**Практична робота №6**

ВИКОРИСТАННЯ MVC АРХІТЕКТУРИ ПРОЄКТУ ДЛЯ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ

**Мета роботи:** отримання навичок розробки веб-застосунку з використанням архітектурного шаблону Model-View-Controller (MVC) на мові програмування PHP. Метою також є освоєння практичних навичок роботи з базами даних, обробки форм, а також підтримки коду відповідно до стандартів та рекомендацій. Результативне виконання цієї практичної роботи повинно забезпечити студенту глибоке розуміння та впевненість у використанні паттерну MVC для створення ефективних та добре організованих веб-застосунків на PHP.

1. **Теоретичні відомості. MVC для розробки сайтів на PHP**

Коли ми розробляємо сайт, то пишемо HTML та PHP код в тому самому файлі, перемішуючи ці мови між собою. Якщо проект маленький, нічого страшного в цьому немає. Але якщо проект почне зростати, то через деякий час його досить важко підтримуватиме через змішування мов. З часом проект ставатиме все більше і більше, код ставатиме все заплутанішим, і вам важко буде його підтримувати.

Для вирішення таких проблем було придумано спеціальний підхід під назвою MVC. Ця абревіатура розшифровується так: M - модель (model), V - уявлення (view), С - контролер (controller). При цьому модель відповідає за отримання даних, подання - за виведення цих даних на екран у потрібному вигляді, а контролер є сполучною ланкою між моделлю і поданням.

На основі MVC створено всі PHP фреймворки. Фреймворк є заготовкою проекту, що спрощує розробку сайту PHP програмістом, позбавляючи його багатьох рутинних операцій. Крім того, фреймворк є набором правил, яким необхідно слідувати при розробці. Дотримання цих правил вносить чітку структуру в проект, забезпечуючи легку підтримку створеного коду, а також легко брати нових людей у ​​проект. Одним із таких правил і є підхід MVC.

Як працює MVC

Розглянемо як працює сайт на MVC. Все починається з того, що користувач вбиває адресу в адресний рядок (найчастіше перейшовши за посиланням). У цей момент контролер розуміє, що саме хотів отримати користувач і просить модель дати відповідні дані. Модель дає контролеру дані. Контролер отримує ці дані та відправляє їх у виставу. Подання показує дані в оформленому потрібному вигляді.

При цьому ми отримуємо поділ праці: контролер не знає, як отримати дані - він просто просить про це модель, уявлення не знає, звідки приходять дані - воно просто їх відображає.

Упевнений, що поки це все звучить не зрозуміло, тому спеціально для вас зробив навчальний міні фреймворк, який використовує MVC підхід. Спочатку я навчу вас користуватися цим фреймворком, щоб ви зрозуміли, як саме слід розробляти на MVC. А потім ви самі, дотримуючись моїх вказівок, напишіть такий самий фреймворк на ООП.

1. **Установка фреймворку.**

***Важливо також можна використовувати Laravel або Symfony framework.***

Для початку фреймворк необхідно завантажити за наступним посиланням:[скачати фреймворк](https://drive.google.com/file/d/1tMYFP_6R-Bi1qAWJX0LyeI2xo2QDLnYW/view?usp=drive_link). Розпакуйте файли фреймворку в якусь папку сервера, наприклад, у framework.local. У PhpMyAdmin створіть базу даних для вашого фреймворку. Потім перейдіть у файл /project/config/connection.php та налаштуйте там підключення до бази даних. Для перевірки роботи перейдіть на наступну адресу: http://adressyourserver/hello/. Там ви маєте побачити вітальне повідомлення. Якщо ви його бачите – значить все працює.

Структура папок

Після розпакування фреймворку у вас будуть дві папки: папка core та папка project.

У папці core розміщується написане мною ядро ​​нашого фреймворку. Під ядром розуміється код, який забезпечує функціонал, яким ви користуватиметеся під час розробки. При використанні фреймворку в цю папку заглядати вам, як правило, не потрібно. У папці project розміщуватимуться файли вашого проекту: контролери, моделі, уявлення, а також налаштування фреймворку. Саме в цій папці вестиметься вся робота з розробки сайту.

1. **Контролери, дії та роути в MVC на PHP**

Перше, із чим ми розберемося – це контролери. Контролери обробляють запити користувача, розуміють, що хотів попросити користувач у сайту, просять відповідні дані з моделі та надсилають їх у виставу.

Контролери є ОВП класи. Один файл – це один клас і, відповідно, один контролер. У нашому фреймворку контролери зберігатимуться у папці project/controllers.

Давайте потренуємось у створенні контролерів. Для розминки зробимо клас PageController, який управлятиме текстовими сторінками на нашому сайті.

Давайте одразу створимо файл для нашого контролера. За правилами нашого фреймворку кожен клас повинен зберігатись у файлі з однойменною назвою (аж до регістру). Тобто наш клас PageController зберігається у файлі PageController.php. Створіть цей файл у папці project/controllers.

Зробимо у цьому файлі наш клас:

<?php

namespace Project\Controllers;

use \Core\Controller;

classPageController extends Controller

{

}

?>

Як ви бачите, наш клас належить простору імен Project\Controllers, дотримуючись угоди про автозавантаження файлів (тобто шлях папок повинен збігатися з простором імен).

Крім того, наш клас успадковується від класу Core Controller, що знаходиться в ядрі фреймворку. У цьому не потрібно шукати глибокий зміст, а просто потрібно прийняти зазвичай фреймворку. Ось це правило: всі створювані вами контролери повинні успадковуватися від класу Core Controller, щоб все працювало, як треба.

Дії

Додамо тепер до нашого контролера методи класу. У термінах MVC методи контролерів називаються діями (англ. actions). Зробимо, наприклад, дві дії - show1 і show2, і в кожній дії виведемо що-небудь на екран:

<?php

namespace Project\Controllers;

use \Core\Controller;

classPageController extends Controller

{

public function show1()

{

echo '1';

}

public function show2()

{

echo '2';

}

}

?>

**2. Роутінг**

Тепер вам необхідно познайомитися з таким поняттям як роутинг. Він є механізмом, за допомогою якого можна викликати певну дію певного контролера через адресний рядок браузера.

Налаштування роутингу зберігаються у файлі /project/config/routes.php і є масивом об'єктів класу \Core\Route. Конструктор цього класу першим параметром приймає URI, на запит якого викликається відповідний метод відповідного контролера. Ім'я контролера та ім'я дії задаються другим та третім параметрами. При цьому ім'я контролера визначається з маленької літери.

Давайте для прикладу додамо два роути (тобто маршруту): перший при зверненні до адреси /my-page1/ викликатиме метод show1 контролера page, а другий - метод show2 цього ж контролера:

<?php

use \Core\Route;

return [

newRoute('/my-page1/', 'page', 'show1'),

newRoute('/my-page2/', 'page', 'show2'),

];

?>

**Параметри в роутах у MVC на PHP**

У попередньому уроці наші роути мали фіксовані адреси. Насправді механізм роутингу складніший - можна зробити те щоб частина URI сторінки потрапляла в іменовані параметри, доступні потім у контролері.

Нехай, наприклад, наші адреси будуть виглядати так: /test/параметр1/параметр2/, де параметр1 і параметр2 - довільні рядки. Ми хочемо, щоб адреси такого виду оброблялися однією дією контролера. Для цього слід вигадати ім'я параметра і перед ним поставити двокрапку, ось так:

<?php

use \Core\Route;

return [

newRoute('/test/:var1/:var2/', 'page', 'act'),

];

?>

У нашому випадку вийде, що всі запити виду /test/параметр1/параметр2/ потраплятимуть на дію act. При цьому перший параметр цієї дії буде потрапляти асоціативний масив з параметрами: текст, який буде на місці першого параметра, потрапляє в елемент масиву з ключем 'var1', а текст другого параметра - в 'var2'.

Нехай, наприклад, в адресному рядку набрано таке: /test/eee/bbb/. Давайте подивимося, що міститиме перший параметр дії:

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

public function act($params)

{

var\_dump($params); // ['var1' => 'eee', 'var2' => 'bbb']

}

}

?>

**Приклад**

Давайте подивимося застосування описаного більш життєвому прикладі. Нехай наш контролер PageController містить масив сторінок (ці дані має віддавати модель, але з моделями ми ще не розібралися, тому нехай дані поки що просто зберігаються в контролері):

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

private $pages;

public function \_\_construct()

{

$this->pages = [

1 => 'сторінка 1',

2 => 'сторінка 2',

3 => 'сторінка 3',

];

}

}

?>

Давайте зробимо дію show, яка виводитиме на екран сторінку з певним номером (id):

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

private $pages;

public function \_\_construct()

{

$this->pages = [

1 => 'сторінка 1',

2 => 'сторінка 2',

3 => 'сторінка 3',

];

}

public function show()

{

// Тут виведемо сторінку з певним номером

}

}

?>

Нехай при зверненні на адресу /page/1/ буде виводиться текст першої сторінки, при зверненні до адреси /page/2/ - текст другої сторінки тощо. Зробимо відповідний роут:

<?php

use \Core\Route;

return [

newRoute('/page/:id/', 'page', 'show'),

];

?>

Реалізуємо описаний метод show:

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

private $pages;

public function \_\_construct()

{

$this->pages = [

1 => 'сторінка 1',

2 => 'сторінка 2',

3 => 'сторінка 3',

];

}

public function show($params)

{

echo $this->pages[ $params['id'] ]; // виводимо сторінку за номером

}

}

?>

Подання в MVC на PHP

Давайте тепер додамо нашим діям уявлення. За правилами нашого фреймворку уявлення зберігаються у папці /project/views/.

Крім того, у нашому фреймворку діє угода: ім'я файлу з поданням має збігатися з ім'ям дії та розміщуватися в папці з такою самою назвою, як ім'я контролера. Наприклад, якщо у нас є контролер PageController, всі його уявлення слід зберігати в папці /project/views/page/. Сам файл уявлення повинен мати розширення .php. Наприклад, якщо в контролері PageController буде дія act, його подання слід розміщувати в /project/views/page/act.php.

Для того, щоб задати уявлення якійсь дії, ця дія має викликати метод render, передавши йому параметр ім'я подання. Ім'я подання має складатися з імені контролера з маленької літери, після якого через / буде записано ім'я дії

Метод render не потрібно реалізовувати в контролері – він успадковується від батьківського класу Core Controller. З іншого боку, для коректної роботи результат роботи методу слід повернути через return.

Отже, нехай ми маємо контролер Page з дією act. Давайте прив'яжемо до цієї дії уявлення:

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

public functionact()

{

return $this->render('page/act');

}

}

?>

Зробимо тепер файл з HTML кодом нашої вистави:

<div>

це уявлення

діїact контролера page

</div>

Передача даних на подання

За допомогою другого параметра методу render можна передавати дані з контролера у виставу. Дані слід передавати як асоціативного масиву. При цьому в поданні ключі масиву стануть змінними з відповідними значеннями.

Давайте подивимося на практиці. Передамо в наше представлення масив із трьома елементами:

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

public functionact()

{

return $this->render('page/act', [

'var1' => 'eee',

'var2' => 'bbb',

'var3' => 'kkk',

]);

}

}

?>

Як ви бачите, ключі нашого масиву - 'var1', 'var2' та 'var3'. Це означає, що у поданні будуть доступні такі змінні і ми можемо вивести їх значення у місцях HTML коду. Зробимо це:

<div>

це уявлення

діїact контролера test

</div>

<ul>

<li><?php echo $ var1; ?></li>

<li><?php echo $ var2; ?></li>

<li><?php echo $ var3; ?></li>

</ul>

**Передача масивів**

На виставу можна також передавати і масиви. Давайте, наприклад, передамо масив користувачів:

<?php

namespace Project\Controllers;

use Core\Controller;

classPageController extends Controller

{

public functionact()

{

return $this->render('page/act', [

'header' => 'список користувачів',

'users' => ['user1', 'user2', 'user3'],

]);

}

}

?>

Можна звернутися до кожного елемента масиву:

<h1><?= $header ?></h1>

<ul>

<li><?= $users[0]; ?></li>

<li><?= $users[1]; ?></li>

<li><?= $users[2]; ?></li>

</ul>

А можна перебрати масив циклом і сформувати необхідний HTML код:

<h1><?= $header ?></h1>

<ul>

<?php foreach ($users as $user): ?>

<li><?= $user; ?></li>

<?php endforeach; ?>

</ul>

**Шаблони в MVC на PHP**

Уявлення, які ми з вами вивчали в попередньому уроці, насправді є контентом сторінки. Окрім контенту, на сторінці, як правило, є ще хедер, сайдбари, футер. Ці частини зазвичай однакові на всіх сторінках сайту.

У нашому фреймворку кожна сторінка сайту є той самий HTML файл шаблону, якого для кожної сторінки сайту підключається в задане місце контент сторінки з представлення.

