Systemy wielowarstwowe

Wprowadzenie do framework'a Symfony

Cechy platformy Symfony

- Bazuje na wzorcu projektowym **MVC**
- Niezależność od systemu bazodanowego
- Programowanie zorientowane obiektowo
- Łatwość w instalacji oraz konfiguracji na większości platform
- Zgodność z najlepszymi standardami oraz wzorcami budowy aplikacji internetowych
- Walidacja formularzy i treści
- Wbudowany mechanizm zarządzanie sesjami
- Łatwość rozbudowy oraz możliwość integracji z innymi bibliotekami
- Wykorzystanie technologii **scaffoldingu.**
- Wbudowana internacjonalizacja (i18n)
- Wbudowana ochrona przed atakami CSRF oraz XSS

Kiedy stosowanie Symfony się opłaca?

- Jeśli tworzymy prostą aplikację/stronę internetową składającą się z kilku(nastu) podstron, korzystającą w niewielkim stopniu z bazy danych, bez konieczności tworzenia szczegółowej dokumentacji – lepiej jest wykorzystać zwykłe PHP.
- Jeśli tworzona aplikacja jest bardziej złożona, posiada rozbudowaną warstwę reguł biznesowych i będzie potencjalnie w przyszłości wzbogacana o nowe możliwości – zaleca się skorzystać z platformy Symfony.

Object-Relational Mapping (ORM)

- ORM – interfejs, którego zadaniem jest tłumaczenie logiki obiektowej na logikę relacyjną (by posługiwać się relacyjnymi bazami danych w sposób obiektowy).
- ORM składa się z obiektów definiujących metody dostępu do danych oraz pozwalających zapisać reguły biznesowe (np. automatyczne doliczanie rabatu).
- Warstwa pośrednicząca = oderwanie od konkretnego silnika bazodanowego
- Możliwość definiowania nowych akcesorów:

```
public function getDane()
{
return $this->getImie().' '.$this->getNazwisko();
}
```

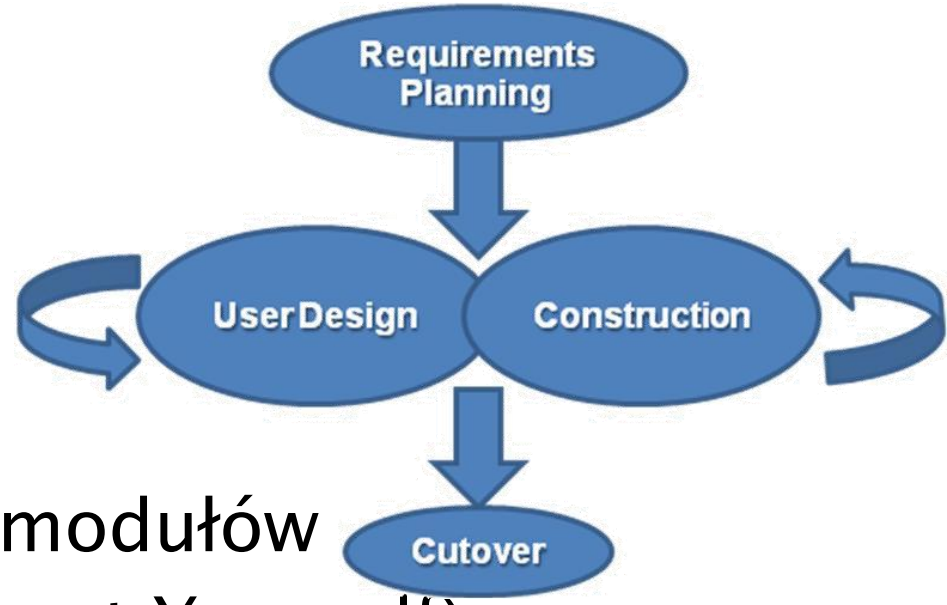
- Symfony wspiera domyślnie dwa ORMy: Propel oraz Doctrine.

Rapid Application Development (RAD)

- Rozpocznij tworzenie tak szybko jak to możliwe
- Proces iteracyjny
- Aplikuje filozofie KISS

(Keep It Simple, Stupid)

- Częsty refactoring
- Wykorzystanie testów modułów
- Zasada DRY (Don’t Repeat Yourself)



YAML Ain't Markup Language

YAML – uniwersalny język formalny przeznaczony do reprezentowania różnych danych w ustrukturalizowany sposób.
Przykład:

PHP

```
$house = array(
    ‘family' => array(
        'name' => 'Doe',
        'parents' => array('John', 'Jane'),
        'children' => array('Paul', 'Mark',
    'Simone')
    ),
    'address' => array(
        'number' => 34,
        'street' => 'Main Street',
        'city' => 'Nowheretown',
        'zipcode' => '12345'
    )
);
```

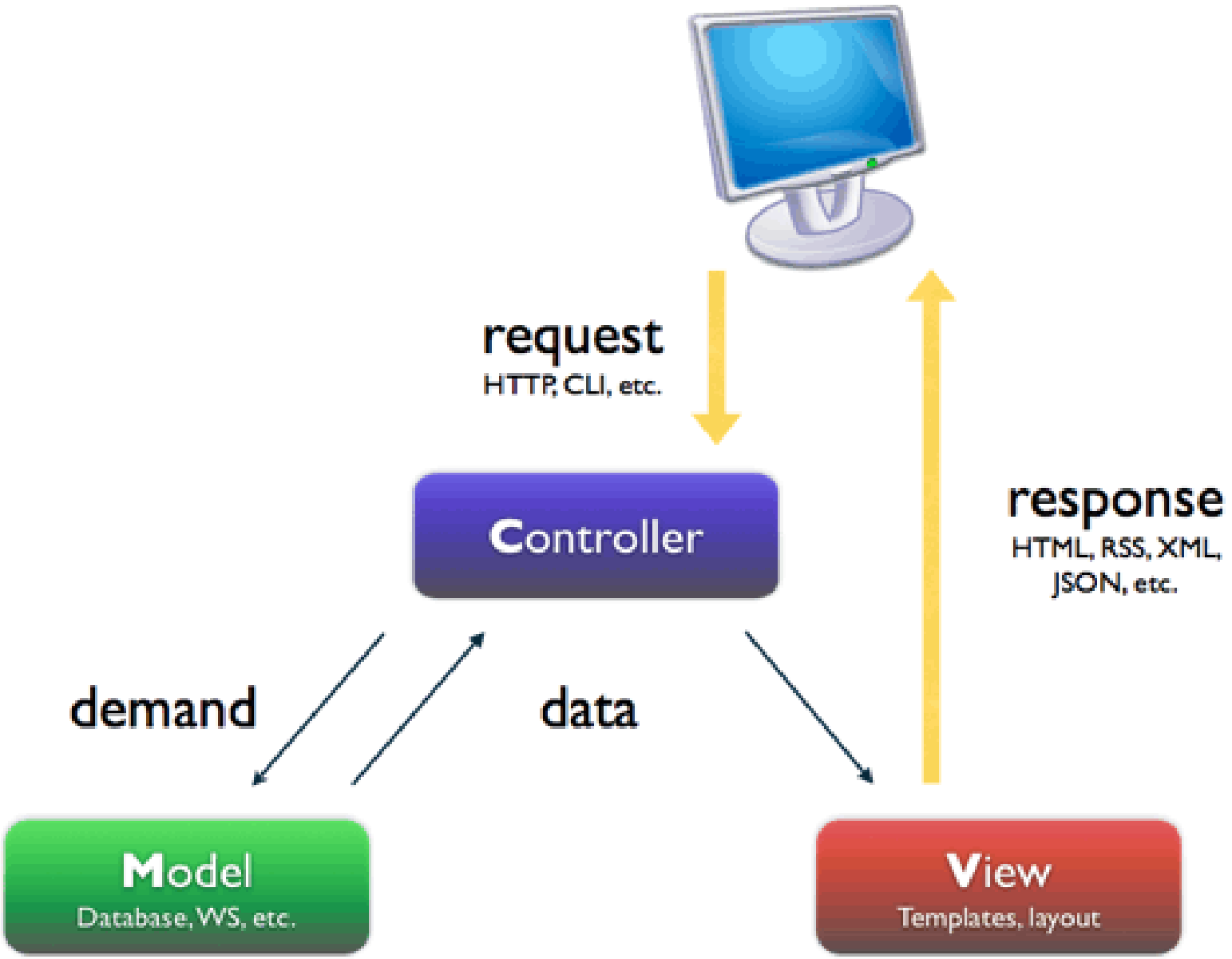
YAML

```
house:
    family:
        name: Doe
        parents:
            - John
            - Jane
        children:
            - Paul
            - Mark
            - Simone
    address:
        number: 34
        street: Main Street
        city: Nowheretown
        zipcode: "12345"
```

