pierwszym wejściu na stronę. W kolejnych aplikacja będzie łądownać wszystkie informacje z bazy, korzystając z ID sesji. Na sesji można wykonać jedną operację, tj. zmianę użytkownika do niej przypisanego - w ten sposób rozwiążemy sprawę logowania i wylogowywania. Jeśli sesja będzie nieaktywna przez dłużej, niż godzinę, skrypt będzie usuwać ją z bazy.

Baza danych

Na początek utwórzmy bazę danych. Jej struktura jest następująca:

```
CREATE TABLE `sessions` (  
  `session_id` varchar(40) NOT NULL,  
  `session_user` int(8) NOT NULL default '0',  
  `session_ip` varchar(15) NOT NULL default '',  
  `session_browser` varchar(128) NOT NULL default '',  
  `session_time` int(11) NOT NULL default '0',  
  PRIMARY KEY (`session_id`)  
) TYPE=MEMORY;  
  
CREATE TABLE `users` (  
  `user_id` int(10) NOT NULL AUTO_INCREMENT,  
  `user_login` varchar(32) NOT NULL,  
  `user_password` varchar(40) NOT NULL,
```

```

`user_lastvisit` int(8) NOT NULL,
PRIMARY KEY (`user_id`)
) TYPE=MyISAM AUTO_INCREMENT=2;

INSERT INTO `users` VALUES (1, 'luser',
'c3ae457bb31ea0b0df811cf615e81cb46fefdbe9', 1138562170);

```

Baza zawiera stworzony już profil jednego użytkownika o loginie *luser* i hasle *pppp*, które w bazie zaszyfrowane jest jednokierunkowym algorytmem SHA1. Zauważ, że dla tabeli *sessions* wybraliśmy typ "MEMORY". Ogranicza to użycie niektórych typów danych, ale nasze dane będą rezydować w pamięci RAM, nie na dysku, co wielokrotnie przyspieszy wykonywane tam operacje. Dodatkowy koszt poniesiemy, jeżeli serwer zostanie zrestartowany (wszystkie dane w tabeli zostaną wymazane), ale akurat w przypadku sesji nie jest to wielka strata. Najwyżej się ludzie będą musieli ponownie zalogować.

Klasy pomocnicze

Zanim przystąpimy do tworzenia samego systemu sesji, utwórzmy sobie dwie klasy pomocnicze. Pierwszą z nich będzie *HttpRequest* gromadząca dane o żądaniu HTTP. Dla potrzeb artykułu skróciłem ją do niezbędnego minimum: adresu IP oraz danych przeglądarki w surowej postaci. Jeżeli potrzebujesz więcej, dodaj sobie odpowiednie pola sam. Pola klasy są hermetyzowane, tj. nie ma do nich bezpośredniego dostępu z kodu aplikacji. Wszystko musi odbywać się poprzez metody pomocnicze, co gwarantuje, że żaden algorytm nie zmieni omyłkowo np. adresu IP w trakcie działania aplikacji. Kod umieść w pliku *request.php*:

```

<?php
class HttpRequest
{
    private $ip;
    private $browser;

    public function __construct()
    {
        $this -> ip = $_SERVER['REMOTE_ADDR'];
        $this -> browser = $_SERVER['HTTP_USER_AGENT'];
    } // end __construct();

    public function getIp()
    {
        return $this -> ip;
    } // end getIp();

    public function getBrowser()
    {
        return $this -> browser;
    } // end getBrowser();
}
?>

```

Adres IP pobieramy z pola `$_SERVER['REMOTE_ADDR']` i jest to adres ostatniego komputera, z którego nadeszło żądanie. Jeżeli zatem użytkownik znajduje się za serwerem proxy, będzie tu podany adres tego serwera. Niestety nic nie da się w tej materii zrobić. Teoretycznie mamy `$_SERVER['X_FORWARDED_FOR']`, ale proxy wpisuje tam sobie, co chce i wcale nie mamy gwarancji, że dane w tym polu nie zostały spreparowane przez jakiegoś hackera. Zaufanie temu adresowi było przyczyną kilku błędów bezpieczeństwa w aplikacji for dyskusyjnych phpBB. Z pierwszym adresem nie ma tego typu kłopotów - jest to adres, na który serwer musi wysłać odpowiedź zwrotną, więc siłą rzeczy prawdziwy być musi.

Druga z klas reprezentuje pojedynczego użytkownika. Zapiszemy ją w pliku *user.php*. Dane są tu także hermetyzowane.

```

<?php

class user
{

```

```

private $id;
private $login;
private $password;
private $lastvisit;
private $construct;

public function __construct($anonymous = true)
{
    if($anonymous == true)
    {
        $this -> id = 0;
        $this -> login = '';
        $this -> password = '';
        $this -> lastvisit = time();
    }
    $this -> construct = true;
} // end __construct();

public function isAnonymous()
{
    return ($this -> id == 0 ? true : false);
} // end isAnonymous();

public function getId()
{
    return $this -> id;
} // end getId();

public function getLogin()
{
    return $this -> login;
} // end getLogin();

public function getPassword()
{
    return $this -> password;
} // end getPassword();

public function getLastvisit()
{
    return date('d.m.Y, H:i', $this -> lastvisit);
} // end getLastvisit();

public function __set($name, $value)
{
    if(!$this -> construct)
    {
        $this -> $name = $value;
    }
} // end __set();

static public function checkPassword($login, $password)
{
    global $pdo;
    $stmt = $pdo -> prepare('SELECT user_id AS `id`,
        user_login AS `login`, user_password AS `password`,
        user_lastvisit AS `lastvisit` FROM users WHERE
        user_login = :login AND user_password = :password');
    $stmt -> bindValue(':login', $login, PDO::PARAM_STR);
    $stmt -> bindValue(':password', sha1($password),
        PDO::PARAM_STR);
}

```

```

$stmt -> execute();
$stmt -> setFetchMode(PDO::FETCH_CLASS, 'user',
    array(0 => false));
if($user = $stmt -> fetch())
{
    // Jezeli uzytkownik o takim loginie i hasle
    // istnieje, zwroc jego rekord w postaci obiektu
    $stmt -> closeCursor();
    return $user;
}
else
{
    $stmt -> closeCursor();
    // Bledy w loginie/hasle zgłaszamy zerem
    return 0;
}
} // end checkPassword();
}
?>

```

Konstruktor daje tu możliwość utworzenia jedynie anonimowego użytkownika. Wszystkie pola są prywatne, jak więc zatem załadować tu dane jakiegokolwiek zarejestrowanego? Otóż zadanie to jest zrzucone na bibliotekę PDO i właśnie pod jej kątem pisana jest klasa. Przykład wczytania tych danych jest widoczny w statycznej metodzie *checkPassword()*, której użyjemy do sprawdzenia, czy logujący się podał prawidłowy login i hasło. Zwraca ona obiekt użytkownika, którego dane podaliśmy, lub 0 w przypadku niepowodzenia. Procedura tworzenia obiektu przebiega w ten sposób:

1. Przygotowujemy zapytanie i faszerujemy je danymi z formularza. Hasło szyfrujemy algorytmem SHA1.
2. Wykonujemy zapytanie.
3. Metodą *setFetchMode()* deklarujemy, że chcemy dane rekordu ładować do obiektu konkretnej klasy.
4. Korzystamy z faktu, że PDO najpierw ładuje dane do nowego obiektu, a potem dopiero wywołuje konstruktor, czego tradycyjnie nie da się w PHP wykonać. Tu do akcji wkracza metoda magiczna *__set()* przechwytyjąca wszystkie zapisy do pól obiektu. Jeżeli konstruktor nie został jeszcze wywołany, pole pomocnicze *construct* nie będzie miało wartości i metoda ta będzie wprowadzać dane do odpowiednich pól. Po wywołaniu konstruktora zostanie ona zablokowana, hermetyzując całą klasę. W ten sposób mamy i prywatne pola, i możliwość zapisu do nich z poziomu PDO. To taka ciekawa sztuczka odnośnie tej biblioteki.

