Krótki kurs tworzenia stron w technologii PHP

Mateusz Gałażyn, Jeremi Niedziela

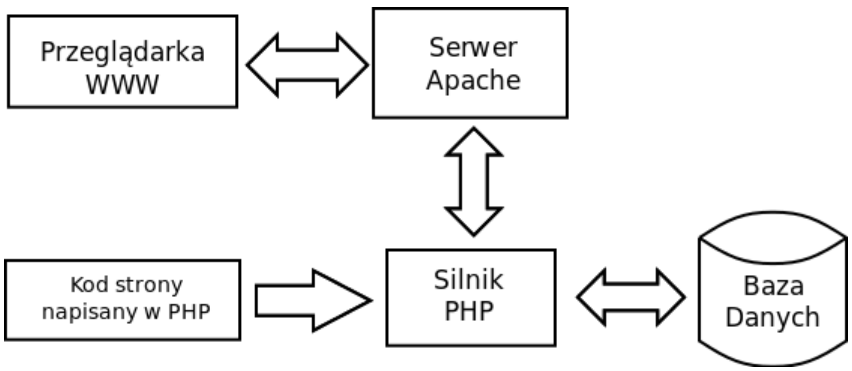
6 maja 2012

1 Wstęp

Ten krótki kurs ma na celu przedstawienie sposobu pisania stron przy użyciu technologii Apache + MySQL + PHP (w skrócie AMP) - jednego z najpopularniejszych zestawów oprogramowania służącego do uruchamiania serwisów WWW. Ten kurs jest bardzo okrojony i zawiera tylko wybrane elementy języka PHP oraz MySQL dlatego aby w pełni zrozumieć metodykę tworzenia aplikacji w oparciu o te języki dobrze jest się wspierać dokumentacjami technicznymi, innymi kursami dostępnymi w internecie, a także specjalistycznymi forami dyskusyjnymi.
Dokumentacja PHP: http://www.php.net/manual/pl/
Dokumentacja MySQL: http://dev.mysql.com/doc/refman/5.5/en/index.html
Dokumentacja Apache: http://httpd.apache.org/docs/2.4/
Kurs PHP @ Wikibooks: http://pl.wikibooks.org/wiki/PHP

1.1 Mechanizm działania systemu AMP

Gdy użytkownik strony uruchamia przeglądarkę i wpisuje w pasek adresu, adres szukanego serwisu WWW, przeglądarka nawiązuje połączenie z serwerem na którym są uruchomione usługi umożliwiające dostęp do strony.
Apache - jest to najszerzej stosowany w internecie serwer HTTP
PHP - jeden z najpopularniejszych języków programowania używany do tworzenia stron WWW.
MySQL - system zarządzania bazami danych za pomocą języka SQL



Rysunek 1: Schemat komunikacji zestawu AMP

Żądanie otrzymane od przeglądarki jest przechwytywane przez serwer Apache, który przetwarzając je uruchamia kod strony napisany w języku PHP. Następnie silnik PHP komunikuje się z bazą danych, pobiera dane, przetwarza je i zamienia na kod HTML, który zwraca serwerowi Apache. W kolejnym kroku serwer Apache wysyła kod HTML razem z obrazami umieszczonymi na stronie i stylami do przeglądarki, która renderuje i wyświetla stronę.

2 Instalacja środowiska AMP

2.1 Instalacja na systemie Windows

2.1.1 Instalacja serwera Apache

2.1.2 Instalacja PHP

2.1.3 Instalacja serwera MySQL

2.2 Instalacja na systemie Linux

2.2.1 Instalacja serwera Apache

2.2.2 Instalacja PHP

2.2.3 Instalacja serwera MySQL

2.3 Konfiguracja

2.3.1 Konfiguracja Apache

2.3.2 Konfiguracja PHP

2.3.3 Konfiguracja MySQL

2.4 Instalacja phpMyAdmin

3 Projekt forum dyskusyjnego

Naszym celem jest jest zbudowanie prostego forum dyskusyjnego. Musimy dać możliwość rejestrowania się poszczególnym użytkownikom, dodawania własnych wątków oraz odpowiadania na już utworzone. Potrzebne będzie też konto administratora, który będzie miał prawo moderacji odpowiedzi w tematach (postów).

3.1 Układ podstron

Gdy zostanie już ustalona lista wymaganych funkcjonalności projektowanego serwisu, następnym etapem jest stworzenie układu poszczególnych podstron.

Strona główna Tutaj trzeba wyświetlić listę tematów (wraz z odnośnikami do podstrony, na którym będzie pojedynczy wątek wyświetlany w całości) które zostały założone na forum, umieścić odnośniki do podstrony z formularzem używanym do zalogowania się i rejestracji dla użytkowników oraz odnośnik do podstrony umożliwiającej założenie nowego tematu.

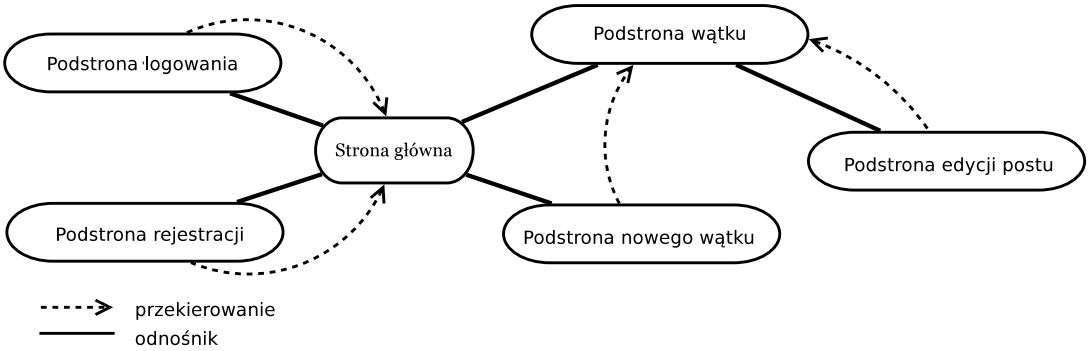
Podstrona wątku W tym miejscu trzeba wyświetlić wszystkie posty w danym wątku, oraz formularz dający możliwość odpowiedzi w tym wątku. Gdy administrator wejdzie na tą podstronę, trzeba dać także dodatkowe możliwości edycji i usuwania poszczególnych postów.

