Krótki kurs tworzenia stron w technologii PHP

Mateusz Gałażyn, Jeremi Niedziela

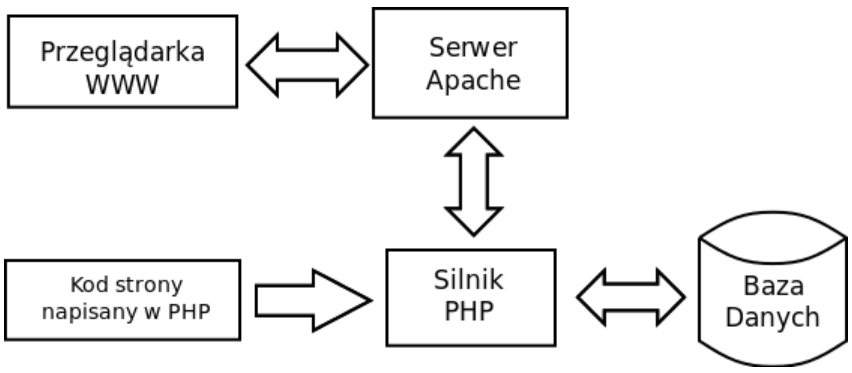
9 maja 2012

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

Aby móc uruchomić stronę opartą o Apache + PHP + MySQL konieczna jest instalacja tych 3 składników oprogramowania.

2.1 Instalacja na systemie Windows

Cały pakiet AMP zainstalujemy w katalogu *C:\etc*. Poszczególne składniki będą umieszczone w katalogach:
C:\etc\apache - serwer apache
C:\etc\php - php
C:\etc\mysql - serwer mysql
C:\etc\www - katalog główny, w którym zostanie umieszczone forum

1. Upewniamy się że mamy zainstalowany "Microsoft Visual C++ 2010 SP1 Redistributable Package (x86)". Jeżeli nie wiemy czy mamy, lub nie mamy, należy zainstalować ze strony: http://www.microsoft.com/download/en/details.aspx?id=8328

2. Wchodzimy na http://www.apachelounge.com/download/. I ściągamy z sekcji "Apache 2.4 win32 binary" pierwszą paczkę z góry. Rozpakowujemy ją do katalogu **C:\etc\apache**.

3. Ściągamy najnowszą wersję PHP ze strony: http://windows.php.net/download. Wersję "VC9 x86 Thread Safe" - plik Zip. Rozpakowujemy ją do katalogu **C:\etc\php**.

4. Ściągamy bibliotekę odpowiedzialną za komunikację PHP i apache ze strony: http://www.apachelounge.com/download/ z sekcji "Apache 2.4 win32 modules", wybieramy plik php5apache2_4.dll-php-5.4-win32.zip i wypakowujemy z archiwum z podkatalogu o największym numerze wersji plik *php5apache2_4.dll* do katalogu **C:\etc\php**.

5. Ściągamy najnowszą wersję MySQL ze strony: http://dev.mysql.com/downloads/mysql/#downloads. Wybieramy platformę "Microsoft Windows" a następnie wersję odpowiednią do naszej architektury, "ZIP Archive". Rozpakowujemy ją do katalogu **C:\etc\mysql**.

2.2 Instalacja na systemie Linux

Instalacja w systemie Linux jest nieco prostsza. Zostanie do pokazane na przykładzie systemu Ubuntu.

1. W terminalu wydajemy polecenie:
```
sudo apt-get install apache2 libapache2-mod-php5 mysql-server \
libapache2-mod-auth-mysql php5-mysql php5-mcrypt
```

2. W trakcie instalacji musimy podać hasło roota do bazy danych. To hasło będzie później używane do nawiązywania połączenia z bazą danych.

3. Serwer Apache2 + php + mysql jest już zainstalowany, skonfigurowany z ustawieniami domyślnymi i włączony, co możemy zweryfikować wchodząc pod adres http://127.0.0.1/.

2.3 Konfiguracja

Konfiguracja ma na celu umożliwienie komunikacji pomiędzy poszczególnymi komponentami pakietu AMP. W systemie Ubuntu po wykonaniu komendy w poprzednim kroku Apache2 jest już w pełni skonfigurowane, można więc opuścić podpunkt o jego konfiguracji.

2.3.1 Konfiguracja Apache

1. Otwieramy edytorem tekstowym plik **C:\etc\apache\conf\httpd.conf** - pod sytemem Windows, **/etc/apache2/apache2.conf** - pod systemem Linux (wymagane uprawnienia roota). Następnie ustawiamy wartości:

2. **ServerRoot "C:/etc/apache"** - miejsce gdzie jest konfiguracja apache, pod systemem linux ścieżką będzie **/etc/apache2**
   Jeżeli nie ma poniższych linijek w konfiguracji, to dopisujemy pod systemem Windows:
   LoadModule php5_module "C:/etc/php/php5apache2_4.dll" - ścieżka do wypakowanego wcześniej pliku php5apache2_4.dll
   AddHandler application/x-httpd-php .php
   PHPIniDir "C:/etc/php" - ścieżka w której zostało zainstalowane php

   Pod sytemem linux, w katalogu **/etc/apache2/conf.d/mods-enabled/** jest plik **php5.load**, upewnijmy się że jego zawartość wygląda następująco:
   LoadModule php5_module /usr/lib/apache2/modules/libphp5.so

3. (Linux) Następnie w pliku **/etc/apache2/mods-enabled/php5.conf** powinien znajdować się fragment odpowiedzialny za ładowanie domyślnie pliku index.php po wejściu na po adres strony przez przeglądarkę:
   <FilesMatch "\.ph(p3?|tml)$">
   SetHandler application/x-httpd-php
   </FilesMatch>

4. Pod sytemem Windows w pliku **C:\etc\apache\conf\httpd.conf** Mamy jeszcze możliwość ustawienia adresu e-mail:
   ServerAdmin admin@twojadomena.com
   Oraz katalogu głównego gdzie będą przechowywane strony WWW:
   DocumentRoot "C:/etc/www"
   <Directory "C:/etc/www">
   Dodatkowo ustawiamy opcje wewnątrz *<Directory "C:/etc/www">*:
   AllowOverride All
   Następnie podmieniamy odpowiednie ścieżki w poniższych opcjach:
   ScriptAlias /cgi-bin/ "C:/etc/apache/cgi-bin/"
   <Directory "C:/etc/apache/cgi-bin">
   I dopisujemy index.php w DirectoryIndex:
   DirectoryIndex index.html index.php

   W systemie linux, te opcje znajdują się w pliku: **/etc/apache2/sites-enabled/000-enabled**. W pliku 000-default można też zdefiniować strony dostępne pod innymi adresami umieszczone w innych katalogach niż */var/www*, wystarczy dodać poniższe dyrektywy:
   <VirtualHost *:80>
   ServerAlias nasz.alias
   ServerName nasz.alias
   VirtualDocumentRoot /nasz/katalog/ze/strona
   <Directory /nasz/katalog/ze/strona>
   Options Indexes FollowSymLinks MultiViews
   AllowOverride All
   Order allow,deny
   allow from all
   </Directory>
   </VirtualHost>
   a następnie do pliku /etc/hosts poniższą linijkę:
   127.0.0.1 nasz.alias
   Po zrestartowaniu serwera strona pod nową domeną powinna być już widoczna.

5. Po skonfigurowaniu serwera apache pod systemem windows, należy otworzyć wiersz poleceń z uprawnieniami administratora (menu start, wpisujemy cmd.exe, PPM, uruchom jako administrator), a następnie przejśc do katalogu C:/etc/apache/bin za pomocą polecenia:
   cd C:\etc\apache\bin
   i wydać komendę:
   httpd.exe -k install
   Zainstaluje ona usługę serwera WWW. Aby uruchomić serwer Apache należy uruchomić aplikację

C:\etc\apache\bin\ApacheMonitor.exe - w zasobniku systemowym pojawiła się ikona, klikając na nią otworzy się menu pozwalające uruchamiać i zatrzymywać serwer Apache.
Pod systemem linux konieczne jest wywołanie komendy:

```
sudo service apache2 start
```

Aby zatrzymać serwer należy wywołać komendę:

```
sudo service apache2 stop
```

2.3.2 Konfiguracja PHP

Konfiguracja PHP pod systemem Windows znajduje się w pliku **C:\etc\php\php.ini**. Jeżeli tego pliku tam nie ma, należy zmienić nazwę pliku php.ini-production na php.ini. Pod systemem linux konfiguracja znajduje się w katalogu **/etc/php5/apache2/php.ini**. Należy ustawić następujące parametry:
error_reporting = E_ALL
display_errors = On
html_errors = On
extension_dir = "/usr/lib/php/modules/" - dla systemu windows ścieżką będzie: **C:\etc\php\ext**
date.timezone = Europe/Warsaw
odblokować następujące rozszerzenia (usunąć średniki z początku linijek):
extension=mysqli.so
extension=mbstring.so
extension=mcrypt.so
extension=pdo_mysql.so
W systemie Windows w nazwach powyższych plików zamiast rozszerzeń .so będą .dll.

2.3.3 Konfiguracja MySQL

Po rozpakowaniu serwera MySQL w systemie Windows musimy zmienić tylko nazwę pliku my-small.ini na my.ini w katalogu **C:\etc\mysql**. W systemie linux zostało to zrobione automatycznie przy instalacji. Serwer MySQL w systemie Windows uruchamia sie za pomocą aplikacji: **C:\etc\mysql\bin\mysqld.exe**. Migający kursor i brak komunikatów o błędzie sygnalizuje poprawne uruchomienie serwera. Pod systemem linux konieczne jest wywołanie komendy:

```
sudo service mysqld start
```

Aby zatrzymać serwer należy wywołać komendę:

```
sudo service mysqld stop
```

2.4 Instalacja phpMyAdmin

phpMyAdmin jest bardzo przydatnym narzędziem pozwalającym na konfigurację i zarządzanie bazą danych z poziomu przeglądarki.

1. Ściągamy paczkę ze strony http://www.phpmyadmin.net/home_page/downloads.php i wypakowujemy do katalogu **/var/www/pma** w systemie linux lub **C:\etc\www\pma** w systemie windows.
2. Zmieniamy w katalogu do którego wypakowaliśmy phpMyAdmin nazwę pliku config.sample.inc.php na config.inc.php, a następnie otwieramy ten plik edytorem tekstowym i modyfikujemy linię:

```
$cfg['Servers'][$i]['AllowNoPassword'] = true;
```

Po uruchomieniu serwera www i MySQL wchodzimy na adres: http://127.0.0.1/pma/ i wpisujemy login "root" i hasło jakie zostało podane przy instalacji serwera (w przypadku gdy nie było podawane żadne hasło, należy zostawić je puste). Po zalogowaniu się po lewej mamy spis aktualnie istniejących baz danych. Jeżeli hasło roota nie zostało ustawione, należy wykonać poniższą procedurę:

1. Klikamy na zakładkę "Użytkownicy" u góry, na środku prawej części okna.
2. Klikamy "Edytuj uprawnienia" przy pierwszym z góry użytkowniku "root".
3. Przewijamy do sekcji "Zmień hasło", wpisujemy nowe hasło użytkownika, zatwierdzamy.
4. Powtarzamy operację dla pozostałych użytkowników o nazwie "root" w bazy danych.

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.

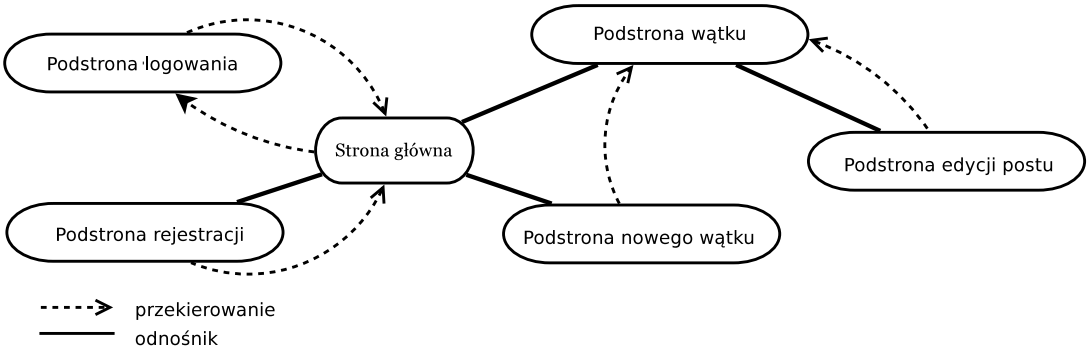
Podstrona wątku W tym miejscu trzeba wyświetlić wszystkie posty w danym wątku, oraz formularz dający możliwość odpowiedzi w tym wątku. Gdy administrator wejdzie na tą podstronę, trzeba dać także dodatkowe możliwości edycji i usuwania poszczególnych postów.

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

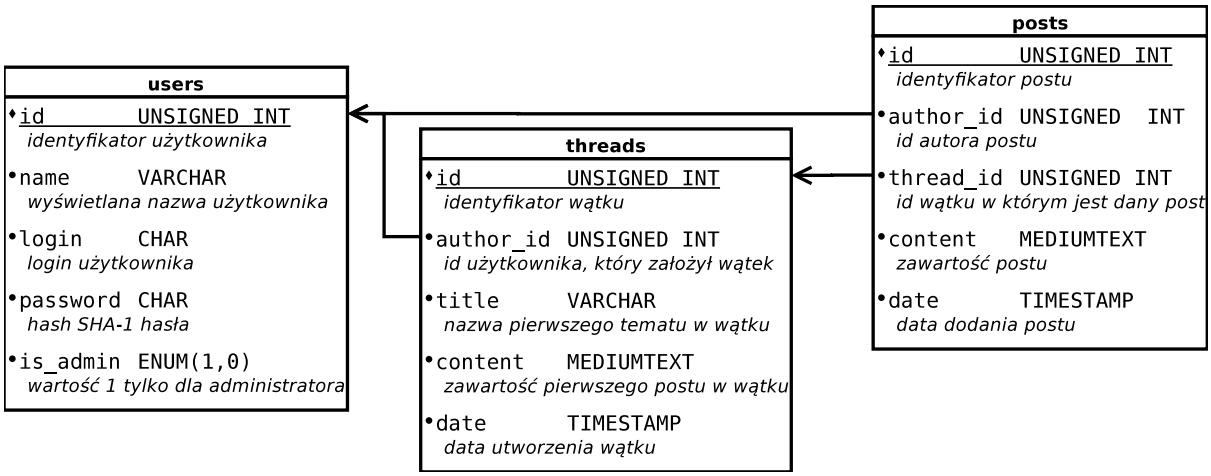
TIME Czas w formacie: HH:MM:SS

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.

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:

[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

```
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*:

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

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:

```
1  <?php
2  $jeden = 'bar';
3  $dwa = "foo";
4  $trzy = null;
5  $liczba = "4";
6  $liczba2 = 3;
7  if($liczba % 2 == 0)
8  {
9          echo 'liczba jest podzielna przez 2 <br />';
10 }
11 else
12 {
13         echo 'liczba nie jest podzielna przez 2 <br />';
14 }
15
16 if($liczba == 4)
17         echo 'Ta liczba to 4!';
18 elseif($liczba < 4)
19         echo 'Ta liczba jest mniejsza od 4.<br />';
20 else
21         echo 'Ta liczba jest wieksza od 4 <br />';
22 echo $jeden.$dwa.($liczba+$liczba2);
23 ?>
```

Zapisując powyższy kod jako plik *index.php* w katalogu /var/www (lub c:/etc/www/ na systemie Windows), 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:

```php
<?php
class Telewizor
{
        private static $iloscTelewizorow = 0;

        private $aktualnyKanal = 0;
        private $poziomGlosnosci = 50;

        public static function pobierzLiczbeTelewizorow()
        {
                return self::$iloscTelewizorow;
        }

        public function __construct()
        {
                ++self::$iloscTelewizorow;
        }

        public function ustawGlosnosc($poziomGlosnosci)
        {
                $this->poziomGlosnosci = $poziomGlosnosci;
        }

        public function ustawKanal($nrKanalu = 0)
        {
                $this->aktualnyKanal = $nrKanalu;
        }
};

function wyswietlaj()
{
        echo 'teraz wyswietlaja '.Telewizor::pobierzLiczbeTelewizorow().' telewizory';
}

new Telewizor;
new Telewizor;
$sharp =  new Telewizor;
$sharp->ustawKanal();
$sharp->ustawKanal(13);
$sharp->ustawGlosnosc(30);
$sharp->ustawGlosnosc(10);

wyswietlaj();
?>
```

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 ele-

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

ment 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. Wszystkie pliki muszą zostać umieszczone w folderze /var/www (lub C:/etc/www na systemie Windows).

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.

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

```php
<?php
class Post
{
        static public $oConnection = null;

        public $iId = null;
        public $iAuthorId = null;
        public $sAuthorName = null;
        public $iThreadId = null;
        public $sContent = null;
        public $sDate = null;

        public function __construct($iId = null){}

        public function save(){}

        public function delete(){}
};
?>
```

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):

```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]);
                }
```

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

```
15                  if($aResult = $oStatement->fetch(PDO::FETCH_ASSOC))
16                  {
17                          $this->iId = $aResult['pid'];
18                          $this->iAuthorId = $aResult['author_id'];
19                          $this->sAuthorName = $aResult['name'];
20                          $this->iThreadId = $aResult['thread_id'];
21                          $this->sContent = $aResult['content'];
22                          $this->sDate = $aResult['date'];
23                  }
24                  else
25                          $this->iId = null;
26          }
27  }
28  ?>
```

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 kolumnom 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

[4] http://www.php.net/manual/en/pdo.prepare.php
[5] http://www.php.net/manual/en/pdostatement.bindvalue.php

(a poprzez celową manipulację możliwe by było nawet skasowanie zawartości bazy[6]). Drugi argument metody *bindValue()* mówi jakiego typu zmienna jest przekazywana. 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()**:

```
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 ?>
```

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

[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

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()**:

```
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.

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

4.3.2 dao/Thread.php

Klasa Thread pozwalająca na manipulację wątkami:

```
1  <?php
2  class Thread
3  {
4          static public $oConnection = null;
5
6          public $iId = null;
7          public $iAuthorId = null;
8          public $sAuthorName = null;
9          public $sTitle = null;
10         public $sContent = null;
11         public $sDate = null;
12         public $aPosts = array();
13
14         static public function getAll() {}
15
16         public function __construct($iId = null){}
17
18         public function getAllPosts(){}
19
20         public function getLastAnswer(){}
21
22         public function save(){}
23
24         public function delete(){}
25 };
26 ?>
```

W linii 4 jest zadeklarowane statyczne pole przechowujące obiekt reprezentujący połączenie z bazą danych. W liniach 6-12 są zadeklarowane właściwości klasy odpowiadające poszczególnym kolumnom w bazie danych. W linii 12 została zadeklarowana zmienna *$aPosts*, która będzie przechowywać wszystkie odpowiedzi w temacie reprezentowanym przez obiekt typu Thread.
Klasa zawiera statyczną metodę *getAll()*, która pozwala na pobranie wszystkich wątków z bazy danych, metodę *getAllPosts()*, która zwraca wszystkie odpowiedzi w temacie, metodę *getLastAnswer()*, która pobiera ostatnią odpowiedź oraz metody *save()* i *delete()* pozwalające na zapisanie do bazy oraz na usunięcie wątku. Konstruktor podobnie jak w klasie Post, przyjmuje id poszukiwanego wątku, jeżeli nie zostanie nic podane przy wywołaniu konstruktora, zostanie utworzony nowy wątek.
Zawartość metody **getAll()**:

```
1  <?php
2  static public function getAll()
3  {
4          $aThreads = array();
5          $oStatement = self::$oConnection->prepare('
6                          SELECT *,`threads`.`id` AS `tid`
7                          FROM `threads`,`users`
8                          WHERE `threads`.`author_id`=`users`.`id`
9                          ORDER BY `threads`.`id` DESC');
10         $oStatement->execute();
11         if($oStatement->errorCode() != 0)
12         {
13                 $aError = $oStatement->errorInfo();
14                 throw new Exception($aError[2]);
15         }
```

```
16          while($aResult = $oStatement->fetch(PDO::FETCH_ASSOC))
17          {
18                  $oThread = new self();
19                  $oThread->iId = $aResult['tid'];
20                  $oThread->iAuthorId = $aResult['author_id'];
21                  $oThread->sAuthorName = $aResult['name'];
22                  $oThread->sTitle = $aResult['title'];
23                  $oThread->sContent = $aResult['content'];
24                  $oThread->sDate = $aResult['date'];
25                  $aThreads[] = $oThread;
26          }
27          return $aThreads;
28  }
29  ?>
```

Budowa jest prawie identyczna jak jak w przypadku konstruktora klasy Post. W linii 4 deklarujemy zmienną *$aThreads*, która będzie przechowywała tablicę wszystkich wątków na forum. W następnych liniach przygotowujemy zapytanie *SELECT*, pobierające listę wątków wraz z nazwami autorów z tablicy 'users', zostaje ono wykonane i jeżeli nie ma żadnego błędu to wchodzimy do pętli while. W warunku pętli while jest pobierany w kolejnych iteracjach każdy z rekordów zwróconych od serwera MySQL. Dla każdego rekordu jest tworzony nowy obiekt typu Thread w linii 18. Następnie w liniach 18-24 są przypisywane do kolejnych pól poszczególne wartości zwrócone w zapytaniu. W linii 25 za pomocą operatora [] jest dopisywany na koniec tablicy *$aThreads* obiekt *$oThread*. Na koniec zwracana jest tablica *$aThreads*.