Którą wersję symfony wybrać?

- Symfony 1.3 / 1.4
  - Posiadają taką samą funkcjonalność.
  - Wersja 1.4 nie jest kompatybilna z projektami utworzonymi w Symfony 1.2 i niższej.
  - Wsparcie dla Symfony 1.3 zakończyło się w listopadzie 2010.
  - Wsparcie dla Symfony 1.4 przewidywane jest do Listopada 2012.
- Symfony 2.0+
  - Inna struktura katalogów (inne znaczenie katalogu app/ i src/)
  - Zmiana w sposobie automatycznego ładowania klas (nazwa klasy musi być odwzorowana w jej ścieżce)
  - Zmiana w korzystaniu z konsoli (zamiast php symfony jest php app/console)
  - Zmiana roli aplikacji (w jedynce jeden projekt posiadał wiele aplikacji, w dwójce projekt ma jedną aplikacje)
  - Pluginy zostały zastąpione przez tzw. Bundle.
  - Zmiany w zastosowaniu routingu i konfiguracji pluginów.
  - Wprowadzone dystrybucje: standard, HelloWorld

Wzorzec MVC


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML,
JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

Ćwiczenie

- Proszę napisać skrypt w języku PHP, którego celem jest wyświetlenie informacji z bazy danych.
- Proszę stworzyć jedną tabelę, zawierającą sztucznie stworzone informacje na wybrany przez Państwa temat i wyświetlić je na stronie w formie tabelarycznej (znaczniki <table><tr><td>).

Przykład: płaskie PHP na MVC

```
<?php
// Łączenie i wysyłanie zapytania
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);
$result = mysql_query('SELECT date, title FROM post', $link); ?>
<html>
<head>
<title>List of Posts</title>
</head>
<body>
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php
// Wyświetlanie rezultatów
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $row['date']);
printf("\t\t<td> %s </td>\n", $row['title']);
echo "\t</tr>\n";
} ?>
</table>
</body>
</html>
<?php  mysql_close($link);   ?>
```

Rozdział PHP na dwie warstwy: kontrolera i widoku

Kontroler index.php

```
<?php
// Łączenie i wysyłanie zapytania
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);
$result = mysql_query('SELECT date, title FROM post', $link);
// Wypełnianie tablicy z danymi
$posts = array();
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
    $posts[] = $row;
mysql_close($link);
// Dołączenie widoku
require('view.php');
?>
```

Widok view.php

```
<html>
<head>
<title>List of Posts</title>
</head>
<body>
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
<tr>
<td><?php echo $post['date'] ?></td>
<td><?php echo $post['title'] ?></td>
</tr>
<?php endforeach; ?>
</table>
</body>
</html>
```

Rozdział PHP na dwie warstwy: kontrolera i widoku

Model
model.php

```php
<?php
function getAllPosts()
{
    //Łączenie, wysyłanie zapytania, wypełnianie tablicy posts
    $link = mysql_connect('localhost', 'myuser', 'mypassword');
    mysql_select_db('blog_db', $link);
    $result = mysql_query('SELECT date, title FROM post', $link);
    $posts = array();
    while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
        $posts[] = $row;
    mysql_close($link);
    return $posts;
}
?>
```

Kontroler
index.php

```php
<?php
// Dołączenie modelu
require_once('model.php');
// Pobranie listy postów
$posts = getAllPosts();
// Dołączenie widoku
require('view.php');
?>
```

