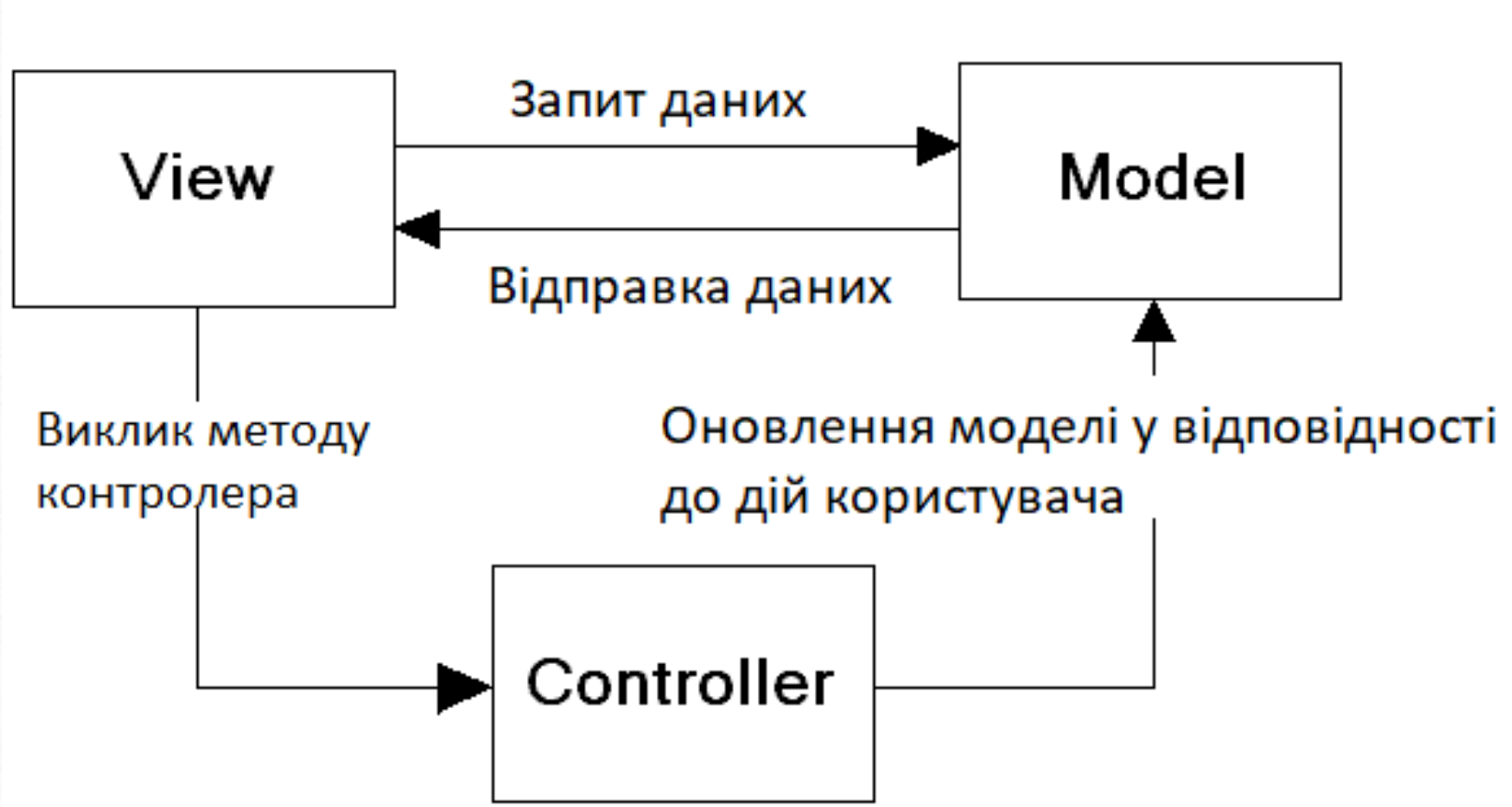
Серверні технології та бекенд розробка

лек

Лекція. Реалізація MVC патерну на прикладі створення сайту-візитки на PHP

Зав.каф.КН, PhD Граф М.С.

Теорія



В архітектурі MVC модель надає дані та правила бізнес-логіки, подання відповідає за інтерфейс користувача, а контролер забезпечує взаємодію між моделлю та поданням.

Типову послідовність роботи MVC-програми можна описати так:

1.При заході користувача на веб-ресурс, скрипт ініціалізації створює екземпляр програми та запускає його на виконання.
При цьому відображається вигляд, скажімо, головної сторінки сайту.

2.Програма отримує запит від користувача та визначає запитані контролер та дію. У разі головної сторінки виконується дія за замовчуванням (*index*).

3.Додаток створює екземпляр контролера і запускає спосіб дії,
у якому, наприклад, утримуватися виклики моделі, зчитують інформацію з даних.

4.Після цього дія формує подання з даними, отриманими з моделі і виводить результат користувачеві.

Модель містить бізнес-логіку програми і включає методи вибірки (це можуть бути методи ORM), обробки (наприклад, правила валідації) і надання конкретних даних, що часто робить її дуже товстою, що цілком нормально.

Вигляд — використовується для визначення зовнішнього відображення даних, отриманих з контролера та моделі.

Контролер — зв'язуюча ланка, що з'єднує моделі, види та інші компоненти робочого додатку. Контролер відповідає за обробку запитів користувача. Контролер не повинен містити SQL-запитів. Їх краще тримати у моделях. Контролер не повинен містити HTML та іншу розмітку. Її варто виносити у види.

Front Controller і Page Controller

Розглянемо два варіанти адресного рядка, якими показується якийсь текст і профіль користувача.

Перший варіант:

1.www.example.com/article.php?id=3
2.www.example.com/user.php?id=4

Тут кожен сценарій відповідає за виконання певної команди.

Другий варіант:

1.www.example.com/index.php?article=3
2.www.example.com/index.php?user=4

А тут усі звернення відбуваються в одному сценарії **index.php** .

Маршрутизація URL

Маршрутизація URL дозволяє налаштувати додаток на прийом запитів з URL, які не відповідають реальним файлам програми, а також використовувати CNC , які семантично значущі для користувачів та кращі для пошукової оптимізації.

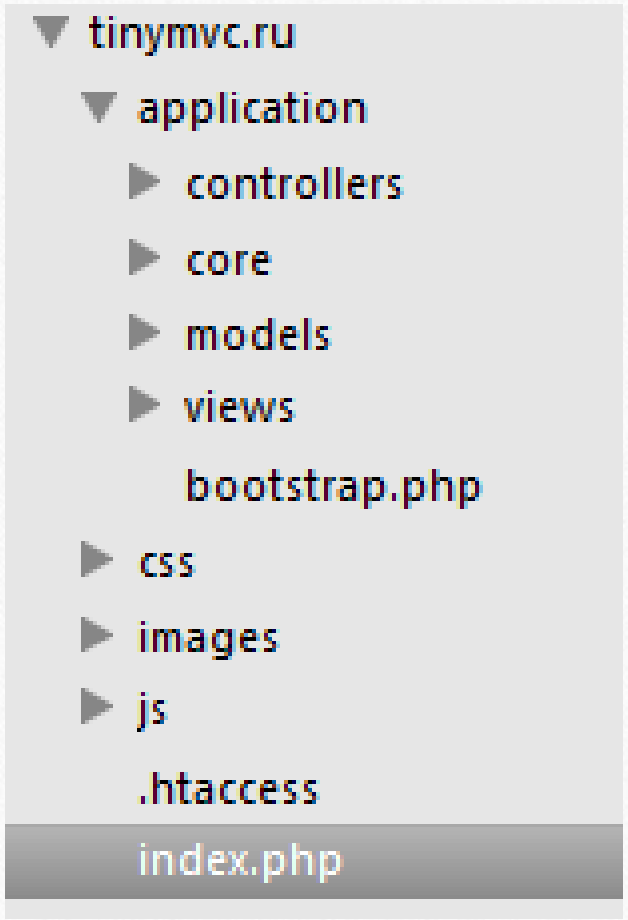
Наприклад, для звичайної сторінки, що відображає форму зворотного зв'язку, URL міг би виглядати так: http://www.example.com/contacts.php?action=feedback

Приблизний код обробки в такому випадку:

```
switch($_GET['action'])
{
    case "about" :
        require_once("about.php"); //
        break;
    case "contacts" :
        require_once("contacts.php");
        break;
    case "feedback" :
        require_once("feedback.php");
        break;
    default :
        require_once("page404.php");
    break;
}
```

Практика

Для початку створюється наступну структуру файлів та папок:



index.php

```php
ini_set('display_errors', 1);
require_once 'application/bootstrap.php';
```

bootstrap.php

```php
require_once 'core/model.php';
require_once 'core/view.php';
require_once 'core/controller.php';
require_once 'core/route.php';
Route::start();
```

Практика

Реалізація маршрутизатора URL

Перший крок, який нам потрібно зробити, записати наступний код в **.htaccess** :