4.3.3 dao/User.php

Klasa User jest odpowiedzialna za identyfikację i zarządzanie użytkownikami.

```
1   <?php
2   class User
3   {
4           public static $oConnection = null;
5           public static $oCurrentUser = null;
6
7           public $iId = null;
8           public $sName = null;
9           public $sLogin = null;
10          public $sPassword = null;
11          public $bIsAdmin = false;
12
13          public static function logIn($sLogin,$sPassword){}
14
15          public static function checkUserName($sName, $sLogin){}
16
17          public static function findById($iId){}
18
19          public function save(){}
20
21  };
22  ?>
```

Podobnie jak inne klasy DAO, klasa User posiada statyczne pole *$oConnection*. Dodatkowo jest zdefiniowane statyczne pole *$oCurrentUser*, które reprezentuje aktualnego użytkownika, który znajduje się na stronie. Właściwości zdefiniowane w liniach 7 - 11 odpowiadają kolumnom w tabeli 'users'. Klasa udostępnia 3 statyczne metody:
logIn() - służąca do wyszukania użytkownika, jeżeli mamy dany jego login i hasło, zwraca obiekt klasy User jeżeli znaleziono, lub wartość *false* w przeciwnym wypadku.
checkUserName() - wyszukuje w bazie czy istnieje już taki login lub nazwa użytkownika jaką podano w argumentach funkcji. W przypadku odnalezienia zwraca *false*, jeżeli nazwa użytkownika lub login są wolne to zwraca *true*.

findById() - wyszukuje użytkownika o numerze id podanym w argumencie funkcji i zwraca obiekt typu User, lub false w wypadku gdy nie znajdzie takiego użytkownika.
save() - zapisuje użytkownika do bazy danych.

Zawartośc metody **logIn()**:

```php
<?php
public static function logIn($sLogin,$sPassword)
{
        $oStatement = self::$oConnection->prepare('
                        SELECT * FROM `users`
                        WHERE `login` = :login
                        AND `password` = :password');
        $oStatement->bindValue(':login', $sLogin, PDO::PARAM_STR);
        $oStatement->bindValue(':password', sha1($sPassword), PDO::PARAM_STR);
        $oStatement->execute();
        if($oStatement->errorCode() != 0)
        {
                $aError = $oStatement->errorInfo();
                throw new Exception($aError[2]);
        }
        if($aResult = $oStatement->fetch(PDO::FETCH_ASSOC))
        {
                $oNewUser = new User();
                $oNewUser->iId = $aResult['id'];
                $oNewUser->sName = $aResult['name'];
                $oNewUser->sLogin = $aResult['login'];
                $oNewUser->sPassword = $aResult['password'];
                $oNewUser->bIsAdmin = (bool) $aResult['is_admin'];
                return $oNewUser;
        }
        return false;
}

?>
```

Składnia tej funkcji jest analogiczna do konstruktora klasy Post. Jedyna różnica jest w linii 9, gdzie przekazujemy do bazy hash wygenerwany funkcją *sha1()*[10] na podstawie hasła.
Zawartość metody **checkUserName()**:

```php
<?php
public function save()
{
        if($this->iId != null) // update
        {
                $oStatement = self::$oConnection->prepare('
                        UPDATE `users`
                        SET `name` = :name,
                                `login` = :login,
                                `password` = :password,
                                `is_admin` = :is_admin
                                WHERE `id` =:id ');
                $oStatement->bindValue(':id', $this->iId, PDO::PARAM_INT);
        }
        else // insert
                $oStatement = self::$oConnection->prepare('
                        INSERT INTO `users` (`name` ,`login` ,`password`, `is_admin`)
                        VALUES (:name, :login, :password, :is_admin);');
```

[10] http://www.php.net/manual/en/function.sha1.php

```
19          $oStatement->bindValue(':name', $this->sName, PDO::PARAM_INT);
20          $oStatement->bindValue(':login', $this->sLogin, PDO::PARAM_INT);
21          $oStatement->bindValue(':password', $this->sPassword, PDO::PARAM_STR);
22          $oStatement->bindValue(':is_admin', $this->bIsAdmin, PDO::PARAM_BOOL);
23          $oStatement->execute();
24          if($oStatement->errorCode() != 0)
25          {
26                  $aError = $oStatement->errorInfo();
27                  throw new Exception($aError[2]);
28          }
29          if($this->iId == null)
30                  $this->iId = self::$oConnection->lastInsertId();
31  }
32  ?>
```

Budowa jest bardzo podobna do metody Post::save().

4.3.4 init.php

Plik init.php jest odpowiedzialny za załadowanie niezbędnych klas, za nawiązanie połączenia z bazą danych oraz za ładowanie obiektu użytkownika z bazy danych. Jest on dołączany do każdej na każdej podstronie forum. Zawartość:

```
1   <?php
2   function init()
3   {
4           include 'config.php';
5           include 'dao/User.php';
6           include 'dao/Post.php';
7           include 'dao/Thread.php';
8           try
9           {
10                  session_start();
11                  $oDb = new PDO($sConnectionString, $sUser, $sPassword);
12                  User::$oConnection = &$oDb;
13                  Thread::$oConnection = &$oDb;
14                  Post::$oConnection = &$oDb;
15
16                  $oUser = null;
17                  if(isset($_SESSION['uid']))
18                  {
19                          User::$oCurrentUser = User::findById($_SESSION['uid']);
20                          if(!is_object(User::$oCurrentUser))
21                                  session_destroy();
22                  }
23
24          }
25          catch(Exception $e)
26          {
27                  die('Wystapil blad polaczenia z baza danych!<br/>'.$e->getMessage());
28          }
29  }
30  ?>
```

W linii 2 została zadeklarowana funkcja init(), która jest potem wywoływana na każdej podstronie forum. Zapobiega to możliwości bezpośredniego odwołania się do pliku init.php ze strony przeglądarki internetowej.
W liniach 4 - 7 zostały załączone pliki z katalogu dao, oraz plik config.php, który zawiera ustawienia konieczne do nawiązania połączenia z bazą danych. W liniach 9 - 29 znajduje się blok trycatch odpowiedzialny za wyłapanie wyjątku rzucanego przy pojawieniu się błędu komunikacji z bazą danych.

W linii 10 jest uruchamiany mechanizm sesji za pomocą funkcji sesion_start()[11]. Sesja reprezentuje pojedyncze połączenie przeglądarki internetowej z serwerem i pozwala przechowywać dane pomiędzy podstronami w specjalnej tablicy $_SESSION. Gdy zapiszemy dane w tej tablicy na jednej z podstron, na innych mamy również do nich dostęp. Ważnym aspektem jest, że tablica *$_SESSION* jest unikalna dla każdego odwiedzającego stronę.
W linii 11 nawiązywane jest połączenie z bazą danych za pomocą konstruktora klasy PDO[12]. Jako parametry zostały przekazane: ciąg połączenia w którym jest zdefiniowana nazwa bazy danych, rodzaj bazy oraz host na którym pracuje serwer MySQL; login oraz hasło do bazy danych.
W liniach 12-14 jest przekazywana referencja obiektu *$oDb* poszczególnym klasom w folderze DAO. Aby wykonać jakąkolwiek operację na bazie danych trzeba mieć dostęp do obiektu *$oDb*.
W linii 17 jest sprawdzane czy jest ustawiona zmienna sesyjna *uid* zawierająca identyfikator użytkownika aktualnie odwiedzającego stronę. Jeżeli jest ustawiona, to następnie za pomocą metody *User::findById()* użytkownik jest ładowany z bazy danych do statycznego pola klasy *User*. Jeżeli *uid* z jakiegoś powodu nie było poprawne, jest wywoływana funkcja session_destroy()[13], która czyści zawartość tablicy *$_SESSION* i jednocześnie niszczy sesję.

4.3.5 config.php

```
1  <?php
2  $sConnectionString = 'mysql:dbname=forum;host=localhost';
3  $sUser = 'root';
4  $sPassword = 'haslo jakie ustawilismy przy konfiguracji mysql';
5  ?>
```

Plik ten zawiera ustawienia połączenia z bazą danych. Należy w nim zmienić hasło na takie jakie zostało ustawione przy konfiguracji serwera MySQL.

4.3.6 index.php

Strona główna naszego forum. Wyświetla wszystkie tematy na forum.

```
1  <?php
2  include 'init.php';
3  try
4  {
5          init();
6          $aThreads = Thread::getAll();
7  }
8  catch(Exception $e)
9  {
10         die('Wystapil blad!<br />'.$e->getMessage());
11 }
12 ?><!DOCTYPE HTML>
13 <html>
14 <head>
15  <title>Forum</title>
16  <meta charset="UTF-8" />
17 </head>
18 <body>
19 <?php if(!is_object(User::$oCurrentUser)):?>
20 <form action="login.php" method="post">
21  Login:<input type="text" name="login" value=""  required="required"/>
22  Haslo:<input type="password" name="password" value=""  required="required"/>
23  <input type="submit" value="Zaloguj sie" />  | <a href="register.php">rejestracja</a>
24 </form>
25 <?php else: ?>
```

[11] http://ro.php.net/manual/pl/function.session-start.php
[12] http://www.php.net/manual/pl/pdo.construct.php
[13] http://ro.php.net/manual/pl/function.session-destroy.php

```
26 Witaj <b><?php echo User::$oCurrentUser->sName; ?></b> ! |
27  <a href="login.php?logout">Wyloguj sie</a>
28 <?php endif; ?>
29 <br /><br />
30 <a href="add.php">Stworz nowy watek</a> <br />
31 <?php if(count($aThreads) > 0):?>
32 Lista watkow na forum:
33 <table border="1">
34 <tr>
35  <th>Lp</th>
36  <th>Temat</th>
37  <th>Autor</th>
38  <th>Data</th>
39 </tr>
40 <?php foreach($aThreads as $i => $oThread): ?>
41 <tr>
42  <td><?php echo $i+1 ?></td>
43  <td><a href="thread.php?id=<?php echo $oThread->iId ?>">
44 <?php echo $oThread->sTitle ?></a>
45 <?php
46  if(is_object(User::$oCurrentUser))
47          if(User::$oCurrentUser->bIsAdmin)
48                  echo ' <a href="delete.php?tid='.$oThread->iId.'">[ usun ]</a>';
49 ?></td>
50  <td><?php echo $oThread->sAuthorName ?></td>
51  <td><?php echo $oThread->sDate ?></td>
52 </tr>
53 <?php endforeach?>
54 </table>
55 <?php else: ?>
56 Brak watkow na forum.
57 <?php endif;?>
58 </body>
59 </html>
```

W liniach 5 i 6 jest uruchamiana funkcja init() z pliku init.php oraz ładowana jest lista wątków do tablicy *$aThreads*.

W linii 19 jeżeli użytkownik nie jest zalogowany (co oznacza, że User::$oCurrentUser ma wartość false) wyświetlamy formularz logowania oraz odnośnik do formularza rejestracji. W przeciwnym wypadku wyświetlany jest komunikat powitalny. Formularz rejestracji jest wyświetlany w liniach 20-24. Atrybuty action i method taga form mówią o tym, gdzie i w jaki sposób mają zostać przesłane dane do skryptu przetwarzającego. Metoda GET polega na przesyłaniu parametrów w adresie podstrony np.: http://127.0.0.1/index.php?parametr1=wartosc1&parametr2=wartosc3&parametr3=wartosc2 . Metoda POST pozwala przesyłać te parametry w sposób ukryty. Koniec instrukcji warunkowej jest w linii 28.

W linii 31 jeżeli tablica *$aThreads* jest niepusta jest wyświetlana lista wątków (linie 40-53) w pętli foreach. W przeciwnym razie jest wyświetlana informacja o braku wątków na forum.

4.3.7 add.php

Formularz dodawania nowego wątku.

```php
<?php
include 'init.php';
try
{
	init();
	$sError = '';
	if(is_object(User::$oCurrentUser))
	{
		if(!empty($_POST))
		{
			if(strlen(trim($_POST['title'])) < 3
				|| strlen(trim($_POST['title'])) > 200 )
				$sError .=
				'Temat watku musi miec od 3 do 200 znakow<br />';
			if(strlen(trim($_POST['content'])) < 3
				|| strlen(trim($_POST['content'])) > 10000)
				$sError .=
				'Tresc watku musi miec conajmniej 3 znaki. <br />';
			if(strlen($sError) == 0)
			{
				$oThread = new Thread;
				$oThread->sTitle =
					str_replace("\n",'<br />',
					htmlspecialchars(trim($_POST['title'])));
				$oThread->sContent =
					str_replace("\n",'<br />',
					htmlspecialchars(trim($_POST['content'])));
				$oThread->iAuthorId = User::$oCurrentUser->iId;
				$oThread->save();
				header('Location: thread.php?id='.$oThread->iId);
			}
		}
	}
}
catch(Exception $e)
{
	die('Wystapil blad!<br />'.$e->getMessage());
}
?><!DOCTYPE HTML>
<html>
<head>
 <title>Forum - dodawanie nowego watku</title>
 <meta charset="UTF-8" />
</head>
<body>
<a href="/">&lt; &lt; Powrot</a> <br />
<?php if(is_object(User::$oCurrentUser)):?>
<form action="add.php" method="post">
Temat: <br /><input type="text" name="title" value="" required="required"/> <br />
Tresc: <br /><textarea name="content" rows="10" cols="50" required="required"></textarea>
<br /><b><?php echo $sError ?></b>
<input type="submit" value="Stworz nowy watek" />
</form>
```

```
54 <?php else: ?>
55 Musisz sie zalogowac aby miec dostep do tej strony!
56 <?php endif; ?>
57 </body>
58 </html>
```

W linii 7 jest weryfikowane czy użytkownik jest zalogowany (niezalogowany użytkownik nie ma możliwości tworzenia tematów, ani odpowiadania na nie). W linii 9 jest sprawdzane, czy formularz tworzenia nowego wątku został wysłany. Po wysłaniu formularza za pomocą metody POST[14] w tablicy *$_POST* są przechowywane wartości wpisane do formularz pod indeksami, takimi jakie były wartości atrybutu *name=""* poszczególnych pól w formularzu.
W linii 11 i 12 jest sprawdzana długość tytułu postu, czy aby na pewno mieści się w zakresie 3-200 znaków. Funkcja trim()/footnotehttp://www.php.net/manual/pl/function.trim.php usuwa z początku i końca ciągu podanego jako argument białe znaki. Jeżeli tytuł nie spełnia powyższych warunków do zmiennej *$sError* jest doklejana informacja o błędzie. Identyczne sprawdzenie jest w linii 15 i 16 dla treści wątku.
W linii 19 jeżeli zmienna *$sError* jest pusta (tzn. tytuł i zawartość posta jest poprawna) jest on dodawany do bazy danych. Funkcja htmlspecialchars()[15] jest odpowiedzialna za zamianę specjalnych znaków na notację htmlową. Funkcja str_replace()[16] zamienia pierwszy parametr na drugi w ciągu znaków podanym jako trzeci parametr.
Funkcja header()[17] przekazuje nagłówek HTTP do przeglądarki, w tym miejscu została użyta do przekazania nagłówka Location, który informuje gdzie przeglądarka ma przekierować swoje żądanie. Na tym miejscu ładowanie strony zostaje zatrzymane.

4.3.8 delete.php

Plik pozwalający usuwać istniejące już tematy i posty.

```
1  <?php
2  include 'init.php';
3  try
4  {
5          init();
6          if(is_object(User::$oCurrentUser))
7          {
8                  if(isset($_GET['tid']))
9                  {
10                         $oThread = new Thread($_GET['tid']);
11                         if($iThreadId = $oThread->iId)
12                                 if(User::$oCurrentUser->bIsAdmin)
13                                         $oThread->delete();
14                 }
15                 if(isset($_GET['pid']))
16                 {
17                         $oPost = new Post($_GET['pid']);
18                         if($iPostId = $oPost->iId)
19                                 if($oPost->iAuthorId == User::$oCurrentUser->iId
20                                         || User::$oCurrentUser->bIsAdmin)
21                                         $oPost->delete();
22                 }
23         }
24         header('Location: index.php');
25 }
```

[14] http://www.php.net/manual/pl/tutorial.forms.php
[15] http://no.php.net/manual/pl/function.htmlspecialchars.php
[16] http://no.php.net/str_replace
[17] http://www.php.net/manual/pl/function.header.php

```
27 catch(Exception $e)
28 {
29         die('Wystapil blad!<br />'.$e->getMessage());
30 }
31 ?>
```

Ten plik tylko odbiera przychodzące zapytanie POST oraz GET i w zależności od uprawnień użytkownika usuwa post lub nie, a następnie przekierowuje do pliku index.php. W linii 8 jest weryfikowane czy strona została wywołana z parametrem GET, tid: np.: http://127.0.0.1/delete.php?tid=15, czy też z parametrem pid (post id): np.: http://127.0.0.1/delete.php?pid=16. W przypadku tid (thread id) usuwamy wątek, wpierw sprawdzając w linii 11 czy wątek istnieje oraz w linii 12 czy aktualnie zalogowany użytkownik jest administratorem. Jeżeli tak to jest wywoływana metoda delete(), na obiekcie reprezentującym wątek. Wątek jest ładowany w linii 10 za pomocą konstruktora do którego został przekazany numer id z parametru tid. W linii 15 jest identyczna procedura usuwania postu, z drobną różnicą, w warunku funkcji if, oprócz weryfikacji uprawnień administratora jest sprawdzenie czy użytkownik który wywołał podstronę delete.php jest autorem postu. Do usunięcia postu jest konieczna prawdziwość jednego z w/w warunków. Na koniec użytkownik jest przekierowywany na stronę główną.

4.3.9 edit.php

Podstrona pozwalająca na edycję postów i tematów na forum.

```
1  <?php
2  include 'init.php';
3  try
4  {
5      init();
6      $sError = '';
7      $bIsAccessDenied = true;
8      $iThreadId = null;
9      $iPostId = null;
10     if(is_object(User::$oCurrentUser))
11     {
12         if(isset($_GET['tid']))
13         {
14             $oThread = new Thread($_GET['tid']);
15             if($iThreadId = $oThread->iId)
16                 if($oThread->iAuthorId == User::$oCurrentUser->iId
17                     || User::$oCurrentUser->bIsAdmin)
18                     $bIsAccessDenied = false;
19         }
20         if(isset($_GET['pid']))
21         {
22             $oPost = new Post($_GET['pid']);
23             if($iPostId = $oPost->iId)
24                 if($oPost->iAuthorId == User::$oCurrentUser->iId
25                     || User::$oCurrentUser->bIsAdmin)
26                 {
27                     $bIsAccessDenied = false;
28                     $iThreadId = $oPost->iThreadId;
29                 }
30         }
31
32         if(!empty($_POST) && !$bIsAccessDenied)
33         {
34             if(isset($_GET['pid']));
35             {
36                 if(strlen(trim($_POST['content'])) < 3
37                 || strlen(trim($_POST['content'])) > 10000)
```

```
38                      $sError .=
39                  'Twoja odpowiedz musi miec conajmniej 3 znaki. <br />';
40                  if(strlen($sError) == 0)
41                  {
42                      $oResponse = new Post($iPostId);
43                      $oResponse->sContent =
44                      str_replace("\n",'<br />',
45                          trim(htmlspecialchars($_POST['content'])));
46                      $oResponse->save();
47                  }
48              }
49              if(isset($_GET['tid']))
50              {
51                  if(strlen(trim($_POST['title'])) < 3
52                  || strlen(trim($_POST['title'])) > 200 )
53                      $sError .=
54                      'Temat watku musi miec od 3 do 200 znakow<br />';
55                  if(strlen(trim($_POST['content'])) < 3
56                      || strlen(trim($_POST['content'])) > 10000)
57                      $sError .= 'Tresc watku musi miec conajmniej 3 znaki. <br />';
58                  if(strlen($sError) == 0)
59                  {
60                      $oThread = new Thread($iThreadId);
61                      $oThread->sTitle =
62                          str_replace("\n",'<br />',
63                          htmlspecialchars(trim($_POST['title'])));
64                      $oThread->sContent =
65                          str_replace("\n",'<br />',
66                          htmlspecialchars(trim($_POST['content'])));
67                      $oThread->save();
68
69                  }
70              }
71              if(strlen($sError) == 0)
72                  header('Location: thread.php?id='.$iThreadId);
73          }
74      }
75  }
76  catch(Exception $e)
77  {
78      die('Wystapil blad!<br />'.$e->getMessage());
79  }
80  ?><!DOCTYPE HTML>
81  <html>
82  <head>
83   <title>Forum - edycja</title>
84   <meta charset="UTF-8" />
85  </head>
86  <body>
87  <a href="thread.php?id=<?php echo $iThreadId; ?>">&lt; &lt; Powrot</a> <br />
88  <?php if($bIsAccessDenied):?>
89  Nie masz dostepu do tej strony!
90  <?php else: ?>
91
92  <?php if(!empty($_GET['tid'])):?>
93  Edycja tematu:
```

```
94  <form action="edit.php?tid=<?php echo $iThreadId; ?>" method="post">
95  Temat: <br /><input type="text" name="title" value="<?php
96  if(empty($_POST))
97      echo $oThread->sTitle;
98  else
99      echo $_POST['title'];
100 ?>" required="required"/> <br />
101 Tresc: <br /><textarea name="content" rows="10" cols="50" required="required">
102 <?php
103 if(empty($_POST))
104     echo $oThread->sContent;
105 else
106     echo $_POST['content'];
107 ?></textarea>
108 <br /><b><?php echo $sError ?></b>
109 <input type="submit" value="Zaktualizuj" />
110 </form>
111 <?php else: ?>
112 Edycja postu:
113 <form action="edit.php?pid=<?php echo $iPostId; ?>" method="post">
114 Tresc: <br /><textarea name="content" rows="10" cols="50" required="required">
115 <?php echo $oPost->sContent; ?></textarea>
116 <br /><b><?php echo $sError ?></b>
117 <input type="submit" value="Zaktualizuj" />
118 </form>
119 <?php endif;?>
120
121 <?php endif; ?>
122 </body>
123 </html>
```

