

ТЕМА 9

Патерн MVC "з нуля"

*Front Controller · маршрутизація · Model · View · Controller ·
автозавантаження класів*

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 9 з 18

7 питань сьогоднішньої теми

01

Навіщо потрібна
архітектура

02

Патерн MVC

03

Front Controller і
маршрутизація

04

Модель (Model)

05

Представлення (View)

06

Контролер (Controller)

07

Структура проєкту й
автозавантаження

Проблема "спагеті-коду"

```
// products.php – усе в одному файлі
session_start();
if (!isset($_SESSION['user_id'])) { header('Location: /login.php'); exit; }

$pdo = new PDO('mysql:host=localhost;dbname=shop', 'user', 'pass');
$products = $pdo->query('SELECT * FROM products')->fetchAll();
?>
<h1>Товари</h1>
<?php foreach ($products as $p): ?>
    <p><?= $p['name'] ?> – <?= $p['price'] ?> грн</p>
<?php endforeach; ?>
```

Чотири відповідальності в одному файлі: авторизація, БД, дані, HTML — неможливо змінити одне, не ризикуючи іншим.

Розподіл відповідальності

Model

Дані та бізнес-логіка

Робота з базою даних, правила предметної області

View

Формування HTML

Представлення даних користувачу, форматування

Controller

Координація

Приймає запит, викликає Model, передає дані у View

Потік виконання запиту



Контролер — координатор: сам не містить ні SQL-запитів, ні HTML-розмітки.

Єдина точка входу: .htaccess

```
# .htaccess — усі запити на index.php,  
# якщо файлу чи теки з таким шляхом не існує  
RewriteEngine On  
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteCond %{REQUEST_FILENAME} !-d  
RewriteRule ^ index.php [QSA,L]
```

Усі HTTP-запити спрямовуються на один файл (index.php), який сам вирішує, який контролер обробить запит — на відміну від підходу "один файл — одна сторінка".

Найпростіший маршрутизатор

```
$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

$routes = [
    '/' => [HomeController::class, 'index'],
    '/products' => [ProductController::class, 'list'],
    '/products/create' => [ProductController::class, 'create'],
];

if (!isset($routes[$uri])) {
    http_response_code(404);
    exit('Сторінку не знайдено');
}

[$controllerClass, $method] = $routes[$uri];
$controller = new $controllerClass();
$controller->$method();
```

Дані та доступ до даних

```
class Product {
    public function __construct(
        public int $id,
        public string $name,
        public float $price,
    ) {}
}

class ProductRepository {
    public function __construct(private PDO $pdo) {}

    public function findAll(): array {
        $rows = $this->pdo->query('SELECT * FROM products')->fetchAll();
        return array_map(
            fn($r) => new Product($r['id'], $r['name'], $r['price']),
            $rows
        );
    }
}
```

Шаблон: лише HTML + вивід

```
<?php // views/products/list.php ?>
<!DOCTYPE html>
<html lang="uk">
<head><meta charset="UTF-8"><title>Товари</title></head>
<body>
  <h1>Каталог товарів</h1>
  <ul>
    <?php foreach ($products as $product): ?>
      <li>
        <?= htmlspecialchars($product->name) ?> –
        <?= number_format($product->price, 2) ?> грн
      </li>
    <?php endforeach; ?>
  </ul>
</body>
</html>
```

renderView(): ob_start() + extract()

```
function renderView(string $viewPath, array $data = []): string {  
    extract($data);    // ['products'=>$arr] -> змінна $products  
    ob_start();  
    require $viewPath;  
    return ob_get_clean();  
}  
  
$html = renderView(__DIR__ . '/views/products/list.php', [  
    'products' => $products,  
]);  
echo $html;
```

Шаблонізатори (Twig, Blade) додають зручний синтаксис і автоекранування, але принцип той самий.

Тонкий координаційний шар

```
class ProductController {
    private ProductRepository $repository;

    public function __construct() {
        $pdo = new PDO('mysql:host=localhost;dbname=shop', 'user', 'pass');
        $this->repository = new ProductRepository($pdo);
    }

    public function list(): void {
        $products = $this->repository->findAll();
        echo renderView(__DIR__ . '/../views/products/list.php', [
            'products' => $products,
        ]);
    }
}
```

Патерн Post/Redirect/Get

```
public function create(): void {  
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
        $this->repository->create($_POST['name'], (float) $_POST['price']);  
        header('Location: /products');  
        exit;  
    }  
    echo renderView(__DIR__ . '/../views/products/create.php');  
}
```

GET — показати порожню форму; POST — обробити й перенаправити (redirect). Це запобігає повторній відправці форми при оновленні сторінки.

Типова структура MVC-проєкту

```
project/
├── public/
│   ├── index.php          (Front Controller)
│   └── .htaccess
├── src/
│   ├── Controllers/
│   │   ├── HomeController.php
│   │   └── ProductController.php
│   ├── Models/
│   │   ├── Product.php
│   │   └── ProductRepository.php
│   └── views/
│       └── products/
│           ├── list.php
│           └── create.php
└── vendor/                (залежності Composer)
```

spl_autoload_register()

```
spl_autoload_register(function (string $class) {  
    // App\Models\Product -> src/Models/Product.php  
    $prefix = 'App\\';  
    $baseDir = __DIR__ . '/src/';  
  
    if (!str_starts_with($class, $prefix)) return;  
  
    $relative = substr($class, strlen($prefix));  
    $file = $baseDir . str_replace('\\\\', '/', $relative) . '.php';  
  
    if (file_exists($file)) {  
        require $file;  
    }  
});
```

Підсумок теми 9

- MVC розділяє застосунок на Model (дані), View (HTML), Controller (координація)
- