Programowanie Aplikacji Internetowych

Laboratorium nr 2

Wzorzec MVC, baza danych MySQL

Poniżej znajdują się zadania, które należy wykonać w ramach laboratoriów, a następnie sporządzić sprawozdanie w formie archiwum .zip. Plik archiwum powinien mieć nazwę zgodną ze wzorem: PAI_Lab<nr_laboratoriu>_<pierwsza_litera_nazwiska>.<nazwisko_bez_polskich_znakow>.zip, np. PAI_Lab2_J.Kowalski.zip. W archiwum powinna znajdować się poniższa struktura katalogów i plik:

```
\projekt-01-01
    | - \projekt-01-01
    | - \lib
    | - \generowanie
    \ - \wstawianie
\<nazwa_własnego_projektu_z_zadania_7>
    | - \<nazwa_własnego_projektu_z_zadania_7>
    | - \lib
    | - \generowanie
    \ - \wstawianie
mysql.pdf
```

Zadania **muszą** być wykonywane w zadanej kolejności – od 1 do 9.

Zadanie 1 Pierwszy projekt

Rozpakowujemy do */var/www zawartość* `https://db.tt/g9AVG7ev`

W folderze tym należy umieścić plik *projekty-01-01/scripts/modules/main/actions.class.php* zawierający definicję klasy *Actions*, która dziedziczy po klasie *ActionsBase*. Nowo zdefiniowana klasa może być początkowo pusta:

```
<?php
class Actions extends ActionsBase
{
}
// brak znacznika ?> zapobiega problemów związanych z białymi znakami po
// znaczniku zamykającym.
```

Ustalamy nazwę akcji: *hello*

Z tego względu dopisujemy w klasie Actions funkcję *execute_hello()* :

```
public function execute_hello()
{
}
```

Funkcja będzie uruchamiana po wystąpieniu akcji *main/hello*.

Tworzymy plik *projekt-01-01/scripts/modules/main/hello.html*, który będzie widokiem naszej akcji:

```
<h1>Hello world!</h1>
```

Teraz tworzymy układ witryny. W tym celu tworzymy plik *projekty-01-01/scripts/templates/layout.html*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">
  <head>
    <title>Projekt 1.1</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
<body>

<?php include "../scripts/modules/$module/$action.html"; ?>

</body>
</html>
```

Teraz utworzymy translację adresu aby po odwiedzeniu adresu URL pierwszy-projekt.html wykonana została akcja *main/hello*. Regułę tą zapiszemy w pliku *project-01-01/scripts/translations.txt*:

```
/pierwszy-projekt.html  index.php?module=main&action=hello
/                   index.php?module=main&action=hello
```

Wreszcie tworzymy skrypt *projekt-01-01/www/index.php*:

```
<?php
error_reporting(E_ALL);              //
ini_set('display_errors', 1);        // komunikaty diagnostyczne -bez względu
                                  // na konfigurację będą wyświetlane błędy
set_include_path(                    // zmieniamy ścieżki poszukiwać
    '../scripts' . PATH_SEPARATOR .  // - zmniejszamy ilość niezbędnych
    '../scripts/include' . PATH_SEPARATOR .       //  instrukcji require_once.
    get_include_path()
);

require_once 'Controller.class.php';

$controller = new Controller();
$controller->dispatch();
```

Plik *project-01-01/www/.htaccess ma* zawierać reguły, które przekierowują wszystkie żądania HTTP do skryptu *projekt-01-01/www/index.php:*

```
DirectoryIndex index.php
RewriteEngine on
RewriteRule .* index.php
```

Natomiast plik *project-01-01/script/.htaccess* ochroni kod php przed niepowołanym dostępem:

```
<Files ~ ".*">
    Order allow,deny
    Deny from all
</Files>
```

W pasku przeglądarki wpisujemy `http://127.0.0.1/projekt-01-01/`, a następnie `http://127.0.0.1/projekt-01-01/www/` lub

`http://127.0.0.1/projekt-01-01/www/pierwszy-projekt.html`.

Po wpisaniu w pasku adresu przeglądarki jednego z dwóch ostatnich adresów przeglądarka wysyła do serwera żądanie

```
GET /projekt-01-01/www/ HTTP/1.1
```

Reguła:

```
RewriteRule .* index.php
```

zawarta w pliku *project-01-01/www/.htaccess* przekierowuje nas do skryptu *projekt-01-01/www/index.php*, gdzie uruchamiany jest kontroler MVC. Kontroler wykonuje translacje adresu URL: usuwa przedrostek `/projekt-01-01/www` i przeszukuje plik *project-01-01/scripts/translations.txt* w poszukiwaniu adresu przyjaznego. Na podstawie odnalezionej reguły ładuje klasę *Actions* modułu *main* i uruchamia metodę *execute_hello()*. Po czym przetwarza szablon *projekty-01-01/scripts/templates/layout.html* gdzie dołączany jest widok akcji *main/hello* (zawarty w pliku *projekt-01-01/scripts/modules/main/helo.html).* Na koniec w źródle strony można podejrzeć wygenerowany kod HTML.

Zadanie 2 Dołączanie zewnętrznych zasobów

Dla dołączenia zewnętrznych zasobów .css, .jpg i/lub .js musimy podjąć dodatkowe działania w celu przeciwdziałania przekierowaniu wszystkich żądań HTTP do skryptu *projekt-01-01/www/index.php* (reguła `RewriteRule .* index.php` z pliku *project-01-01/script/.htaccess*). Najlepszym rozwiązaniem będzie utworzenie folderu */img*, */css* oraz */js* w katalogu *projekt-01-01/www/* i umieszczeniu w nich pliku *.htaccess:*

```
RewriteEngine off
```

Następnie w celu uniezależnienia folderu dostępu do zasobów od stosowanych przyjaznych URL wykorzystamy zmienną *$path_prefix* prowadzącą do folderu, w którym znajduje się skrypt *projekt-01-01/www/index.php*. Za ustawienie wartości zmiennej *$path_prefix* odpowiada kontroler.

W pliku *projekty-01-01/scripts/templates/layout.html* dopisujemy:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">
  <head>
    <title>Projekt 1.1</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <link rel="stylesheet" type="text/css" href="<?php echo
$path_prefix; ?>css/style.css" />
  </head>
<body>

<?php include "../scripts/modules/$module/$action.html"; ?>

</body>
</html>
```

Tworzymy nowy moduł *projekty-01-01/scripts/modules/obrazy/actions.class.php:*

```
<?php
class Actions extends ActionsBase
{
    public function execute_zdjecie()
    {
    }
}
```

oraz od razu nowy widok akcji *obrazy/zdjecie* w pliku *projekty-01-01/scripts/modules/obrazy/zdjecie.html:*

```
<div>
    <img src="<?php echo $path_prefix; ?>img/%nazwa_pliku%.jpg" alt="" />
</div>

<h1>Wlazłkotek na schody...</h1>
```

Jak widać tutaj także posługujemy się zmienną *$path_prefix*. W miejsce **%nazwa_pliku%** należy wstawić nazwę przykładowego pliku jpg pobranego z internetu i zapisanego w folderze *projekt-01-01/www/img/*.

W następnej kolejności dopisujemy nowe reguły translacji w pliku *project-01-01/scripts/translations.txt*:

```
/pierwszy-projekt.html  index.php?module=main&action=hello
/                   index.php?module=main&action=hello
/index.html   index.php?module=obrazy&action=zdjecie
/a/b/c/d/     index.php?module=obrazy&action=zdjecie
```

Na koniec tworzymy style CSS w pliku *projekt-01-01/www/css/style.css:*

```
body {
    font-family: "Trebuchet MS", sans-serif;
    margin: 0;
    padding: 100px;
    text-align: center;
    font-size: 200%;
    background: url('../img/%nazwa_pliku%.png') repeat-x #fbf4a4;
    color: #7d4902;
}
img {
    border: 1px solid black;
}
```

W miejsce **%nazwa_pliku%** należy wstawić nazwę przykładowego pliku jpg, będącego tłem strony, pobranego z internetu i zapisanego w folderze *projekt-01-01/www/img/*. Nie stosujemy tuaj zmiennej *$path_prefix* ze względu na to, iż ścieżka do pliku z tłem musi być poprawna względem pliku *style.css*, w którym została użyte, a nie względem dokumentu HTML. Na koniec otwieramy w przeglądarce stronę o adresie: http://127.0.0.1/projekt-01-01/www/index.html.

Zadanie 3 Błędy 404

W naszym kontrolerze MVC strona błędu jest generowana automatycznie po stwierdzeniu, że adres URL zawarty w bieżącym żądaniu HTTP nie występuje w pliku *project-01-01/scripts/translations.txt*. Treść komunikatu prezentowanego na stronie błędu powstaje po przetworzeniu widoku *main/404*. W module *main* nie musimy definiować żadnej metody, gdyż metoda *execute_404()* dla akcji *404* jest zdefiniowana w klasie bazowej. Wystarczy, że utworzymy widok *projekty-01-01/scripts/modules/main/404.html:*

```
<p>
Błąd! Strona o podanym adresie nie istnieje!<br />
</p>
```

Uruchamiamy aplikację i wpisujemy błędny adres URL: `http://127.0.0.1/projekt-01-01/www/dfdfd`.

Zadanie 4 Zmienne i widoki

Do przekazywania danych z metody *execute_x()* do widoku *x.html* służy metoda *set()* klasy *Actions*. Jej pierwszym parametrem jest nazwa zmiennej, a drugim – wartość przekazywanej zmiennej.

Na początku *w* pliku *projekt-01-01/scripts/modules/tekst/actions.class.php* tworzymy nowy moduł *tekst* z akcją *drukuj:*

```
<?php

require_once 'vh-array.inc.php';

class Actions extends ActionsBase
{
    public function execute_drukuj()
    {
   $tmp1 = date('d.m.Y');
   $this->set('dzisiejsza_data', $tmp1);

   $tmp2 = date('h:i');
   $this->set('godzina', $tmp2);

   $tmp = file_get_contents('../scripts/dane/ojciec_i_syn.txt');
        $tmp = nl2br($tmp);
        $this->set('tytul', 'Ojciec i syn');
        $this->set('tresc', $tmp);

   $tmp3 = file_get_contents('../scripts/dane/140-kolory-css.txt');
        $dane = string2HArray($tmp3, ':');
        $this->set('kolory', $dane['items']);
    }
}
```

Następnie widok powyższej akcji w pliku *projekt-01-01/scripts/modules/tekst/drukuj.html*:

```
<h1><?php echo $dzisiejsza_data; ?></h1>
```

```
<h1><?php echo $tytul; ?></h1>

<p>
    Godzina: <?php echo $godzina; ?>
</p>

<p>
    <?php echo $tresc; ?>
</p>

<h1>Kolory</h1>
<table>
    <tr>
        <th>Kolor</th>
        <th>Nazwa</th>
        <th>RGB</th>
    </tr>
    <?php foreach ($kolory as $k) : ?>
        <tr>
            <td style='background: #<?php echo $k[0]; ?>'></td>
            <td><?php echo $k[1]; ?></td>
            <td><code><?php echo $k[0]; ?></code></td>
        </tr>
    <?php endforeach; ?>
</table>
```

Dodajemy nowy przyjazny URL w pliku *project-01-01/scripts/translations.txt*:

```
/pierwszy-projekt.html  index.php?module=main&action=hello
/                       index.php?module=main&action=hello
/index.html  index.php?module=obrazy&action=zdjecie
/a/b/c/d/    index.php?module=obrazy&action=zdjecie
/tekst.html  index.php?module=tekst&action=drukuj
```

Na koniec poniższe dwa pliki umieszczamy w folderze *projekt-01-01/scripts/dane*:

`https://db.tt/webHpa7Z`

Uruchamiamy aplikację i w pasku adresu przeglądarki wpisujemy: `http://127.0.0.1/projekt-01-01/www/tekst.html`.