Podstrona logowania Zawierać będzie formularz logowania umożliwiający uwierzytelnienie użytkownika wchodzącego stronę. Uwierzytelnienie będzie polegało na porównaniu nazwy użytkownika i hasła z obecnymi w bazie danych. Po zalogowaniu się, użytkownik zostanie przekierowany z powrotem na stronę główną. Podstrona dostępna tylko dla niezalogowanych użytkowników.

Podstrona rejestracji Odpowiedzialna za obsługę rejestracji nowego użytkownika. Trzeba będzie wyświetlić formularz rejestracji, odebrać i zweryfikować dane oraz dodać je do bazy danych.

Podstrona nowego wątku Podstrona z formularzem umożliwiającym dodanie nowego wątku. Po odebraniu danych od użytkownika trzeba będzie dokonać ich weryfikacji i dodać do bazy danych. Na koniec trzeba przekierować użytkownika do podstrony wątku.

Podstrona edycji postu Podstrona z formularzem umożliwiającym edycję własnych postów, a także w przypadku administratora - edycji każdego postu. Przekierowuje z powrotem do podstrony wyświetlającej cały wątek.



Rysunek 2: Układ podstron forum

3.2 Projekt bazy danych

Informacje w bazie danych są przechowywane w postaci tabel, podobnie jak w arkuszach kalkulacyjnych dostępnych w popularnych pakietach biurowych. W bazie projektowanego forum będziemy przechowywać listę użytkowników, listę tematów oraz listę postów. Dla każdej z tych list trzeba utworzyć oddzielną tabelę: users, threads, posts. W każdej z tych tabel konieczne jest stworzenie odpowiednich kolumn służących do przechowywania danych we właściwych typach. Ustawienie typu danej przechowywanej w każdej kolumnie przyśpiesza pracę serwera MySQL poprzez automatyczną optymalizację realizowanych zapytań do bazy oraz optymalizację sposobu przechowywania tabel na dysku.

3.2.1 Typy danych MySQL

W języku MySQL jest wiele typów danych (dokładny opis: http://dev.mysql.com/doc/refman/5.5/en/data-type-overview.html), te najważniejsze to:

Typy znakowe:

CHAR Przechowuje ciąg znaków do 255 elementów.

VARCHAR Tak jak **CHAR** przechowuje ciąg znaków do 255 elementów. Główną różnicą pomiędzy nimi jest to, że **VARCHAR** zmienia swój rozmiar w zależności od przyjętej liczby znaków, a **CHAR** zawsze rezerwuje miejsce 255 znaków niezależnie od długości zapisywanego ciągu.

TEXT Przechowuje ciąg do 65535 znaków.

MEDIUMTEXT Przechowuje ciąg do 16777215 znaków.

Typy liczbowe:

TINYINY Przechowuje liczby od -128 do 127 (0 do 255 w przypadku **UNSIGNED**).

SMALLINT Przechowuje liczby od -32768 to 32767 (0 do 65535 w przypadku **UNSIGNED**).

MEDIUMINT Przechowuje liczby od -8388608 do 8388607 (0 do 16777215 w przypadku **UNSIGNED**).

FLOAT Małe liczby zmiennoprzecinkowe.

DOUBLE Liczby zmiennoprzecinkowe o podwojonej precyzji.

Każdy typ liczbowy występuje także w wersji **UNSIGNED**, która pozwala przechowywać dwa razy większe liczby bez przechowywania informacji o znaku.

Inne typy:

DATE Data w formacie YYYY-MM-DD

DATETIME Data w formacie YYYY-MM-DD HH:MM:SS

TIMESTAMP Znacznik czasu w formacie: YYYYMMDDHHMMSS

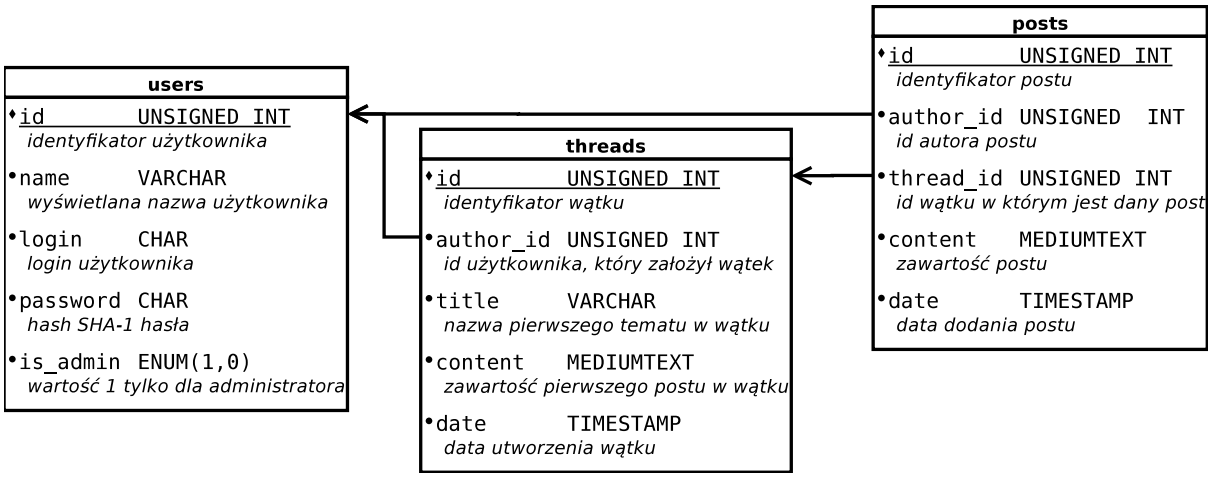
TIME Czas w formacie: HH:MM:SS

ENUM Podobnie jak w C++, definiuje listę określonych wartości jakie może przyjąć to pole.

SET Podobnie jak **ENUM**, lecz pole w tabeli może przyjąć kilka wartości na raz.

3.2.2 Projekty tabel

Tabela *users* powinna zawierać kolumny przechowujące informacje o użytkowniku takie jak: nazwa użytkownika, login używany przy identyfikacji na stronie, hash SHA-1[1] hasła oraz pole determinujące czy dany użytkownik ma uprawnienia administracyjne (wartość 1, gdy użytkownik jest administratorem).
Tabela *threads* powinna zawierać kolumny przechowujące tytuł wątku, zawartość pierwszego posta oraz data utworzenia wątku.
Tabela *posts* powinna zawierać kolumny przechowujące treść posta i datę jego utworzenia.
Dodatkowo w każdej z tabel powinna znajdować się kolumna *id* identyfikująca w każdej tabeli pojedynczy rekord. To rozwiązania ułatwia odwoływanie się do poszczególnych wpisów (rekordów) w tabeli, bez konieczności ponownego przeszukiwania całej tabeli. W tabelach *posts* i *threads* oprócz informacji o postach i wątkach musimy utworzyć tak zwane relacje - kolumny wiążące poszczególne posty z użytkownikami oraz wątkami. Jest to tak zwana relacja **jeden do wielu**. Zagadnienie to można rozwiązać w prosty sposób: w tabeli *threads* tworzymy kolumnę *author_id*, którą będzie zawierała id rekordu w tabeli *users* odpowiadającego użytkownikowi, który stworzył dany wątek. Analogiczną operację trzeba wykonać dla tabeli posts: tworzymy kolumnę *author_id* oraz dodatkową *thread_id*, która definiuje wątek do którego przynależy dany post.



Rysunek 3: Schematy tabel i ich wzajemne relacje

Optymalizacja Oprócz precyzowania typów danych dla każdej kolumny, aby ułatwić sobie i serwerowi pracę można zdefiniować dodatkowe atrybuty dla poszczególnych kolumn. Kolumna *id* występująca w każdej z tabel musi być zdefiniowana z atrybutami *PRIMARY KEY* (co jest zaznaczone na schemacie podkreśleniem) - oznacza to, że kolumna jest wykorzystywana do identyfikacji rekordów w tabeli, *AUTO_INCREMENT* - każdy dodawany rekord ma ustawiane automatycznie *id* o 1 większe niż największe *id* w tabeli, *NOT NULL* - nie może mieć wartości 0.

[1] W bazach danych **NIGDY** nie powinno się przechowywać haseł w jawnej postaci tekstu. W przypadku gdy osoba trzecia uzyska nieautoryzowany dostęp do bazy danych, będzie mogła skopiować listę haseł wszystkich użytkowników, co nie powinno mieć nigdy miejsca. Zamiast tego przechowuje się specjalne hashe wygenerowane na podstawie haseł. Więcej informacji na temat hashy: pl.wikipedia.org

3.2.3 Kod SQL

Nadszedł czas na stworzenie zaprojektowanej przez nas struktury tabel. Uruchamiamy serwer MySQL, a następnie poleceniem (dla systemu Linux):

```
mysql -u root -p
```

uruchamiamy wiersz poleceń MySQL. Na systemie Windows aby uruchomić wiersz poleceń, wpierw trzeba uruchomić terminal, a w nim wpisać:

```
C:> cd C:\sciezka\do\katalogu\mysql\bin
C:\sciezka\do\katalogu\mysql\bin> mysql.exe -u root -p
```

Następnie po wpisaniu hasła, aby stworzyć bazę danych wydajemy polecenie:

```
CREATE DATABASE forum;
```

Aby przystąpić do tworzenia tabel, trzeba wybrać nowo utworzoną bazę danych:

```
USE forum;
```

Możemy teraz utworzyć tabelę *users* wydając poniższe polecenie:

```
1  CREATE TABLE `forum`.`users` (
2          `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
3          `name` VARCHAR( 255 ) NOT NULL ,
4          `login` CHAR( 255 ) NOT NULL ,
5          `password` CHAR( 255 ) NOT NULL ,
6          `is_admin` ENUM( '1', '0' ) NOT NULL ,
7          UNIQUE (
8                  `name` ,
9                  `login`
10         )
11 );
```

Ważne jest umieszczenie na końcu wyrażenia średnika, ponieważ informuje to interpreter MySQL że to jest koniec komendy. Bez średnika interpreter będzie czekał na dalsze wprowadzanie komend, dopóki nie napotka średnika. W linii 1 podajemy nazwę bazy danych, a po kropce jest nazwa tworzonej tabeli. Nazwy bazy, tabel i kolumn są wzięte w ` `. W liniach 2-6 zdefiniowane są kolumny tabeli oddzielone przecinkami. Składnia jest następująca:

```
`nazwa_kolumny` TYP NULL [dodatkowe atrybuty] ,
```