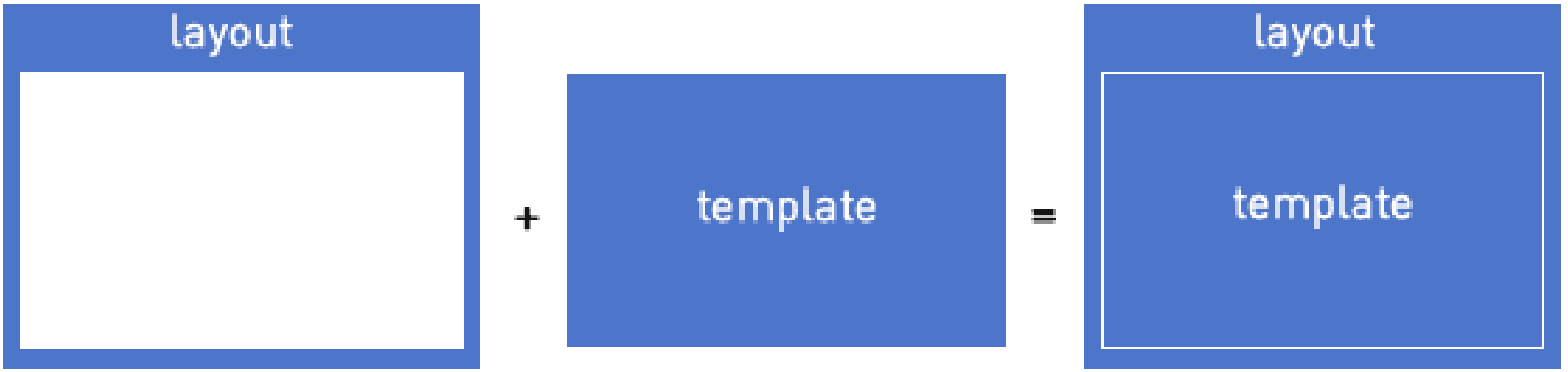
Dalsza abstrakcja modelu

Model
model.php

```
function getAllPosts()
{
    $link = open_connection('localhost', 'myuser', 'mypassword');
    $result = query_database('SELECT date, title FROM post', 'blog_db',
      $link);
    $posts = array();
    while ($row = fetch_results($result))
    {
    $posts[] = $row;
    }
    close_connection($link);
    return $posts;
}
function open_connection($host, $user, $password)
        return mysql_connect($host, $user, $password);

function close_connection($link)
        mysql_close($link);

function query_database($query, $database, $link) {
     mysql_select_db($database, $link);
     return mysql_query($query, $link);
}

function fetch_results($result)
        return mysql_fetch_array($result, MYSQL_ASSOC);
```

Warstwa pośrednia do mysql

Dalszy podział warstwy widoku



Dalszy podział warstwy widoku

```
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['date'] ?></td>
        <td><?php echo $post['title'] ?></td>
    </tr>
<?php endforeach; ?>
</table>
```

mytemplate.php

```
<?php
$title = 'List of Posts';
$posts = getAllPosts();
include(layout.php);
?>
```

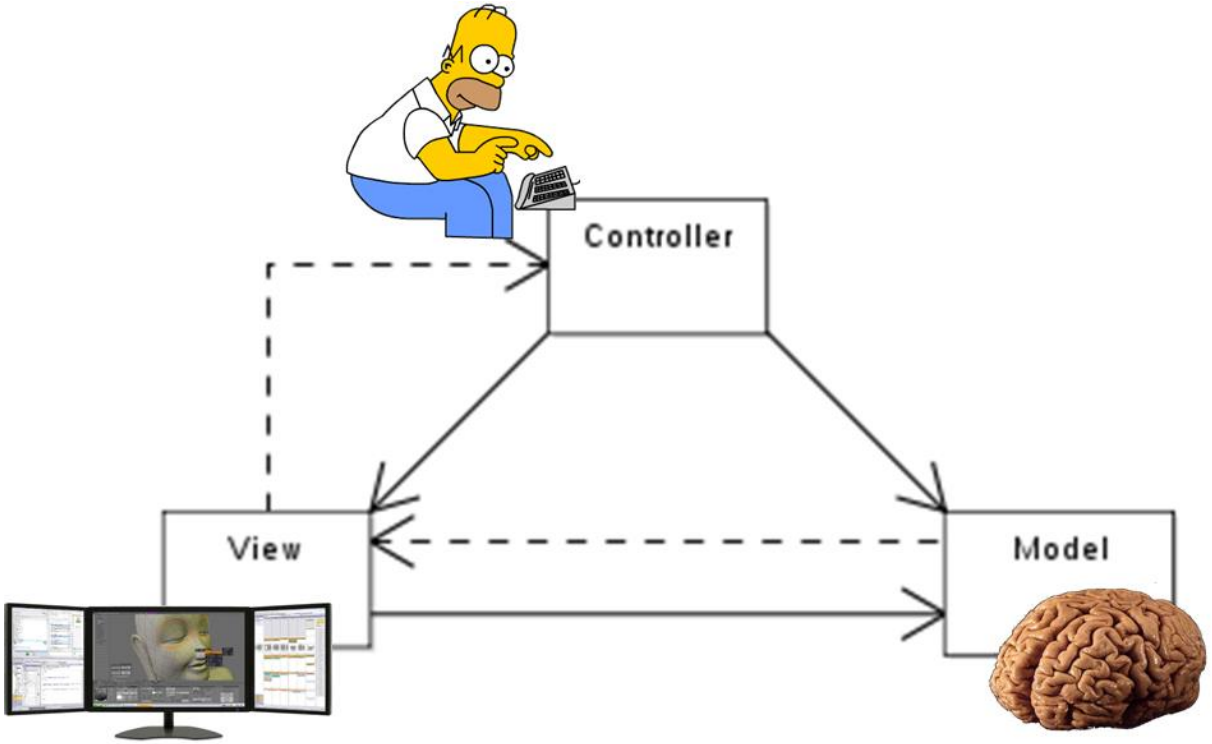
view.php

```
<html>
<head>
<title><?php echo $title ?></title>
</head>
<body>
<?php include('mytemplate.php'); ?>
</body>
</html>
```