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .* index.php [L]
```

Маршрутизацію ми помістимо в окремий файл **route.php** директорію core. У цьому файлі опишемо клас Route, який запускатиме методи контролерів, які у свою чергу генеруватимуть вигляд сторінок.

Практика Реалізація маршрутизатора URL

```php
class Route
{
   static function start()
   {
      // контроллер та дія за замовчанням
      $controller_name = 'Main';
      $action_name = 'index';

      $routes = explode('/', $_SERVER['REQUEST_URI']);

      // потримуємо ім'я контролера
      if ( !empty($routes[1]) )
      {
         $controller_name = $routes[1];
      }

      // потримуЗємо ім'я екшена
      if ( !empty($routes[2]) )
      {
         $action_name = $routes[2];
      }

      // додаємо префікси
      $model_name = 'Model_'.$controller_name;
      $controller_name = 'Controller_'.$controller_name;
      $action_name = 'action_'.$action_name;

      // подхоплюємо файл з класом моделі (файлу моделі може і не бути)

      $model_file = strtolower($model_name).'.php';
      $model_path = "application/models/".$model_file;
      if(file_exists($model_path))
      {
         include "application/models/".$model_file;
      }
```

route.php

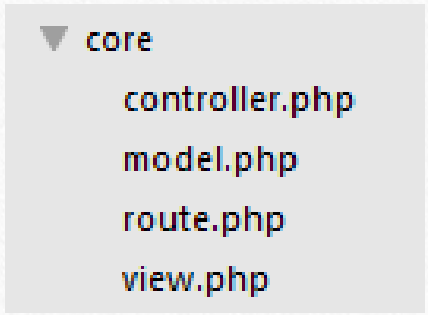
```php
      // підхоплюємо файл з класом контроллера
      $controller_file = strtolower($controller_name).'.php';
      $controller_path = "application/controllers/".$controller_file;
      if(file_exists($controller_path))
      {
         include "application/controllers/".$controller_file;
      }
      else
      {          /*  правильніше було б тут прописати виключення,   але для спрощення одразу робимо редирект на сторінку 404    */
         Route::ErrorPage404();
      }
      // створюємо контроллер
      $controller = new $controller_name;
      $action = $action_name;

      if(method_exists($controller, $action))
      {
         // викликаємо дію контроллера
         $controller->$action();
      }
      else
      {      // тут також можна прописати виключення
         Route::ErrorPage404();
      }

   }
   function ErrorPage404()
   {
      $host = 'http://'.$_SERVER['HTTP_HOST'].'/';
      header('HTTP/1.1 404 Not Found');
      header("Status: 404 Not Found");
      header('Location:'.$host.'404');
   }
}
```

роутер виконає такі дії:

1.підключить файл model_portfolio.php із папки models, що містить клас Model_Portfolio;
2.підключить файл controller_portfolio.php із папки controllers, що містить клас Controller_Portfolio;
3.створить екземпляр класу Controller_Portfolio і викличе стандартну дію — action_index, описану в ньому.

Якщо користувач спробує звернутися за адресою неіснуючого контролера, наприклад:
example.com/ufo
його перекине на сторінку «404»:
example.com/404
Те ж саме станеться якщо користувач звернеться до дії, яка не описана в контролері.

реалізації MVC, продовження



model.php

```php
class Model
{

    public function get_data()
    {
    }
}
```

view.php

```php
class View
{
    //public $template_view; // тут можна вказати загальний вигляд за замовчанням.

    function generate($content_view, $template_view, $data = null)
    {
        /*
        if(is_array($data)) {
            // перетворюємо елементи масиву у змінні
            extract($data);
        }
        */

        include 'application/views/'.$template_view;
    }
}
```

controller.php

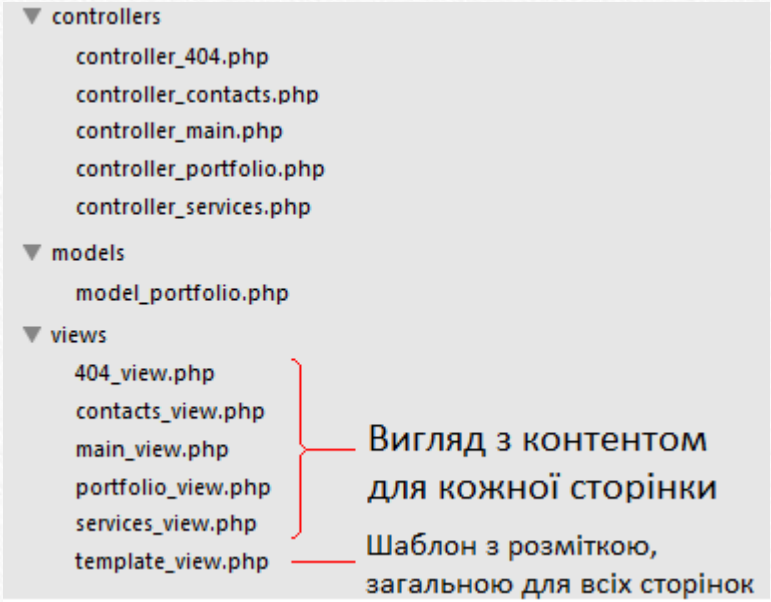
```php
class Controller {