Na początku podajemy nazwę kolumny, potem nazwę typu, jeżeli jest to typ liczbowy bez znaku, za nazwą typu dodajemy *UNSIGNED*. Jeżeli jest to typ znakowy *CHAR* lub *VARCHAR* to wpisujemy w nawiasie maksymalną długość ciągu znaków. Jeżeli jest to typ *ENUM*, lub *SET* w nawiasie podajemy dopuszczalne wartości. Następnie podajemy informację czy w tą kolumnę może zostać wpisana wartość pusta - parametr *NULL*, lub gdy nie: *NOT NULL*. Jeżeli chcemy ustawić dodatkowe opcje dla kolumny wpisujemy je oddzielając spacją. Dla kolumny *id* zostały ustawione opcje *AUTO_INCREMENT* oraz *PRIMARY KEY*. Po wymienionych kolumnach można zdefiniować dodatkowe indeksy, w linii 7 zdefiniowany jest indeks *UNIQUE*: dla kolumn umieszczonych w nawiasie wartości w pól w rekordach nie mogą się powtarzać.

Tabela *threads*:

```
1  CREATE TABLE `forum`.`threads` (
2          `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
3          `author_id` INT UNSIGNED NOT NULL ,
4          `title` VARCHAR( 255 ) NOT NULL ,
5          `content` MEDIUMTEXT NOT NULL ,
6          `date` TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
7  );
```

Kod jest prawie identyczny jak w poprzednim przypadku. Różnica pojawia się jedynie w linii 6. Po typie danych *TIMESTAMP* zdefiniowana jest opcja *ON UPDATE CURRENT_TIMESTAMP*, która mówi, że po aktualizacji rekordu silnik bazy danych automatycznie umieści w kolumnie *date* aktualny znacznik czasu. Następnie jest podana informacja, że wartość pola nie może być wartością *NULL*. Parametr *DEFAULT CURRENT_TIMESTAMP* mówi nam, że gdy przy wpisywaniu wartości do bazy danych nie podamy wartości tego pola, serwer MySQL automatycznie wypełni to pole aktualną godziną.

Tabela *posts*:

```
CREATE TABLE `forum`.`posts` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`author_id` INT UNSIGNED NOT NULL ,
`thread_id` INT UNSIGNED NOT NULL ,
`content` MEDIUMTEXT NOT NULL ,
`date` TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Składnia jest analogiczna jak w poprzednim wypadku. Powyższe trzy zapytania *CREATE* tworzą już kompletną, ale jeszcze pustą strukturę naszej bazy danych.

4 Kod strony

Przed przystąpieniem do pracy nad kodem strony, warto wyposażyć się w jakikolwiek edytor z kolorowaniem składni. Na systemy z rodziny Windows dobrym rozwiązaniem jest Notepad++, dostępny ze strony: http://notepad-plus-plus.org/download/v6.1.2.html. Na systemach GNU (w tym tych z rodziny Linux) może to być Emacs, Vim, Nano lub używany w środowisku Gnome, Gedit.

4.1 Składnia języka PHP

Przykładowy kod strony w języku php:

```
<?php
$jeden = 'bar';
$dwa = "foo";
$trzy = null;
$liczba = "4";
$liczba2 = 3;
if($liczba % 2 == 0)
{
        echo 'liczba jest podzielna przez 2 <br />';
}
else
{
        echo 'liczba nie jest podzielna przez 2 <br />';
}

if($liczba == 4)
        echo 'Ta liczba to 4!';
elseif($liczba < 4)
        echo 'Ta liczba jest mniejsza od 4.<br />';
else
        echo 'Ta liczba jest wieksza od 4 <br />';
echo $jeden.$dwa.($liczba+$liczba2);
?>
```

Zapisując powyższy kod jako plik *index.php* w katalogu *DocumentRoot* zdefiniowanym przy Konfiguracji Apache, a następnie wchodząc na stronę http://127.0.0.1/ otrzymamy wynik:

```
liczba jest podzielna przez 2
Ta liczba to 4!
barfoo7
```

Kod PHP aby został wykonany musi być zawarty w tagach *<?php ?>*. W linijkach 2 - 6 zostały zdefiniowane zmienne. Nazwy zmiennych muszą być poprzedzone znakiem $ zarówno przy definiowaniu jak i późniejszym odwoływaniu się do nich. Ciąg znaków przypisywany do zmiennej można zawszeć w apostrofach lub cudzysłowach. Między tymi dwoma sposobami jest niewielka różnica: gdy zawrzemy ciąg znaków w cudzysłowach PHP przeszukuje zawartość ciągu pod kątem znaków specjalnych (np.: \n \r itp.), w przypadku apostrofów przeszukiwanie nie jest dokonywane (ta forma jest nieco szybsza). W liniach 7 - 21 są umieszczone struktury *if - else*. W linii 7 jest wykonywane modulo na zmiennej *$liczba*,

a następnie wynik tej operacji jest przyrównywany do zera. Należy zwrócić uwagę iż w PHP brak jest typów zmiennych - zależnie od potrzeb zmienna jest automatycznie traktowana jako ciąg znaków, lub jako liczba. W linii 9 do wyświetlania jest używana struktura języka: *echo*. W linii 22 został użyty operator konkatenacji (kropka) do połączenia trzech ciągów znaków w jeden. Najpierw zostało wykonane dodawanie dwóch liczb w nawiasie, a następnie ich suma została zamieniona na ciąg znaków, który został doklejony do fraz 'bar' i 'foo'. W PHP operator kropki nie jest używany w odniesieniu do obiektów (jak to ma miejsce w Javie i w C++).
Przykładowa klasa w PHP:

```
1  <?php
2  class Telewizor
3  {
4          private static $iloscTelewizorow = 0;
5
6          private $aktualnyKanal = 0;
7          private $poziomGlosnosci = 50;
8
9          public static function pobierzLiczbeTelewizorow()
10         {
11                 return self::$iloscTelewizorow;
12         }
13
14         public function __construct()
15         {
16                 ++self::$iloscTelewizorow;
17         }
18
19         public function ustawGlosnosc($poziomGlosnosci)
20         {
21                 $this->poziomGlosnosci = $poziomGlosnosci;
22         }
23
24         public function ustawKanal($nrKanalu = 0)
25         {
26                 $this->aktualnyKanal = $nrKanalu;
27         }
28 };
29
30 function wyswietlaj()
31 {
32         echo 'teraz wyswietlaja '.Telewizor::pobierzLiczbeTelewizorow().' telewizory';
33 }
34
35 new Telewizor;
36 new Telewizor;
37 $sharp =  new Telewizor;
38 $sharp->ustawKanal();
39 $sharp->ustawKanal(13);
40 $sharp->ustawGlosnosc(30);
41 $sharp->ustawGlosnosc(10);
42
43 wyswietlaj();
44 ?>
```

Wynik:

```
teraz wyswietlaja 3 telewizory
```

Składnia jest bardzo podobna do tej znanej z C++, za wyjątkiem braków typów zmiennych. W liniach 4 - 7 zostały zadeklarowane i zainicjalizowane domyślnymi wartościami właściwości klasy. W linii 11 jest

odwołanie do statycznej właściwości klasy. W PHP dostęp do statycznych pól klasy jest uzyskiwany za pomocą wyrażenia self i operatora dostępu "::". W linii 14 znajduje się konstruktor, którego zadaniem jest zwiększanie ilości stworzonych telewizorów. W PHP konstruktory i destruktory noszą zawsze nazwę *__construct()* i *__destruct()* niezależnie od nazwy klasy. W deklaracji każdej funkcji i metody zawsze znajduje się słowo kluczowe function. W linii 21 jest ustawiany poziom głośności w telewizorze. Dostęp do właściwości obiektów uzyskać można tylko za pomocą wskaźnika *$this* wskazującego na aktualny obiekt, dostępnego w każdej **niestatycznej** metodzie klasy. Należy zwrócić uwagę, że gdy odwołujemy się do właściwości klasy za pomocą *$this* nie używamy już *$* przed nazwą pola klasy. W linii 24 została zdefiniowana funkcja przyjmująca parametr, który ma już wartość domyślną. Jest to pewien odpowiednik znanego z C++ przeciążania funkcji, gdy kilka funkcji mogło mieć tą samą nazwę - różnił je tylko zestaw argumentów. W PHP przeciążanie funkcji znane z C++ jest niedozwolone, zamiast tego stosuje się wartości domyślne argumentów. Pozwala to na wywołanie funkcji bez podawania argumentów, którym zostały przypisane wartości domyślne w nagłówku funkcji. Przykład takiego wywołania jest w linii 38 i 39. Obiekty tworzy się za pomocą operatora *new*, przykład znajduje się w liniach 35 - 37. W liniach 35 i 36 tworzone są obiekty klasy Telewizor, które nie są do niczego przypisywane. W linii 37 tworzony obiekt tej samej klasy przypisany do zmiennej *$sharp*. Dostęp do pól i metod uzyskiwany jest za pomocą operatora -> .

Zagnieżdżanie wewnątrz kodu HTML PHP umożliwia przeplatanie kodu HTML i PHP, co umożliwia generowanie zawartości stron w czasie rzeczywistym. Przykład kodu:

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4   <title>Forum</title>
5   <meta charset="UTF-8" />
6  </head>
7  <body>
8  <?php if(!is_object(User::$oCurrentUser)):?>
9  <a href="register.php">rejestracja</a>
10 <?php else: ?>
11 Witaj <b><?php echo User::$oCurrentUser->sName; ?></b>
12 <?php endif; ?>
13 <br /><br />
14 <?php if(count($aThreads) > 0):?>
15 Lista watkow na forum:
16 <table border="1">
17 <?php foreach($aThreads as $i => $oThread): ?>
18 <tr>
19  <td><?php echo $i+1 ?></td>
20  <td>
21 <a href="thread.php?id=<?php echo $oThread->iId ?>">
22 <?php echo $oThread->sTitle ?></a>
23 <?php
24  if(is_object(User::$oCurrentUser))
25          if(User::$oCurrentUser->bIsAdmin)
26                  echo ' <a href="delete.php?tid='.$oThread->iId.'">[ usun ]</a>';
27 ?></td>
28  <td><?php echo $oThread->sAuthorName ?></td>
29  <td><?php echo $oThread->sDate ?></td>
30 </tr>
31 <?php endforeach?>
32 </table>
33 <?php else: ?>
34 Brak watkow na forum.
35 <?php endif;?>
36 </body>
37 </html>
```