Klasa sesji

Przystąpimy teraz do napisania klasy sesji (plik *session.php*). Na początek zadeklarujemy dwie stałe identyfikujące czas trwania każdej sesji oraz nazwę ciastka do przesyłania ID. Następnie rozpoczniemy budowę samej klasy:

```

<?php

define('COOKIE_NAME', 'tuibgidf'); // losowe
define('COOKIE_EXPIRE', 3600); // 1 godzina

class session
{
    private $id;
    private $ip;
    private $browser;
    private $time;
    private $user;
}

```

Pierwsze cztery pola identyfikują poszczególne informacje, jakie można znaleźć w sesji. Ostatnie, *\$user*, nie będzie przechowywać ID użytkownika, lecz obiekt *user* tak, aby mieć dostęp także do innych jego danych

(np. loginu). W kodzie znajdzie się pięć metod:

1. `__construct()` - konstruktor klasy. Jego zadaniem jest załadowanie istniejącej sesji lub stworzenie nowej, anonimowej w przypadku błędu.
2. `create()` - tworzy nową, anonimową sesję.
3. `update(user $user)` - zmienia użytkownika przypisanego do danej sesji na podstawie dostarczonego obiektu klasy `user`. Użyjemy jej do logowania i wylogowywania.
4. `garbageCollector()` - usuwa stare sesje z bazy.
5. `getUser()` - zwraca obiekt użytkownika przypisany do sesji.

Na pierwszy ogień pójdzie oczywiście konstruktor.

```
public function __construct()
{
    global $pdo, $request;

    // Kontrola poprawnosci ciastka
    if(!isset($_COOKIE[COOKIE_NAME]))
    {
        $_COOKIE[COOKIE_NAME] = '';
    }
    if(strlen($_COOKIE[COOKIE_NAME]) != 40)
    {
        $this -> create();
    }
}
```

W tym fragmencie kontrolujemy poprawność danych zawartych w ciastku. Musi ono istnieć i mieć długość dokładnie 40 znaków, inaczej będziemy zmuszeni stworzyć nową, anonimową sesję.

```
// Wyslanie zapytania o sesje. Od razu sprawdzamy
// jej waznosc oraz zgodnosc IP i przegladarki
$stmt = $pdo -> prepare('SELECT
    session_user, session_ip,
    session_browser, session_time FROM sessions
WHERE session_id = :sid AND
    session_ip = :sip AND
    session_browser = :sbrowser AND
    session_time > :time
');
$stmt -> bindValue(':sid', $_COOKIE[COOKIE_NAME],
    PDO::PARAM_STR);
$stmt -> bindValue(':sip', $request -> getIp(),
    PDO::PARAM_STR);
$stmt -> bindValue(':sbrowser', $request -> getBrowser(),
    PDO::PARAM_STR);
$stmt -> bindValue(':time', time() - COOKIE_EXPIRE,
    PDO::PARAM_INT);
$stmt -> execute();
```

Tutaj dokonujemy pobrania rekordu sesji. Zauważ, że już na tym etapie sprawdzane jest, czy zgadzają się adres IP, przeglądarka oraz czas ważności. Jeżeli któryś z tych elementów będzie niepoprawny, biblioteka nawet nie otrzyma rekordu. Poszczególne elementy podpinamy do zapytania metodami `bindValue()`, unikając w ten sposób jakichkolwiek problemów z ustawieniami magic quotes.

```
if($session = $stmt -> fetch(PDO::FETCH_ASSOC))
{
    $stmt -> closeCursor();
    $this -> id = $_COOKIE[COOKIE_NAME];
    $this -> ip = $session['session_ip'];
    $this -> browser = $session['session_browser'];
    $this -> time = $session['session_time'];
    setcookie(COOKIE_NAME, $this -> id,
```

```

        time() + COOKIE_EXPIRE);
    $stmt = $pdo -> prepare('UPDATE sessions SET
        session_time = :time WHERE session_id = :sid');
    $stmt -> bindValue(':sid', $_COOKIE[COOKIE_NAME],
        PDO::PARAM_STR);
    $stmt -> bindValue(':time', time(), PDO::PARAM_INT);
    $stmt -> execute();