    public $model;
    public $view;

    function __construct()
    {
        $this->view = new View();
    }

    function action_index()
    {
    }
}
```

Метод *action_index* - це дія, що викликається за умовчанням, його ми перекриємо при реалізації класів нащадків.

Реалізація класів нащадків Model та Controller, створення View's

Наш сайт-візитка складатиметься з наступних сторінок:

1.Головна
2.Послуги
3.Портфоліо
4.Контакти
5.А також сторінка «404»



template_view.php

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="utf-8">
    <title>Главная</title>
</head>
<body>
    <?php include 'application/views/'.$content_view; ?>
</body>
</html>
```

Для надання сайту презентабельного вигляду зверстаємо CSS шаблон і інтегруємо його в наш сайт шляхом зміни структури HTML-розмітки та підключення CSS та JavaScript файлів:

```
<link rel="stylesheet" type="text/css" href="/css/style.css" />
<script src="/js/jquery-1.6.2.js" type="text/javascript"></script>
```

Реалізація класів нащадків Model та Controller, створення View's

Створюємо головну сторінку

controller_main.php

```php
class Controller_Main extends Controller
{

    function action_index()
    {
        $this->view->generate('main_view.php', 'template_view.php');
    }
}
```

main_view.php

```html
<h1>Вітаємо Вас!</h1>
<p>
<img src="/images/office-small.jpg" align="left" >
<a href="/">Назва/a> - якийс опис даної назви
</p>
```

Для відображення головної сторінки можна скористатися однією з наступних адрес:

- example.com
- example.com/main
- example.com/main/index

Реалізація класів нащадків Model та Controller, створення View's

Реалізація класів нащадків Model та Controller, створення View's

Створюємо сторінку «Портфоліо»

Файл моделі **model_portfolio.php** помістимо до папки models. Ось його вміст:

```php
class Model_Portfolio extends Model
{
    public function get_data()
    {
        return array(

            array(
                'Year' => '2012',
                'Site' => 'http://адреса',
                'Description' => 'Промо-сайт темного пива Kozel від і т.д.'
            ),
            array(
                'Year' => '2012',
                'Site' => 'http://fff.ua',
                'Description' => 'Україномовний каталог телефонів компанії Zopo і т.д.'
            ),
            // todo
        );
    }
}
```

Реалізація класів нащадків Model та Controller, створення View's

Створюємо сторінку «Портфоліо»

Клас контролера моделі міститься у файлі **controller_portfolio.php** , ось його код:

```php
class Controller_Portfolio extends Controller
{

    function __construct()
    {
        $this->model = new Model_Portfolio();
        $this->view = new View();
    }

    function action_index()
    {
        $data = $this->model->get_data();
        $this->view->generate('portfolio_view.php', 'template_view.php', $data);
    }
}
```

Реалізація класів нащадків Model та Controller, створення View's

Створюємо сторінку «Портфоліо»

portfolio_view.php .

```
<h1>Портфоліо</h1>
<p>
<table>
    Всі наступні проекти є видуманими
    <tr><td>Год</td><td>Проект</td><td>Опис</td></tr>
    <?php

    foreach($data as $row)
    {
        echo '<tr><td>'.$row['Year'].'</td><td>'.$row['Site'].'</td><td>'.$row['Description'].'</td></tr>';
    }

    ?>
</table>
</p>
```

Висновки

Шаблон MVC використовується як архітектурна основа в багатьох фреймворках і CMS, які створювалися для того, щоб мати можливість розробляти якісно складніші рішення за більш короткий термін. Це стало можливо завдяки підвищенню рівня абстракції, оскільки є межа складності конструкцій, якими може оперувати людський мозок.

Але використання веб-фреймворків, типу Yii або Kohana, що складаються з декількох сотень файлів, при розробці простих веб-додатків (наприклад, сайтів-візиток) не завжди доцільно. Тепер ми вміємо створювати красиву MVC модель, щоб не перемішувати Php, Html, CSS та JavaScript код в одному файлі.

Дякую за увагу!