W liniach 6-9 są zadeklarowane zmienne używane do kontroli dostępu, przechowywania informacji o błędzie, id wątku oraz id postu. Jeżeli użytkownik jest zalogowany i jest autorem tematu (postu) lub jest administratorem to w linijkach 12-30 ładowany jest temat (post) do zmiennej $oThread ($oPost) oraz ustawiana zmienna $bIsAccessDenied na false (dostęp do strony jest umożliwiony). W linii 32 następuje weryfikacja, czy został wysłany formularz metodą POST oraz czy mamy dostęp do edycji tematu (postu). Jeżeli tak, to dalej dokonuje się weryfikacja identycznie jak formularzu dodawania nowego tematu (postu), z tą różnicą że ładowany jest juz temat z bazy za pomocą id podanego w parametrze tid (pid). Na koniec w linii 72 jest wykonywane przekierowanie do strony wątku.
W linii 88 jest wykonywana weryfikacja uprawnień. W przypadku ich braku jest wyświetlana stosowna informacja. Koniec if znajduje się w linii 121. W linii 92 następuje rozróżnienie na formularz edycji postu i formularz edycji tematu - ustawiony parametr tid informuje o edycji tematu. Koniec if znajduje się w linii 119.

4.3.10 login.php

Podstrona odpowiedzialna za zalogowanie użytkownika po podaniu loginu i hasła.

```
1  <?php
2  include 'init.php';
3  try
4  {
5          init();
6          if(isset($_GET['logout']))
7          {
8                  session_destroy();
9                  header('Location: index.php');
10         }
```

```
12      if(isset($_POST))
13          if($oUser = User::logIn($_POST['login'],$_POST['password']))
14          {
15              $_SESSION['uid'] = $oUser->iId;
16              header('Location: index.php');
17          }
18  }
19
20  catch(Exception $e)
21  {
22      die('Wystapil blad!<br />'.$e->getMessage());
23  }
24  ?><!DOCTYPE HTML>
25  <html>
26  <head>
27   <title>Forum</title>
28   <meta charset="UTF-8" />
29  </head>
30  <body>
31  <a href="/">&lt; &lt; Powrot</a> <br />
32  Niepoprawny login lub haslo!
33  </body>
34  </html>
```

W liniach 6-10 jest obsługiwane wylogowywanie użytkownika. Polega ono jedynie na zniszczeniu sesji i przekierowaniu spowrotem do strony głównej. W linii 12 i 13 jeżeli został wysłany formularz logowania się jest ładowany użytkownik z bazy danych, jeżeli operacja się powiedzie jest ustawiana zmienna sesyjna uid na id zalogowanego użytkownika, a następnie jest wykonywane przekierowanie na stronę główną. Gdy użytkownik nie istnieje to przekierowanie nie zostanie wykonane i zostanie wyświetlony komunikat o błędzie.

4.3.11 register.php

Podstrona umożliwia rejestrację nowego użytkownika.

```
1   <?php
2   include 'init.php';
3   try
4   {
5       init();
6       $sError = '';
7       $bRegistrationComplete = false;
8       if(!is_object(User::$oCurrentUser))
9       {
10          if(!empty($_POST))
11          {
12              if(!preg_match("#^([a-z0-9_-]{3,30})$#si",$_POST['login']))
13                  $sError .= 'Nieprawidlowy login - moze skladac sie
14                  tylko z liter, cyfr oraz znakow "_" oraz \'-\'.
15                  Login musi miec od 3 do 30 znakow.<br/>';
16              if(!preg_match(
17                  "#^([a-z0-9_-\s]{3,30})$#si",
18                  trim($_POST['username'])))
19                  $sError .= 'Nieprawidlowa nazwa uzytkownika - moze
20                  skladac sie tylko z liter, cyfr, spacji oraz znakow
21                  "_" oraz \'-\'. Nazwa uzytkownika musi miec od 3 do
22                  30 znakow.<br/>';
```

```
16              if(!preg_match(
17                  "#^([a-z0-9_-]{5,30})$#si",
18                  trim($_POST['password'])))
19                  $sError .= 'Nieprawidlowe haslo - moze skladac sie
20                  tylko z liter, cyfr oraz znakow "_" oraz \'-\'.
21                  Haslo musi miec od 5 do 30 znakow.<br/>';
22              if($_POST['password'] != $_POST['password2'])
23                  $sError .= 'Podano rozne hasla. <br/>';
24              if(strlen($sError) == 0)
25                  if(!User::checkUserName(
26                          trim($_POST['username']),
27                          trim($_POST['login'])))
28                      $sError .= 'Podana nazwa uzytkownika lub
29                      login juz istnieje. <br />';
30              if(strlen($sError) == 0)
31              {
32                  $bRegistrationComplete = true;
33                  $oUser = new User;
34                  $oUser->sName = trim($_POST['username']);
35                  $oUser->sPassword = sha1(trim($_POST['password']));
36                  $oUser->sLogin = trim($_POST['login']);
37                  $oUser->save();
38                  if($oUser->iId != null)
39                      $_SESSION['uid'] = $oUser->iId;
40              }
41          }
42      }
43  }
44
45  catch(Exception $e)
46  {
47      die('Wystapil blad!<br />'.$e->getMessage());
48  }
49  ?><!DOCTYPE HTML>
50  <html>
51  <head>
52   <title>Forum - rejestracja</title>
53   <meta charset="UTF-8" />
54  </head>
55  <body>
56  <a href="/">&lt; &lt; Powrot</a> <br />
57  <?php
58  if(!$bRegistrationComplete):
59  if(!isset($_SESSION['uid'])):
60  ?>
61  Rejestracja na forum. Aby moc pisac posty i zakladac nowe
62  tematy konieczna jest rejestracja.  <br /> <br />
63  <form action="register.php" method="post">
64   Twoj login: <br />
65  <input type="text" name="login" value="" required="required" />
66  <br />
67   Wyswietlana nazwa uzytkownika: <br />
68  <input type="text" name="username" value="" required="required" />
69    <br />
70   Haslo:  <br />
71  <input type="password" name="password" value=""  required="required" />
72   <br />
```

```
73  Powtorz haslo:  <br />
74 <input type="password" name="password2"  required="required" />
75 <br />
76 <b><?php echo $sError ?></b>
77  <input type="submit" value="Zarejestruj sie" />
78 </form>
79 <?php else: ?>
80 Aby moc zarejestrowac nowe konto wpierw sie wyloguj.
81 <?php endif;?>
82 <?php else: ?>
83 Rejestracja zakonczona. Twoje konto zostalo automatycznie zalogowane na forum.
84 <?php endif;?>
85 </body>
86 </html>
```

Po sprawdzeniu, czy użytkownik jest zalogowany i czy został wysłany formularz, w linii 12 jest sprawdzana poprawność podanego loginu w formularzu rejestracyjnym za pomocą wyrażenia regularnego[18] o składni z Perla. Użyta została do tego funkcja preg_match()[19]. Jako pierwszy argument zostało podane wyrażenie regularne w postaci ciągu znaków zamkniętego w " ", jako drugi parametr jest podany ciąg przeszukiwany pod kątem zawartości. Funkcja zwraca 1 jeżeli przeszukiwany ciąg pasuje do wzorca lub 0 w przeciwnym przypadku. Wzorzec jest zawarty pomiędzy znakami "#". Litery s oraz i oznaczają, żeby ignorować znaki nowej linii oraz wielkość znaków. Znak "^" oznacza początek ciągu znaków, dozwolone klasy znaków, które mogą się pojawić w porównywanym ciągu, są wpisane w nawiasie kwadratowym: litery a-z, cyfry 0-9, znaki " _" oraz "-". \s oznacza spację. Za nawiasem kwadratowym są liczby w klamrach informujące o maksymalnej i minimalnej długości (od 3 do 30 znaków) porównywanego ciągu. Koniec ciągu jest oznaczony "$".
W linii 16 i 23 w analogiczny sposób są sprawdzane hasło i nazwa użytkownika.
W linii 32, jeżeli nie było błędu w weryfikacji hasła, loginu i nazwy użytkownika, jest dokonywane sprawdzenie czy istnieje już taka nazwa użytkownika i login w bazie danych za pomocą funkcji User::checkUserName(). Jeżeli wszystkie weryfikacje odbyły się poprawnie to w liniach 40-44 użytkownik jest dodawany do bazy danych. Hasło przed wpisaniem do bazy danych jest szyfrowane funkcją sha1(). W linii 46, użytkownik po wpisaniu do bazy jest logowany na stronie.

4.3.12 thread.php

Podstrona wyświetlająca wątek i pozwalająca na odpowiadanie w nim.

```
1  <?php
2  include 'init.php';
3  try
4  {
5          init();
6          $oThread = null;
7          $aPosts = array();
8          $sError = '';
9          if(isset($_GET['id']))
10         {
11                 $oThread = new Thread($_GET['id']);
12                 if($oThread->iId == null)
13                         $oThread = null;
14                 else
15                 {
16                         if(!empty($_POST) && is_object(User::$oCurrentUser))
17                         {
18                                 if(strlen(trim($_POST['content'])) < 3
19                                         || strlen(trim($_POST['content'])) > 10000)
```

[18] http://pl.wikipedia.org/wiki/Wyrażenie_regularne
[19] http://www.php.net/manual/pl/function.preg-match.php

```
20                      $sError .= 'Twoja odpowiedz musi miec
21                              conajmniej 3 znaki. <br />';
22                  if(strlen($sError) == 0)
23                  {
24                      $oResponse = new Post;
25                      $oResponse->iAuthorId =
26                          User::$oCurrentUser->iId;
27                      $oResponse->iThreadId = $oThread->iId;
28                      $oResponse->sContent =
29                          str_replace(
30                          "\n",
31                          '<br />',
32                          trim(
33                          htmlspecialchars($_POST['content'])
34                          ));
35                      $oResponse->save();
36                  }
37              }
38              $aPosts = $oThread->getAllPosts();
39          }
40      }
41  }
42  catch(Exception $e)
43  {
44      die('Wystapil blad!<br />'.$e->getMessage());
45  }
46  ?><!DOCTYPE HTML>
47  <html>
48  <head>
49   <title>Forum</title>
50   <meta charset="UTF-8" />
51  </head>
52  <body>
53  <a href="/">&lt; &lt; Powrot</a> <br />
54  <?php if(is_object($oThread)): ?>
55  <h1><?php echo $oThread->sTitle; ?></h1>
56  <dl>
57  <dt>[ <?php echo $oThread->sDate ?> ] <b>
58  <?php echo $oThread->sAuthorName ?></b> napisal:
59  <?php
60  if(is_object(User::$oCurrentUser))
61  {
62      if(User::$oCurrentUser->iId == $oThread->iAuthorId
63          || User::$oCurrentUser->bIsAdmin)
64          echo ' <a href="edit.php?tid='.$oThread->iId.'">
65                  [ edytuj ]</a>';
66      if(User::$oCurrentUser->bIsAdmin)
67          echo ' <a href="delete.php?tid='.$oThread->iId.'">
68                  [ usun ]</a>';
69  }
70  ?></dt>
71  <dd><?php echo $oThread->sContent ?></dd>
72  <?php  foreach($aPosts as $oPost): ?>
73  <dt>[ <?php echo $oPost->sDate ?> ] <b>
74  <?php echo $oPost->sAuthorName ?></b> napisal:
```

```
75 <?php
76 if(is_object(User::$oCurrentUser))
77 {
78     if(User::$oCurrentUser->iId == $oPost->iAuthorId
79         || User::$oCurrentUser->bIsAdmin)
80         echo ' <a href="edit.php?pid='.$oPost->iId.'">[ edytuj ]</a>
81         <a href="delete.php?pid='.$oPost->iId.'">[ usun ]</a>';
82 }
83 ?></dt>
84 <dd><?php echo $oPost->sContent ?></dd>
85 </dl>
86 <?php endforeach;?>
87 <br />
88 <?php if(is_object(User::$oCurrentUser)):?>
89 <form action="thread.php?id=<?php echo $oThread->iId?>" method="post">
90 Odpowiedz: <br />
91 <textarea name="content" rows="10" cols="50" required="required">
92 <?php if(!empty($_POST['content']) && strlen($sError) > 0)
93     echo htmlspecialchars($_POST['content']);?></textarea>
94 <br /><b><?php echo $sError ?></b>
95 <input type="submit" value="Stworz nowy watek" />
96 </form>
97 <?php endif;?>
98 <?php else: ?>
99 Nie ma takiego watku.
100 <?php endif; ?>
101 </body>
102 </html>
```

W linii 11 jest ładowany żądany wątek, w zależności od podanego parametru id. Jeżeli nie zostanie podany poprawny parametr id, to w linii 99 jest wyświetlany komunikat o błędzie.
W linii 16 jest dokonywane sprawdzenie, czy został wysłany formularz odpowiedzi oraz czy użytkownik jest zalogowany. W dalszej części są weryfikowane dane przesłane z formularza (czyli treść odpowiedzi). Jeżeli weryfikacja przebiegła poprawnie, obiekt $oPost jest dodawany do bazy danych za pomocą metody save().
W linii 38 są pobierane wszystkie odpowiedzi bazy danych i przypisywane do zmiennej $aPosts.
W liniach 60-69 jeżeli użytkownik jest użytkownikiem lub administratorem przy temacie pojawia się możliwość edycji tematu. Dodatkowo tylko administrator ma możliwość usunięcia tematu.
W liniach 72-86 są wyświetlane wszystkie odpowiedzi. Przy każdej odpowiedzi użytkownik lub administrator ma możliwość usunięcia lub modyfikacji swojej odpowiedzi (administrator może modyfikować i usuwać każde).
W liniach 88-97 jeżeli użytkownik jest zalogowany, to jest wyświetlany formularz odpowiedzi w wątku.

5 Zakończenie

Jeżeli udało nam się bezbłędnie napisać wszystkie pliki, to powinniśmy po wejściu na stronę http://127.0.0.1/ ujrzeć stronę główną forum. Ostatnią rzeczą jest utworzenie konta administratora. Aby to zrobić należy:

1. Zarejestrować się na forum
2. Po zalogowaniu się na stronie http://127.0.0.1/pma/ klikamy po lewej stronie na nazwę bazy forum.
3. Klikamy po lewej stronie na napis users.
4. Znajdujemy swoje konto po prawej stronie. Klikamy na pole w kolumnie is_admin i zmieniamy wartość na 1.
5. Wartość została automatycznie zapisana i nasze konto ma już uprawnienia administratora.