```

Jeżeli udało się wczytać rekord, zamykamy kursor (przypominam, że nie wolno o tej czynności zapominać podczas pracy z biblioteką PDO!) i przypisujemy dane rekordu do naszego obiektu. Czas aktualności sesji jest odświeżany zarówno w ciastku, jak i w bazie danych poprzez zapytanie UPDATE.

```

        if($session['session_user'] == 0)
        {
            // sesja anonimowa
            $this -> user = new user(true);
        }
        else
        {

```

Tutaj obsługujemy sytuację, gdy sesja należy do niezalogowanego użytkownika. Ręcznie tworzymy wtedy obiekt klasy user z parametrem TRUE (anonimowość).

```

        // sesja zalogowanego
        $stmt = $pdo -> prepare('SELECT user_id AS `id`,
            user_login AS `login`,
            user_password AS `password`,
            user_lastvisit AS `lastvisit`
            FROM users WHERE user_id=:uid');
        $stmt -> bindValue(':uid',
            $session['session_user'], PDO::PARAM_INT);
        $stmt -> execute();
        $stmt -> setFetchMode(PDO::FETCH_CLASS, 'user',
            array(0 => false));
        if($this -> user = $stmt -> fetch())
        {
            $stmt -> closeCursor();
        }
        else
        {
            $stmt -> closeCursor();
            $this -> create();
        }
    }

```

Aby pobrać dane osoby zalogowanej, musimy wysłać jeszcze jedno zapytanie, którego wynik zwracamy jako klasę user. Identyczną sztuczkę stosowaliśmy już wcześniej, pisząc tamtą klasę. Tu jest ona niemal idealnie zdublowana. Jeżeli rekord udało się pobrać, zamykamy kursor, w przeciwnym razie musimy jeszcze powołać do życia sesję anonimową. To już koniec konstruktora:

```

    }
    else
    {
        $stmt -> closeCursor();
        $this -> create();
    }
} // end __construct();

```

Kolejnym etapem jest metoda *create()*. Nie ma w niej żadnych rozgałęzień. Po prostu generuje ona ID sesji na podstawie aktualnego czasu i adresu IP, dodaje odpowiedni rekord do bazy i wysyła ciastko, dodatkowo tworząc na końcu obiekt *user*.

```

private function create()
{
    global $pdo, $request;

```

```

$this -> garbageCollector();

// utworz nowa anonimowa sesje. Wczesniej usun stare z bazy
$this -> id = sha1(uniqid(time()).$request->getIp());
setcookie(COOKIE_NAME, $this -> id, time() + COOKIE_EXPIRE);
$stmt = $pdo -> prepare('INSERT INTO sessions (session_id,
    session_user, session_time, session_browser,
    session_ip) VALUES (
    :session_id, 0, :session_time,
    :session_browser, :session_ip
)');
$stmt -> bindValue(':session_id', $this -> id,
    PDO::PARAM_STR);
$stmt -> bindValue(':session_ip', $request -> getIp(),
    PDO::PARAM_STR);
$stmt -> bindValue(':session_browser',
    $request -> getBrowser(), PDO::PARAM_STR);
$stmt -> bindValue(':session_time', time(), PDO::PARAM_INT);
$stmt -> execute();
$this -> user = new user(true);
} // end create();

```

Zauważ, że to tutaj wywołujemy metodę do usuwania starych rekordów. Warunek czasu ważności nałożyliśmy w konstruktorze na sesje, zatem nie ma potrzeby inicjować czyszczenia przy każdym wejściu na stronę. Tworzenie nowej sesji jest wykonywane rzadziej i mniej obciąża bazę, a efekt będzie taki sam.