Zadanie 5 Pre- i postprzetwarzanie

Mechanizm pre- i postprzetwarzania polega na tym, że kontroler przed wywołaniem metody *execute_x()* wywołuje metodę wirtualną *preActions()*, po wywołaniu metody *execute_x()* zaś – metodę wirtualną *postActions()*. Dzięki wykorzystaniu polimorfizmu możemy modyfikować kod tych dwóch metod bez konieczności ingerencji w kod biblioteki MVC.

W celu prezentacji zasady działania preprzetwarzania dodamy do naszego projektu moduł, który będzie

prezentował fraszki Jana Kochanowskiego. Na każdej kolejnej podstronie serwisu pojawi się menu główne prezentujące tytuły fraszek dostępnych w serwisie. W tym celu wykorzystamy pliki tekstowe zawierające treść fraszek. Każda fraszka będzie zapisana w osobnym pliku o identycznej strukturze: w pierwszym wierszu pliku występuje tytuł utworu, a w kolejnych – jego treść. Jako adresu URL strony prezentującej wybraną fraszkę użyjemy tytułu fraszki w postaci np. do_goscia.html.

Skoro menu fraszek ma być generowane na wszystkich podstronach serwisu nie może być generowane w żadnym konkretnym module. Jeżeli generowanie menu umieścimy w metodzie *preActions()* kontrolera, wówczas menu będzie dostępne we wszystkich akcjach, wszystkich modułów. W tym celu musimy przygotować naszą własną implementację kontrolera (w pliku *projekt-01-01/scripts/MyController.class.php*), który będzie dziedziczył po kontrolerze z biblioteki MVC:

```
<?php

require_once 'Controller.class.php';
require_once 'slugs.inc.php';

class MyController extends Controller
{
    public function preActions()
    {
        $this->menu = array();    // dzięki temu te dwie zmienne będą
        $this->slugi = array();   // dostępne we wszystkich kontrolerach
        $plks = glob('../scripts/dane/fraszki/*.txt');       // wyszukujemy
        foreach ($plks as $plk) {                          // wszystkie pliki txt
            $tmp = file($plk);
            $opcja = array(
                'nazwapliku' => $plk,
                'tytul' => trim($tmp[0]),
                'slug' => string2slug($tmp[0]),   // zamienia spacje na '_'
            );                                    // i usuwa polskie znaki
            $this->menu[] = $opcja;
            $this->slugi[] = $opcja['slug'];
        }
        $this->set('menu', $this->menu);   // przekazanie tablicy do widoku
    }
}
```

Następnie modyfikujemy skrypt *projekt-01-01/www/index.php*:

```
<?php

error_reporting(E_ALL);
ini_set('display_errors', 1);

set_include_path(
    '../scripts' . PATH_SEPARATOR .
```

```
    '../scripts/include' . PATH_SEPARATOR .
    get_include_path()
);

require_once 'MyController.class.php';

$controller = new MyController();
$controller->dispatch();
```

Potem tworzymy moduł *fraszka* z akcją *show* w pliku *projekt-01-01/scripts/modules/fraszka/actions.class.php*:

```
<?php

class Actions extends ActionsBase
{
    public function execute_show()
    {
        if (
            isset($_GET['slug']) &&
            str_ivslug($_GET['slug']) &&
            in_array($_GET['slug'], $this->controller->slugi)
        ) {
            $index = array_search($_GET['slug'], $this->controller->slugi);
            $tresc = file($this->controller->menu[$index]['nazwapliku']);
            $tresc[0] = '';
            $tresc = implode('', $tresc);
            $this->set('tresc', $tresc);
            $this->set('tytul', $this->controller->menu[$index]['tytul']);
        } else {
            $this->execute_404();
        }
    }
}
```

oraz widok w pliku *projekt-01-01/scripts/modules/fraszka/show.html*:

```
<h3><?php echo $tytul; ?></h3>
<p>
    <?php echo nl2br($tresc); ?>
</p>
```