W linii 8 jest wstawiona funkcja *if()*. Należy zwrócić uwagę na dwukropek postawiony za nawiasem - w tym wypadku cała zawartość aż do linii 10, w której występuje *else:* jest wyświetlana gdy warunek jest spełniony[2]. Koniec *if()* jest w linii 12. W linii 11 wyświetlane jest pole *sName* zmiennej *User::$oCurrentUser* za pomocą wyrażenia *echo*. W linii 17 występuje wyrażenie *foreach()*. Ten zapis należy zrozumieć jako: dla każdego elementu tablicy *$aThreads* wykonaj pętlę, ale na początku pętli przypisz ten element na którym będziemy operować do zmiennej $oThread, a klucz pod jakim ten element znajdował się w tablicy $aThreads do zmiennej *$i*. W przypadku zwykłych tablic, w których jest po prostu lista elementów, w zmiennej *$i* będzie znajdować się numer elementu w tablicy (w PHP tablice są numerowane od 0, tak jak w C/C++), a gdy *$aThreads* będzie tablicą asocjacyjną (taką, w której kluczami są ciągi znaków) *$i* będzie zawierało ciąg znaków, który był kluczem dla wartości *$oThread*. Koniec pętli *foreach()* jest w linii 31.
Szczegółowe informacje na temat budowy języka można znaleźć w podręczniku PHP:
http://www.php.net/manual/pl/langref.php.

4.2 Struktura plików forum

Kod tworzonego forum będzie pogrupowany w następujące pliki:

dao/Thread.php Klasa reprezentująca pojedynczy wątek. Umożliwia tworzenie nowego wątku, modyfikację i listowanie wszystkich odpowiedzi.

dao/Post.php Klasa reprezentująca pojedynczą odpowiedź na wątek. Umożliwia tworzenie i modyfikację poszczególnych postów.

dao/User.php Klasa reprezentująca użytkownika - używana do zarządzania i identyfikacji użytkowników forum.

add.php Podstrona odpowiedzialna za dodawanie nowego wątku.

init.php Plik załączany w każdej podstronie odpowiedzialny za tworzenie połączenia z bazą danych i za dołączenie wymaganych plików.

index.php Strona główna forum. Wyświetla listę wątków na forum.

delete.php Podstrona odpowiedzialna za usuwanie wątków i odpowiedzi na forum.

register.php Podstrona rejestracji nowego użytkownika.

config.php Plik zawierający dane do połączenia z bazą danych.

edit.php Podstrona edycji wątku i postu.

login.php Podstrona odpowiedzialna za logowanie się użytkownika na forum.

thread.php Podstrona wyświetlająca zawartość całego wątku.

Nazwa folderu dao jest skrótem od Data Access Object. Zawiera on Klasy udostępniające warstwę abstrakcji pomiędzy kodem strony a bazą danych.

4.3 Zawartość poszczególnych plików

Na początek stwórzmy zawartość katalogu *dao/* - obiekty umożliwiające dostęp do bazy danych.

4.3.1 dao/Post.php

Klasa umożliwiająca zarządzanie odpowiedziami, będzie zawierać metody *save()* oraz *delete()*, które będą odpowiedzialne za zapisywanie posta do bazy (a także modyfikowanie już istniejącego). W poniższych listingach została przyjęta konwencja poprzedzania nazw zmiennych literą reprezentującą umowny typ wartości jaki będzie w danej zmiennej przechowywany. Odpowiednio i - integer, s - string, a - array, b - bool.

[2] dokładnie: gdy statyczne pole klasy *User::$oCurrentUser* nie jest obiektem - służy do tego funkcja *is_object()*

Wskazówka: W trakcie pisania kodu, gdy nasza zaprojektowana aplikacja nie będzie działać jak powinna, można podczas debuggowania kodu wspomóc się funkcją *var_ dump()*[3].

```
1  <?php
2  class Post
3  {
4          static public $oConnection = null;
5
6          public $iId = null;
7          public $iAuthorId = null;
8          public $sAuthorName = null;
9          public $iThreadId = null;
10         public $sContent = null;
11         public $sDate = null;
12
13         public function __construct($iId = null){}
14
15         public function save(){}
16
17         public function delete(){}
18 };
19 ?>
```

W linii 4 zdefiniowane jest statyczne pole *$oConnection*, które będzie reprezentować obiekt klasy PDO (dostarczonej wraz z PHP). Klasa PDO jest używana do komunikacji z bazą danych. W każdej z klas Post, Thread, User zostało umieszczone statyczne pole, które będzie przechowywało referencję do obiektu klasy PDO, aby móc się komunikować z bazą danych.
W liniach 6-11 są umieszczone pola reprezentujące odpowiednie kolumny w bazie danych, odpowiednio kolumny id, author_id, thread_id, content, date. Dodatkowo została zadeklarowane pole *$sAuthorName*, która będzie przechowywała nazwę autora postu. W linii 13 został zadeklarowany konstruktor przyjmujący domyślnie wartość *null*. Gdy będzie tworzony nowy post, do konstruktora nie trzeba podawać żadnego parametru. W przypadku gdy będzie potrzeba stworzenia obiektu reprezentującego już istniejący post, zmienna *$iId* będzie reprezentowała id odpowiedniego rekordu już istniejącego w bazie.
Zawartość metody **__construct()** (która znajduje się pomiędzy klamrami w wyżej wymienionym listingu):