Teraz pora na metodę *update()*. Wspomniałem, że używać jej będziemy zarówno do logowania, jak i wylogowywania. W każdym przypadku generowany będzie nowy ID sesji oraz podmieniane dane w rekordzie. Dodatkowo, jeżeli przekazemy tutaj obiekt anonimowego użytkownika, oznaczać to będzie wylogowywanie i konieczność aktualizacji czasu ostatniej wizyty u osoby dotychczas zalogowanej. Od tego elementu właśnie rozpocznie się ta część kodu:

```

public function update(user $user)
{
    global $pdo, $request;

    if($user -> isAnonymous())
    {
        if($this -> user -> isAnonymous())
        {
            throw new Exception('Próba przerejestrowania
                anonimowego użytkownika!');
        }
        // Aktualizacja ostatnich odwiedzin, jeśli
        // wylogowujemy usera.
        $stmt = $pdo -> prepare('UPDATE users SET
            user_lastvisit = :lastvisit
            WHERE user_id = :uid');
        $stmt -> bindValue(':lastvisit', time(),
            PDO::PARAM_INT);
        $stmt -> bindValue(':uid', $this -> user -> getId(),
            PDO::PARAM_INT);
        $stmt -> execute();
    }
}

```

Zwróć uwagę na jedną rzecz. Jeśli zarówno przekazany obiekt, jak i ten znajdujący się już w sesji, są anonimowe, to generujemy wyjątek. Taka sytuacja nigdy nie powinna się zdarzyć, lecz jeśli wystąpi, nasza aplikacja zawiera jakiś błąd, który potencjalnie może zagrozić bezpieczeństwu. Dlatego też lepiej to zgłosić w jakiś widowiskowy sposób, niż próbować reagować automatycznie udając, że nic się nie stało. Analogiczny mechanizm możemy zastosować, jeżeli oba obiekty dotyczą osób zalogowanych. W ten sposób zostaną nam dwie poprawne politycznie możliwości: jeden obiekt jest anonimowy, drugi zalogowany (lub na odwrót).

Teraz końcówka tej metody, czyli aktualizacja danych sesji:

```

        // Zmiana ID sesji oraz przypisanie do niej usera
        $newId = sha1(uniqid(time().$request->getIp()));
        setcookie(COOKIE_NAME, $newId, time() + COOKIE_EXPIRE);
        $stmt = $pdo -> prepare('UPDATE sessions SET
            session_id = :new_id, session_user = :user
            WHERE session_id = :sid');
        $stmt -> bindValue(':new_id', $newId, PDO::PARAM_STR);
        $stmt -> bindValue(':sid', $this -> id, PDO::PARAM_STR);
        $stmt -> bindValue(':user', $user -> getId(),
            PDO::PARAM_INT);
        $stmt -> execute();

        $this -> id = $newId;
        $this -> user = $user;
    } // end update();

```

Oto zakończenie kodu klasy zawierające ostatnie metody:

```

private function garbageCollector()
{
    global $pdo;

    // Usun stare sesje i przenies do uzytkownikow
    // czas ostatniej aktywnosci jako ostatnia wizyte
    $pdo -> exec('UPDATE users, sessions
        SET users.user_lastvisit = sessions.session_time
        WHERE users.user_id=sessions.session_user AND
        sessions.session_time < '.(time() - COOKIE_EXPIRE));
    $pdo -> exec('DELETE FROM sessions WHERE
        session_time < '.(time() - COOKIE_EXPIRE));
} // end garbageCollector();

public function getUser()
{
    return $this -> user;
} // end getUser();
}
?>

```

Nadmienię tylko, że metoda *garbageCollector()* używa dwóch zapytań bez względu na to, ile rekordów sesji osób zalogowanych znajdowało się w bazie. Ot, taki przykład pokazujący, że inwestowanie w lepsze poznanie języka SQL zawsze owocuje lepszymi osiągnięciami. Większość początkujących (i nie tylko) nie wie nawet, że można zapytania UPDATE oraz DELETE wywoływać na kilku tabelach naraz, a tymczasem często pozwala to znacząco zmniejszyć ilość zapytań, szczególnie gdy dane połączone są wielką ilością relacji. Usuwanie tego wszystkiego ręcznie jest bardzo kłopotliwe, dlatego więc nie pokusić się o wykasowanie nieużywanych rekordów w pięciu tabelach jednym zapytaniem? To jest możliwe i zachęcam wszystkich do zgłębiania swej wiedzy o SQL'u.