Do identyfikacji fraszki użyjemy zmiennej URL o nazwie *slug*. Zatem wewnętrzny URL dla fraszki *Do gościa* przyjmie postać:

index.php?module=fraszka&action=show&slug=do_goscia

Kolejnym plikiem, który musimy zmodyfikować, jest układ witryny *projekt-01-01/scripts/templates/layout.html*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">
  <head>
    <title><?php if (isset($tytul)): ?>
             Jan Kochanowski / Fraszki / <?php echo $tytul; ?>
         <?php else: ?>
             Błąd 404!
         <?php endif; ?>
    </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link rel="stylesheet" type="text/css" href="<?php echo $path_prefix; ?
>css/style.css" />
  </head>
<body>
<div id="pojemnik">
    <div id="naglowek">
        <h1>Jan Kochanowski</h1>
        <h2>Fraszki</h2>
        <ol>
            <?php foreach ($menu as $opcja) : ?>
                <li><a href="index.php?module=fraszka&action=show&slug=<?
php echo $opcja['slug']; ?>"><?php echo $opcja['tytul']; ?></a></li>
            <?php endforeach; ?>
        </ol>
    </div>

    <div id="tresc">
       <?php include "../scripts/modules/$module/$action.html"; ?>
    </div>
</body>
</html>
```

Ostatecznie pozostaje nam już tylko dodać przyjazne URL w pliku *projekt-01-01/scripts/translations.txt*:

```
/pierwszy-projekt.html  index.php?module=main&action=hello
/                       index.php?module=main&action=hello
/index.html  index.php?module=obrazy&action=zdjecie
/a/b/c/d/    index.php?module=obrazy&action=zdjecie
/tekst.html      index.php?module=tekst&action=drukuj
/index2.html        index.php?module=fraszka&action=show&slug=do_goscia
/REGEXP1.html     index.php?module=fraszka&action=show&slug=REGEXP1
```

Napis REGEXP1 zawarty w ostatniej regule jest synonimem wyrażenia reguralnego:

([^\/" <>&]+)

Ostatecznie pozostaje nam już tylko umieścić fraszki w folderze *projekt-01-01/scripts/dane/fraszki*:

https://db.tt/PmLJSkyk

Uruchamiamy aplikację i w pasku adresu przeglądarki wpisujemy: `http://127.0.0.1/projekt-01-01/www/index2.html`.

Zadanie 6 Wizualne projektowanie baz danych

Ostatnim brakującym ogniwem architektury MVC w naszym projekcie jest warstwa M. Informacje dostępne na stronach WWW będą przechowywane w bazie danych. Kontroler za pośrednictwem warstwy M pobierze z niej odpowiednie rekordy, po czym przekaże je do widoku.

W celu skrócenia czasu i ułatwienia produkcji aplikacji wykorzystamy dwa nażędzia: wizualny edytor projektowania baz danych oraz oprogramowanie automatycznie generujące klasy PHP zapewniające dostęp do bazy danych z poziomu skryptu PHP.

Do wizualnego projektowania baz danych służy program MySQL Workbench. Uruchamiamy aplikację i wybieramy z menu głównego File->New Model. Dwukrotnie klikamy na **mydb** i w pole *Name* wpisujemy: `tatry`. Następnie z listy *Collation* wybieramy `utf_polish_ci`. Po czym klikamy dwukrotnie ikoną *Add diagram*. Z pionowego paska narzędzi wybieramy ikonę *Place a New Table*, a następnie klikamy w obszar roboczy aplikacji. Pojawi się nowa tabela o nazwie `table1`. Podwójnie klikamy na utworzonej tabeli – otworzy to edytor właściwości tabeli. W polu *Name* wprowadzamy nazwę tabeli: `szczyt` i przechodzimy do zakładki *Columns*. Tworzymy trzy kolumny tabeli:

- `szczyt_id` – klucz główny typu INT (zaznaczamy PK i AI).
- `nazwa` – nazwa szczytu typu VARCHAR (do 45 znaków)
- `wysokosc` – wysokość szczytu w m nad p. m. typu INT

W celu uzyskania dostępu do bazy danych z naszego projektu w aplikacji PHP wykorzystamy oprogramowanie ORM, które wygeneruje nam dla naszego modelu bazy danych klasy dostępu. Pobieramy skrypty generujące:

`https://db.tt/jBUFmmTv`

W programie MySQL Workbench wykonujemy operację *Plugins/Catalog/Doctrine 0.4.1: Copy Generated Doctrine Schema to Clipboard*[1]. Uruchamiamy dowolny edytor plików tekstowych i wykonujemy operację *Edycja->Wklej*. Plik zapisujemy jako *generowanie/input/schema.yml*. Gotowy projekt modelu bazy danych zapisujemy w *generowanie/input/tatry.mwb* oraz wykonujemy operacje eksportowania *File->Export->Forward Engineer SQL CREATE Script...* do pliku *generowanie/input/tatry.sql.* Następnie w edytorze tekstu tworzymy plik konfiguracyjny *generowanie/input/conf.ini:*