Файл із шаблоном розміщується наступним шляхом: /project/layouts/default.php. Відповідно до правил фреймворку, у цьому файлі доступна змінна $content. У тому місці, де буде виведена ця змінна та відбудеться вставка контенту сторінки.

За промовчанням цей файл містить наступний найпростіший код:

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<?= $content ?>

</body>

</html>

№1⊗ppOpUFmLy

Розмістіть у файлі з шаблоном такий макет сайту:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>тайтл сторінки</title>

</head>

<body>

<header>

хедер сайту

</header>

<div class="container">

<aside class="sidebar left">

лівий сайдбар

</aside>

<main>

<?= $content ?>

</main>

<aside class="sidebar right">

правий сайдбар

</aside>

</div>

<footer>

футер сайту

</footer>

</body>

</html>

Зайдіть на будь-яку дію будь-якого контролера. Подивіться, що змінилося.

**Тайтл сторінки**

У файлі шаблону також доступна змінна $title, що містить тайтл сторінки. Очевидно, що цей заголовок також буде різним для різних сторінок. Давайте використовуємо цю змінну за призначенням:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title><?= $title ?></title> <!-- виводимо тайтл сторінки -->

</head>

<body>

...

</body>

</html>

Для того, щоб задати тайтл для певного уявлення, необхідно в контролері записати його у властивість title:

<?php

namespace Project\Controllers;

use Core\Controller;

classTestController extends Controller

{

public functionact()

{

// Задамо тайтл:

$this->title = 'Дія act контролера test';

// Отрендері уявлення, передавши якісь дані:

return $this->render('test/act', [

'var1' => 'eee',

'var2' => 'bbb',

'var3' => 'kkk',

]);

}

}

?>

**Ресурси в MVC на PHP**

Під час розробки сайту потрібні деякі ресурси, доступні за прямим посиланням. Це CSS файли, JavaScript файли, картинки та інше. Для таких речей у фреймворку існує папка /project/webroot/. Все розміщене в цій папці буде доступне за прямим посиланням.

Давайте, наприклад, підключимо до нашого сайту файл CSS. Нехай ім'я файлу styles.css. Розмістимо цей файл у папці webroot і підключимо так:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<linkrel="stylesheet" href="/project/webroot/styles.css">

<title><?= $title ?></title>

</head>

<body>

...

</body>

</html>

**Моделі MVC на PHP**

Давайте тепер розберемося із моделями. Моделі також, як і контролери, є ООП класи. Їх слід розміщувати у папці /project/models/. Давайте в цій папці у файлі Page.php зробимо модель Page, що відповідає за дані сторінок нашого сайту:

<?php

namespace Project\Models;

use \Core\Model;

classPage extends Model

{

}

?>

Як ви бачите, наша модель успадковує від класу Core Model. Від цього класу наша модель успадковує захищені методи findOne та findMany. Метод findOne параметр приймає SQL запит і повертає перший запис з БД, що потрапила в результат запиту. Метод findMany також приймає запит SQL, але повертає масив записів.

При цьому вам не потрібно писати код, що встановлює з'єднання з базою даних, а також обробляти результати запиту через mysqli\_fetch\_assoc. Фреймворк робить це за вас. Вам просто потрібно вибрати, чи очікуєте ви від запиту один запис з БД або кілька і використати відповідну функцію, отримавши результатом осудний масив, яким можна просто взяти і скористатися далі.

Спробуємо практично

Нехай у вашій базі даних є таблиця page з записами. Давайте зробимо модель Page з двома способами. Перший метод отримуватиме один запис за її id, а другий метод - отримувати масив записів з діапазону id:

<?php

namespace Project\Models;

use \Core\Model;

class Page extends Model

{

public functiongetById($id)

{

return $this->findOne("SELECT \* FROM page WHERE id=$id");

}

public functiongetByRange($from, $to)

{

return $this->findMany("SELECT \* FROM page WHERE id>=$from AND id<=$to");

}

}

?>

Давайте тепер попрацюємо з нашою моделлю у контролері:

<?php

namespace Project\Controllers;

use \Core\Controller;

use \Project\Models\Page; //підключаємо нашу модель

classPageController extends Controller

{

public functiontest() {

$page=new Page; // створюємо об'єкт моделі

$data = $page->getById(3); // Отримаємо запис з id=3

var\_dump($data);

$data = $page->getById(5); // Отримаємо запис з id=5

var\_dump($data);

$data = $page->getByRange(2, 5); // записи з id від 2 до 5

var\_dump($data);

}

}

?>

**Застосування MVC на PHP**

Погляньмо на спільне застосування всього вивченого. Нехай, наприклад, у нас на сайті є якісь статті-сторінки. Ми хочемо показати одну статтю, або показати всі статті. Давайте розглянемо, як це зробити у нашому фреймворку

**Таблиця у БД**

Для початку створимо таблицю pages у базі даних. Додамо туди колонки id, title та text. Заповнимо цю таблицю якими-небудь даними через PhpMyAdmin.

Модель

Зробимо модель з методом для отримання одного запису та методом для отримання всіх записів:

<?php

namespace Project\Models;

use \Core\Model;

class Page extends Model

{

public functiongetById($id)

{

return $this->findOne("SELECT \* FROM pages WHERE id=$id");

}

public functiongetAll()

{

return $this->findMany("SELECT id, title FROM pages");

}

}

?>

**Роутінг**

Зробимо відповідний роутинг:

<?php

use \Core\Route;

return [

newRoute('/page/:id', 'page', 'one'),

newRoute('/pages/', 'page', 'all'),

];

?>

Контролер

Зробимо контролер:

<?php

namespace Project\Controllers;

use \Core\Controller;

use \Project\Models\Page;

classPageController extends Controller

{

public function one($params)

{

$page = (new Page) ->getById($params['id']);

$this->title = $page['title'];

return $this->render('page/one', [

'text' => $page['text'],

'h1' => $this->title

]);

}

public functionall()

{

$this->title = 'Список усіх сторінок';

$pages = (new Page) ->getAll();

return $this->render('page/all', [

'pages' => $pages,

'h1' => $this->title

]);

}

}

?>

Уявлення

Зробимо уявлення для методу one:

<h1><?= $h1; ?></h1>

<div id="content">

<?= $text; ?>

</div>

Зробимо уявлення для методу all:

<h1><?= $h1; ?></h1>

<div id="content">

<table>

<tr>

<th>id</th>

<th>title</th>

<th>посилання</th>

</tr>

<?php foreach ($pages as $page): ?>

<tr>

<td><?= $page['id']; ?></td>

<td><?= $page['title']; ?></td>

<td><ahref="/page/<?= $page['id']; ?>/">посилання на сторінку</td>

</tr>

<?php endforeach; ?>

</table>

</div>

**Постановка задачі.**

Розробити простий веб-застосунок на основі паттерну MVC у якому є хоча б одна модель даних та організовувати CRUD операції над цією моделью.

У застосунку потрібно реалізувати доступ до БД та маршрутизації.

Варіанти:

1. **Портфоліо:**
   * Ваш досвід та роботи.
   * Проекти, якими ви пишаєтесь.
2. **Особистий блог:**
   * Розкажіть про свої думки, інтереси та пригоди.
3. **Онлайн-резюме:**
   * Ваш досвід роботи та освіта.
   * Навички та досягнення.
4. **Подія або весільний сайт:**
   * Інформація про захід.
   * Фотогалерея.
5. **Продукт чи послуга:**
   * Опис товару чи послуги.
6. **Особиста історія успіху:**
   * Ваш шлях до досягнення мети.
   * Надихаючі моменти.
7. **Освітній ресурс:**
   * Уроки чи поради з конкретної теми.
8. **Командний проект:**
   * Проект, у якому ви брали участь у команді.
   * Результати та досягнення.
9. **Фотогалерея подорожей:**
   * Ваші фотографії із різних місць.
   * Короткі розповіді про пригоди.
10. **Інтерактивний портфель дизайнера:**
    * Показ роботи у цікавій формі.
11. **Сайт для благодійності:**
    * Опис мети та завдань організації.
    * Можливість зробити пожертву.
12. **Сайт для заходу чи конференції:**
    * Розклад заходів.
13. **Сайт-візитка компанії:**
    * Коротка інформація про компанію.
    * Контактні дані.
14. **Кулінарний блог:**
    * Рецепти, фотографії страв.
    * Поради щодо приготування.
15. **Технологічний стартап:**
    * Опис ідеї.
    * Переваги та цілі проекту.

**Результати:**

Звіт має містити:

1. Тема та мета роботи
2. Короткі теоретичні відомості
3. Результати створеного веб-застосунку в вигляді скріншотів або тестів
4. Лістинг вихідного коду проєкту
5. Висновки