Przykładowe użycie

Pokażemy teraz przykładowe użycie klasy *session*. Mamy trzy pliki:

1. *index.php* - treść ogólnodostępna.
2. *secure.php* - treść dostępna wyłącznie dla zalogowanych.
3. *login.php* - logowanie lub wylogowywanie.

Zacniemy od pliku *common.php*, który wykona wszystkie czynności administracyjne, tj. dołączy niezbędne biblioteki, utworzy ich obiekty oraz zainicjuje połączenie z bazą danych.

```

<?php
require('./request.php');
require('./user.php');

```

```

require('./session.php');

$request = new httpRequest;
$pdo = new PDO('mysql:host=localhost;port=3306;dbname=test',
    'root', 'root');
$pdo -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$session = new session;

?>

```

Plik *index.php* jest bardzo prosty. Ładujemy w nim wyżej podany kod i wyświetlamy dane. Dodatkowo, pokazujemy osobne komunikaty w zależności od tego, czy ktoś jest zalogowany, czy nie. Aby to sprawdzić, stosujemy metodę *isAnonymous()* dostępną w klasie *user*. Całość jest oczywiście zamknięta w bloku wyjątku.

```

<?php
    try
    {
        require('./common.php');

        if($session -> getUser() -> isAnonymous())
        {
            echo '<p>Witaj nieznanomy,
            <a href="login.php">Zaloguj sie</a>!</p>';
        }
        else
        {
            echo '<p>Witaj, '.$session -> getUser() -> getLogin().'!
            Ostatnio odwiedziłeś nas '.
            $session -> getUser() -> getLastvisit().'!
            (<a href="login.php">Wyloguj sie</a></p>';
        }

        echo 'To są dane dostępne dla wszystkich.
        <a href="secure.php">Tutaj są dane
        dostępne dla zalogowanych</a>.';
    }
    catch(PDOException $exception)
    {
        echo 'Błąd bazy danych: '.$exception->getMessage();
    }

?>

```

Plik *secure.php* będzie się różnić głównie tym, że w przypadku prawdziwości warunku *\$session -> getUser() -> isAnonymous()* przerzuci on nas do pliku *login.php*.

```

<?php
    try
    {
        require('./common.php');

        if($session -> getUser() -> isAnonymous())
        {
            header('Location: login.php');
            die();
        }
        else
        {
            echo '<p>Witaj, '.$session -> getUser() -> getLogin().'!
            Ostatnio odwiedziłeś nas '.
            $session -> getUser() -> getLastvisit().'!
            (<a href="login.php">Wyloguj sie</a></p>';
        }

        echo 'To są tajne dane.';
    }

```

```

    }
    catch(PDOException $exception)
    {
        echo 'Bład bazy danych: '.$exception->getMessage();
    }
?>

```

Ostatnim plikiem jest *login.php*. Musi on reagować inaczej w zależności od tego, czy ma do czynienia z osobą zalogowaną, czy nie. W pierwszym przypadku dokonuje on wylogowania, w drugim - wyświetla formularz z zapytaniem o login i hasło.

```

<?php
    try{
        ob_start();
        require('./common.php');

        if($session -> getUser() -> isAnonymous())
        {

            if($_SERVER['REQUEST_METHOD'] == 'POST')
            {
                // Dane przyszły z formularza, czas je sprawdzić
                $result = user::checkPassword($_POST['login'],
                    $_POST['haslo']);

                if($result instanceof user)
                {
                    $session -> update($result);
                    echo 'Dziekujemy, zostales zalogowany jako '.
                        $session -> getUser() -> getLogin().'.'.
                        <a href="index.php">Powrot</a>';
                }
                else
                {
                    echo 'Nieprawidlowy login i/lub haslo!';
                }
            }
            else
            {
                echo '<form method="post" action="login.php">
                    Login: <input type="text" name="login"/><br/>
                    Haslo: <input type="password" name="haslo"/><br/>
                    <input type="submit" value="Zaloguj"/>
                </form>';
            }
        }
        else
        {
            // Jezeli plik ten odpalony przez osobe zalogowana,
            // to ja wylogowujemy
            $session -> update(new user(true));
            echo 'Zostales wylogowany';
        }
        ob_end_flush();
    }
    catch(PDOException $exception)
    {
        echo 'Bład bazy danych: '.$exception->getMessage();
    }
?>

```

Zwróć uwagę na przebieg samego procesu kontroli. Najpierw pytamy metodę *user::checkPassword()*, czy przyslane dane są prawdziwe. Jeżeli tak, otrzymamy w ten sposób obiekt *user* z danymi osoby zalogowanej,

który możemy przerzucić do metody `$session -> update()` i zalogować daną osobę. W przypadku problemów otrzymamy wynik 0 i należy wtedy wyświetlić komunikat o błędnym loginie i hasle. Ponieważ w tym miejscu kod dosyć gęsto korzysta ciastek, na wszelki wypadek stosujemy funkcje kontroli wyjścia `ob_start()` oraz `ob_end_flush()` do zbuforowania kodu HTML, aby jedno drugiemu nie weszło w paradę. Na tym kończy się kod, możesz go spokojnie przetestować.

Zakończenie

Zaprojektowana przez nas klasa sesji jest dosyć elastyczna, niemniej do bardziej zaawansowanych zastosowań, np. obsługi uprawnień, będzie ona musiała zostać zintegrowana z mechanizmami autoryzacji, które najczęściej pisze się oddzielnie. Nie ma tu także żadnego systemu do przesyłania sesją jakichś dodatkowych danych, co jest podstawą implementacji sesji dostępnej w PHP. Dorobić takie coś nie jest trudno. Wystarczy dodatkowa tablica, metody magiczne `__get()` oraz `__set()` i funkcja `serialize()`. Zwróć uwagę, że tworząc własne systemy, nie zawsze musisz postępować według powyższego schematu. Podam tu przykład z mojej praktyki. Pisząc własne aplikacje, przygotowuję dwie klasy sesji. Jedna przeznaczona jest do użytku ogólnego, druga do panelu administracyjnego. Różnica w działaniu jest zasadnicza. W panelu administracyjnym nie ma sesji anonimowych. Jeżeli sesji nie ma, ktoś się jeszcze nie zalogował i jest wykopywany do formularza logowania.

Mam nadzieję, że artykuł wyjaśnił sprawy dotyczące tworzenia systemu sesji i zaowocuje on nowymi, może nawet lepszymi implementacjami. W razie jakichkolwiek problemów, zapraszam na forum WebCity.pl.

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**

Licencja

Artykuł rozpowszechniany jest na licencji **Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 2.5 Polska**.

Wolno:

- kopiować, rozpowszechniać, odtwarzać i wykonywać utwór

Na następujących warunkach:

1. *Uznanie autorstwa*. Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę*.
2. *Użycie niekomercyjne*. Nie wolno używać tego utworu do celów komercyjnych.
3. *Bez utworów zależnych*. Nie wolno zmieniać, przekształcać ani tworzyć nowych dzieł na podstawie tego utworu.

W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.

Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Powyższe postanowienia w żaden sposób nie naruszają uprawnień wynikających z dozwolonego użytku ani żadnych innych praw.

Internetowa skrócona wersja licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/>

Pełen tekst licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/legalcode>

Podczas publikacji należy podać następujące informacje:

1. Link do internetowej skróconej wersji licencji.
2. Informację o wersji publikowanego tekstu.
3. Informacje o autorze w następujący sposób:

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**