```
;ORM
orm       = doctrine
filename  = schema.yml

;database
host      = localhost
database  = tatry
```

1 W razie braku opcji należy zainstalować Propel plugin w MySQL Workbench – instrukcja znajduje się na końcu sprawozdania.

```
username  = admin
password  = password
encoding  = utf8
collate   = utf8_polish_ci

;MySQL root
root         = root
rootpassword = AX1BY2CZ3
```

Oczywiście zmieniamy wartości zmiennych tak aby pasowały do naszych ustawień. Uruchamiamy skrypt *generowanie/run.bat*, po czym *create-db.bat* oraz *create-db-filled.bat*. Pozostaje nam tylko wypełnić tebelę `szczyt`. W tym celu stworzymy skrypt wypełniający *wstawianie/wstaw.php*:

```
<?php

set_include_path(
    '../projekt-01-01/scripts' . PATH_SEPARATOR .
    '../projekt-01-01/scripts/include' . PATH_SEPARATOR .
    '../lib' . PATH_SEPARATOR .
    get_include_path()
);

require_once 'vh-array.inc.php';
require_once 'Doctrine.php';

spl_autoload_register(array('Doctrine', 'autoload'));
$conn                                                                    =
Doctrine_Manager::connection('mysql://login:password@localhost/tatry');
Doctrine::loadModels('./../lib');
$conn->setCollate('utf8_polish_ci');
$conn->setCharset('utf8');

$dane = string2HArray(file_get_contents('tatry.txt'));
foreach ($dane['items'] as $sz) {
    $szczyt = new Szczyt();
    $szczyt->nazwa = $sz[0];
    $szczyt->wysokosc = $sz[1];
    $szczyt->save();
}
```

oraz plik z danymi *wstawianie/tatry.txt:*

```
Świnica                        |     2301
Mały Kozi Wierch               |     2228
Kozi Wierch                     |     2291
Zadni Granat                    |     2240
Pośredni Granat                |     2234
```

```
Skrajny Granat                  |     2225
Kasprowy Wierch                 |     1987
Żółta Turnia                    |     2087
Goryczkowa Czuba                |     1913
Kopa Magury                     |     1704
Beskid                          |     2012
Kościelec                       |     2155
Mały Kościelec                  |     1853
Wielki Wołoszyn                 |     2155
Wielka Buczynowa Turnia         |     2184
Wielka Koszysta                 |     2193
```

Przechodzimy do folderu *wstawianie* i wywołujemy skrypt *wstaw.php* poleceniem:

```
php -f wstaw.php
```

Czas na napisanie modułu *szczyty/lista* obsługującego naszą bazę danych. Tworzymy skrypt *projekt-01-01/scripts/modules/szczyty/actions.class.php*:

```
<?php

class Actions extends ActionsBase
{
    public function execute_lista()
    {
        $szczyty = Doctrine_Query::create()->from('Szczyt')->fetchArray();
        $this->set('szczyty', $szczyty);
    }
}
```

oraz widok *projekt-01-01/scripts/modules/szczyty/lista.html:*

```
<table>
    <tr>
        <th>Szczyt</th>
        <th>Wysokość<br />m n.p.m.</th>
    </tr>

<?php foreach ($szczyty as $sz) : ?>
    <tr>
        <td><?php echo $sz['nazwa']; ?></td>
        <td><?php echo $sz['wysokosc']; ?></td>
    </tr>
<?php endforeach; ?>

</table>
```

Na koniec musimy skopiować folder (i jego zawartość) */lib* wygenerowany przez skrypt generujący do folderu *projekt-01-01/scripts*, zmodyfikować skrypt *projekt-01-01/www/index.php:*

```
<?php

error_reporting(E_ALL);
ini_set('display_errors', 1);

set_include_path(
    '../scripts' . PATH_SEPARATOR .
    '../scripts/include' . PATH_SEPARATOR .
    get_include_path()
);

require_once 'Doctrine.php';
spl_autoload_register(array('Doctrine', 'autoload'));
$conn =
Doctrine_Manager::connection('mysql://login:password@localhost/tatry');
Doctrine::loadModels('./scripts/lib');
$conn->setCollate('utf8_polish_ci');
$conn->setCharset('utf8');

set_include_path('../scripts/lib' . PATH_SEPARATOR . get_include_path());

require_once 'MyController.class.php';
$controller = new MyController();
$controller->dispatch();
```

oraz ustawić przyjazny URL w pliku *projekt-01-01/scripts/translations.txt*:

```
/pierwszy-projekt.html  index.php?module=main&action=hello
/                       index.php?module=main&action=hello
/index.html  index.php?module=obrazy&action=zdjecie
/a/b/c/d/    index.php?module=obrazy&action=zdjecie
/tekst.html      index.php?module=tekst&action=drukuj
/index2.html       index.php?module=fraszka&action=show&slug=do_goscia
/REGEXP1.html     index.php?module=fraszka&action=show&slug=REGEXP1
/tatry.html  index.php?module=szczyty&action=lista
```

Pozostaje nam już tylko uruchomić naszą aplikację: http://127.0.0.1/projekt-01-01/www/tatry.html

Zadanie 7 Własna aplikacja

Wykonaj aplikację internetową prezentującą w postaci witryny WWW …

Zadanie 8 Analiza kodu bibliotek MVC

1) Wyjaśnić mechanizm działania przyjaznych URL.

2) Wyjaśnić mechanizm działania kontrolera.

Zadanie 9 Instrukcja instalacji i konfiguracji serwera bazy danych

Sporządzić instrukcję „krok po kroku” instalacji i konfiguracji serwera MySQL oraz phpMyAdmin.

Instalacja Doctrine plugin w MySQL Workbench

Pobieramy plugin: http://code.google.com/p/mysql-workbench-doctrine-plugin/downloads/detail?name=MySQL-Workbench-Doctrine-Plugin-0.4.1.zip i rozpakowujemy plik .lua.

W programie MySQL Workbench wybieramy z głównego menu Scripting->Install Plugin/Module... i wskazujemy ścieżkę do rozpakowanego pliku. Uruchamiamy ponownie MySQL Workbench.

Instalacja Doctrine

```
pear channel-discover pear.doctrine-project.org
pear remote-list -c doctrine
pear install doctrine/Doctrine
```

Instalacja Smarty UNIX

1. apt-get install smarty
2. Edytujemy /etc/php5/apache2/php.ini. Ustawiamy include_path na ".:/usr/share/php:/usr/share/pear:/usr/share/php/smarty/libs".
3. restartujemy serwer: /etc/init.d/apache2 restart