[3] http://www.php.net/manual/en/function.var-dump.php

```php
<?php
public function __construct($iId = null)
{
        $this->iId = $iId;
        if($iId != null)
        {
                $oStatement = self::$oConnection->prepare('
                                SELECT *,`posts`.`id` AS `pid`
                                FROM `posts`,`users`
                                WHERE `posts`.`author_id`=`users`.`id`
                                AND `posts`.`id` = :id
                                ORDER BY `posts`.`id` ASC');
                $oStatement->bindValue(':id', $iId, PDO::PARAM_INT);
                $oStatement->execute();
                if($oStatement->errorCode() != 0)
                {
                        $aError = $oStatement->errorInfo();
                        throw new Exception($aError[2]);
                }
                if($aResult = $oStatement->fetch(PDO::FETCH_ASSOC))
                {
                        $this->iId = $aResult['pid'];
                        $this->iAuthorId = $aResult['author_id'];
                        $this->sAuthorName = $aResult['name'];
                        $this->iThreadId = $aResult['thread_id'];
                        $this->sContent = $aResult['content'];
                        $this->sDate = $aResult['date'];
                }
                else
                        $this->iId = null;
        }
}
?>
```

W linii 5, jeżeli zostało podane id postu rozpoczyna się procedura odczytu z bazy danych. W pierwszym kroku konieczne jest utworzenie szkieletu zapytania za pomocą metody *prepare()*, do której jako parametr przekazujemy zapytanie. Aby wyciągnąć dane z bazy trzeba użyć zapytania *SELECT*, którego składnia jest następująca:

```
SELECT `kolumna1`,`kolumna2`,`kolumna3` FROM `tabela`;
```

Oznacza to, że pobieramy dane z kolumn: `kolumna1`, `kolumna2` i `kolumna3` z tabeli `tabela`. Zamiast listy kolumn można także podać gwiazdkę *, która oznacza, że wybieramy z tabeli wszystkie kolumny. Można w jednym zapytaniu *SELECT* pobierać dane z kilku tabel na raz - trzeba wtedy je wymienić po klauzuli *FROM* oddzielając je przecinkami. Gdy odwołujemy się do kilku tabel na raz, należy pamiętać o poprzedzeniu nazw kolumn nazwą tabeli. Zapytanie SQL wyglądałoby w ten sposób:

```
SELECT `tabela`.`kolumna1`,`tabela`.`kolumna2`,`tabela`.`kolumna3`,`tabela2`.`kolumna5`
FROM `tabela`,`tabela2`;
```

Kolejny przykład:

```
SELECT `kolumna1`,`kolumna2`,`kolumna3`
FROM `tabela`
WHERE `kolumna1` = `kolumna2`
ORDER BY `kolumna1` ASC;
```

Powyższe zapytanie zwraca tylko te rekordy z tabeli `tabela`, w których wartości w kolumnie `kolumna1` są identyczne jak w `kolumna`. Jak widać słowo kluczowe *WHERE* służy do tworzenia warunków, jakie muszą spełniać zwrócone rekordy. Warunki w części zapytania po *WHERE* mogą być łączone za pomocą operatorów *OR* oraz *AND*. Wiersz który zawiera frazę *ORDER BY* mówi serwerowi MySQL, żeby zwrócone wyniki zostały posortowane według zawartości kolumny `kolumna1` rosnąco (wyraz *ASC* - ascending). Można także posortować wyniki malejąco: *DESC* - descending. Fragmenty zapytania z *WHERE* oraz *ORDER BY* są opcjonalne. W zapytaniu można dodatkowo przypisać zwracanym kolum-

nom nową nazwę - służy do tego słowo klucz *AS*:

```
SELECT `kolumna1`,`kolumna2` AS `ilosc`,`kolumna3`
FROM `tabela`
WHERE `kolumna1` = `kolumna2`
ORDER BY `kolumna1` ASC;
```

W powyższym zapytaniu, zwrócony wynik będzie zawierał kolumnę ‘kolumna2‘ nazwaną jako ‘ilosc‘. Wracając do zapytania przekazanego jako argument do metody *prepare()*[4], wybieramy za pomocą niego wszystkie kolumny z tabel ‘posts‘ i ‘users‘, nazywając kolumnę ‘id‘ w tabeli ‘posts‘ jako ‘pid‘, a następnie pobieramy rekordy z obydwu tabel, dla których id postu odpowiada id użytkownika w tabeli ‘users‘ (klauzula *WHERE*: ‘posts‘.‘author_id‘=‘users‘.‘id‘) oraz id postu jest równe numerowi id przez nas poszukiwanemu. Na koniec zwrócone dane są posortowane względem id postu. To zapytanie powinno zwrócić jeden wynik (ponieważ, kolumna ‘id‘ jest kluczem głównym w tabeli ‘posts‘) - post przez nas poszukiwany, dodatkowo wraz z nazwą autora postu w kolumnie ‘name‘. W zapytaniu nie zostało wpisane id przez nas poszukiwane - zamiast tego został umieszczony tag *:id*, który za pomocą funkcji *bindValue()*[5] jest zastępowany wartością id postu. To rozwiązanie ma na celu oczyszczenie danych wejściowych z niebezpiecznych znaków, które mogłyby zepsuć składnię zapytania i uniemożliwić jego poprawne wykonanie (a poprzez celową manipulację możliwe by było nawet skasowanie zawartości bazy[6]). Metoda *prepare()* zwraca obiekt typu *PDOStatement* reprezentujący zapytanie które jest wysyłane do bazy danych.
W linii 14 zapytanie jest wykonywane. W linii 15 jest sprawdzana poprawność wysłanego zapytania, gdy zapytanie zostanie wykonane niepoprawnie zostanie rzucony wyjątek w linii 18, który będzie zawierał informację o błędzie który wystąpił. Funkcja *errorInfo()*[7] zwraca dostępną informację o błędzie w formie tablicy. W trzecim elemencie (o indeksie 2) znajduje się informacja opisowa o błędzie, którą przekazujemy konstruktorowi klasy Exception.
W linii 20, gdy zapytanie zostało wykonane bez żadnego błędu zostaje przypisana do zmiennej $aResult tablica asocjacyjna (decyduje o tym parametr *PDO::FETCH_ASSOC* przekazany do metody *fetch()*[8]). Jeżeli *$aResult* jest niepuste (co w PHP jest równoznaczne wartości true) zostanie spełniony warunek i w liniach 22-27 dane ze zwróconej tablicy asocjacyjnej są przypisywane odpowiednim właściwościom obiektu. Należy zwrócić uwagę, na to, że klucze w zwróconej tablicy odpowiadają kolumnom zdefiniowanym w zapytaniu. Jeżeli warunek z linii 20 nie zostanie spełniony, przypisujemy właściwości *$iId* wartość *null*, która odpowiada operacji tworzenia nowego indeksu. Wartość pola *$iId* jest rozpoznawana w metodzie *save()*.
Zawartość metody **save()**:

[4] http://www.php.net/manual/en/pdo.prepare.php
[5] http://www.php.net/manual/en/pdostatement.bindvalue.php
[6] http://pl.wikipedia.org/wiki/SQL_injection
[7] http://www.php.net/manual/en/pdostatement.errorinfo.php
[8] http://www.php.net/manual/en/pdostatement.fetch.php

```
1  <?php
2  public function save()
3  {
4          if($this->iId != null) // update
5          {
6                  $oStatement = self::$oConnection->prepare('
7                          UPDATE `posts`
8                          SET
9                                  `author_id` = :aid,
10                                 `thread_id` = :tid,
11                                 `content` = :content
12                         WHERE `id` =:id ');
13                 $oStatement->bindValue(':id', $this->iId, PDO::PARAM_INT);
14         }
15         else // insert
16                 $oStatement = self::$oConnection->prepare('
17                         INSERT INTO `posts` (`author_id` ,`thread_id` ,`content`)
18                         VALUES (:aid, :tid, :content);');
19         $oStatement->bindValue(':aid', $this->iAuthorId, PDO::PARAM_INT);
20         $oStatement->bindValue(':tid', $this->iThreadId, PDO::PARAM_INT);
21         $oStatement->bindValue(':content', $this->sContent, PDO::PARAM_STR);
22         $oStatement->execute();
23         if($oStatement->errorCode() != 0)
24         {
25                 $aError = $oStatement->errorInfo();
26                 throw new Exception($aError[2]);
27         }
28         if($this->iId == null)
29                 $this->iId = self::$oConnection->lastInsertId();
30 }
31 //$
32 ?>
```

W linii 4 jest rozróżnienie czy wpisujemy nową wartość do bazy, czy też aktualizujemy istniejący już rekord.
W linii 6 jest przygotowywane zapytanie *UPDATE* przed wykonaniem go. Składnia zapytania jest podobna do składni zapytania *SELECT*. Po frazie *UPDATE* podawana jest nazwa tabeli którą aktualizujemy, następnie po wyrazie *SET* podawane są nowe wartości odpowiednich kolumn, rozdzielone przecinkami. Nie trzeba wymieniać wszystkich kolumn jakie są w tabeli - te niewymienione zachowają swoją niezmienioną wartość. Klauzula *WHERE* w zapytaniu *UPDATE* determinuje który rekord w bazie danych ma zostać zmodyfikowany. Jej składnia jest identyczna jak w przypadku zapytania *SELECT*. Za pomocą zapytania *UPDATE* można modyfikować jeden lub wiele rekordów w bazie danych na raz.
W linii 15 jest przygotowywane zapytanie wstawiające nową wartość do bazy danych. Po frazie *INSERT INTO* jest wymieniona nazwa tabeli, a w nawiasie oddzielone przecinkami wartości kolumn. Ważne jest, że przy zapytaniu *INSERT* należy wymienić wszystkie kolumny z tabeli, którym nadaliśmy atrybut *NOT NULL* przy projektowaniu bazy danych. Po wyrażeniu *VALUES* są podane wartości które zostaną wstawione do bazy danych. Wartości kolejno odpowiadają wcześniej wymienionym kolumnom i muszą być oddzielone przecinkami. W powyższych zapytaniach znowu użyliśmy fraz zastępczych :aid, :tid etc., które zostają potem dopiero podmienione za pomocą funkcji *bindValue()*.
W linii 22 tak jak w poprzednim przypadku jest weryfikowana poprawność zapytania.
W linii 28 jeżeli był wykonywany *INSERT* odzyskujemy wartość kolumny 'id' za pomocą metody *lastInsertId()*[9], które umożliwi nam późniejsze odwoływanie się do umieszczonego w bazie postu.

Zawartość metody **delete()**:

[9] http://www.php.net/manual/en/pdo.lastinsertid.php

```
1  <?php
2  public function delete()
3  {
4          $oStatement = self::$oConnection->prepare('
5                  DELETE FROM `posts` WHERE `id` = :id');
6          $oStatement->bindValue(':id', $this->iId, PDO::PARAM_INT);
7          $oStatement->execute();
8          if($oStatement->errorCode() != 0)
9          {
10                 $aError = $oStatement->errorInfo();
11                 throw new Exception($aError[2]);
12         }
13         $this->iId = null;
14 }
15 ?>
```

W linii 5 jest przygotowywane zapytanie usuwające post z bazy danych. Po wyrażeniu *DELETE FROM* podana jest nazwa tabeli z której usuwamy rekordy, a klauzula WHERE determinuje który rekord ma zostać usunięty.

4.3.2 dao/Thread.php