layout.php

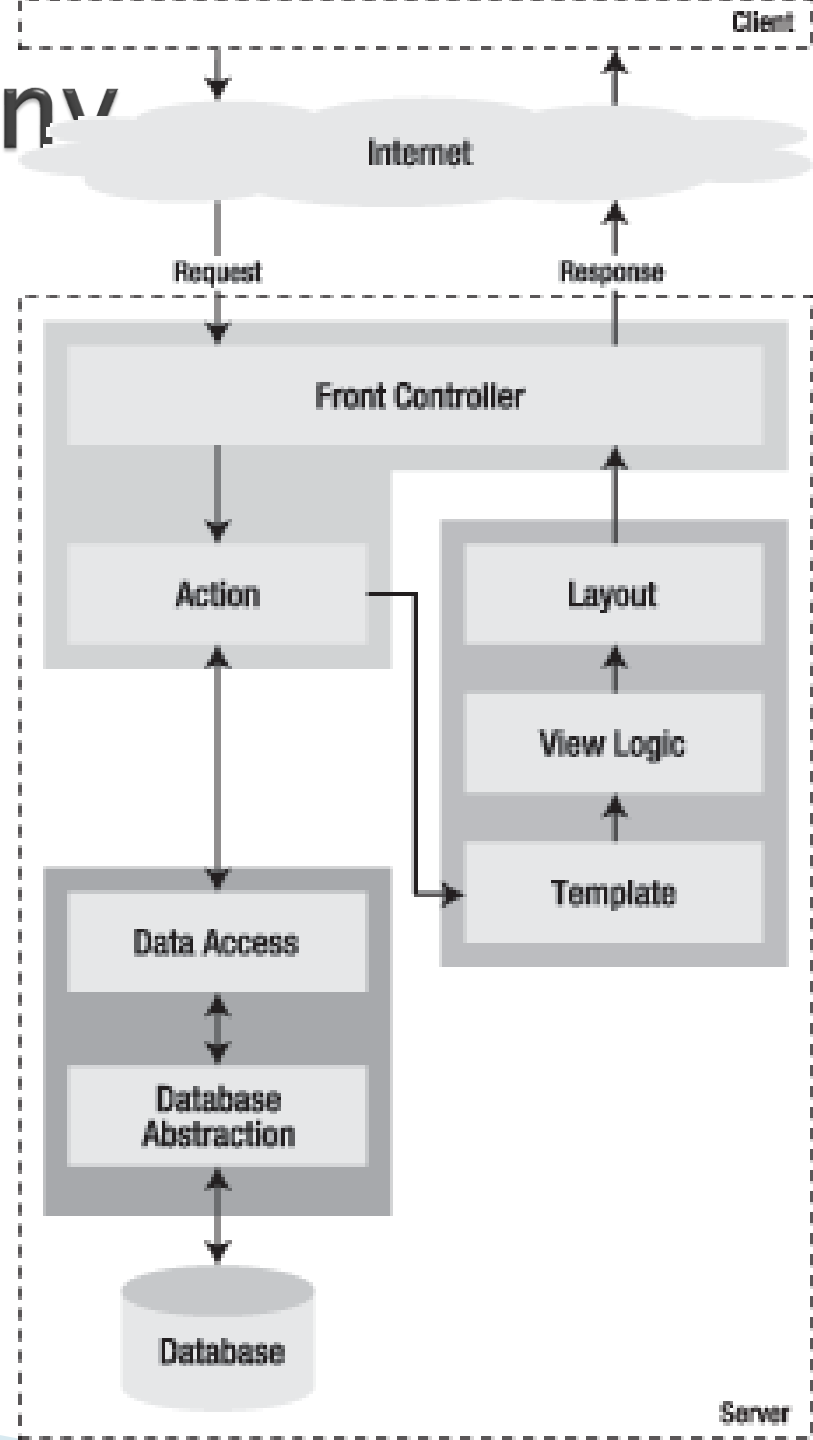
Ćwiczenie

- Proszę przepisać stworzony w poprzednim ćwiczeniu skrypt, tak aby był zgodny z modelem MVC.



Model MVC w Symfony

- Podział kontrolera
  - Front controller
  - Akcje (actions)
- Elementy wymagane do stworzenia strony w Symfony:
  - Model layer
    - Database abstraction
    - Data access
  - View layer
    - View
    - Template
    - Layout
  - Controller layer
    - Front controller
    - Action



Ten sam przykład w Symfony

List Action

```php
<?php
class weblogActions extends sfActions {
    public function executeList() {
        $this->posts = PostPeer::doSelect(new Criteria());
    }
}
```

List Template

```
<?php slot('title', 'List of Posts') ?>
<h1>List of Posts</h1>
<table>
    <tr><th>Date</th><th>Title</th></tr>
    <?php foreach ($posts as $post): ?>
        <tr>
            <td><?php echo $post->getDate() ?></td>
            <td><?php echo $post->getTitle() ?></td>
        </tr>
    <?php endforeach; ?>
</table>
```

Layout

```
<html>
<head>
<title><?php include_slot('title') ?></title>
</head>
<body>
<?php echo $sf_content ?>
</body>
</html>
```

Instalacja frameworka

- Musisz mieć zainstalowany lokalnie Apache + PHP + MySQL.

Np. XAMPP wersja **portable**.

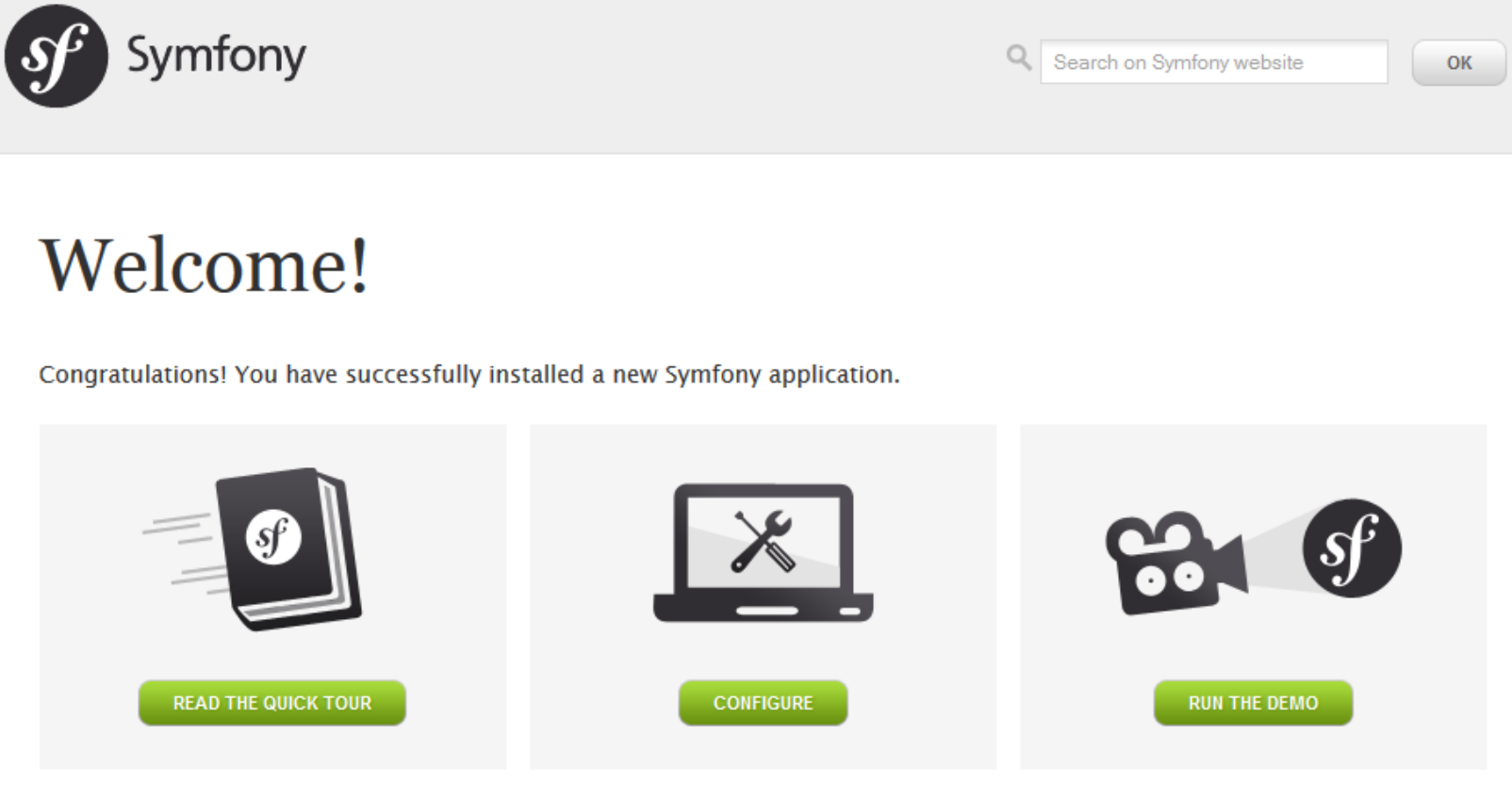
- http://www.ens.ro/2012/03/22/symfony2-jobeet-day-1-starting-up-the-project/
- Zainstaluj środowisko.
- Katalog z frameworkiem powinien mieć nazwę **StudAd** (na stronie jest jobeet).

Projekt powinien być dostępny pod adresem:
http://localhost:8080

Proszę włączyć odpowiednie moduły apache (np. *rewrite_module*)

Proszę nie pobierać wersji z subversion.

Poprawnie zainstalowane Symfony



Tworzenie projektu

- Stwórz nowy projekt o nazwie Ens/StudAdBundle.
- Polecam do katalogu z symfony (StudAd) dodać symfony.bat (modyfikując ścieżkę do php). Wówczas wystarczy:

```
symfony list
```

Zamiast

```
php app/console list
```

Na następne zajęcia: gotowe instalacje Symphony, stworzone projekty.

Sprawdź poprawność wygenerowanego projektu – w katalogu StudAd/src powinny znajdować się podkatalogi Ens/StudAdBundle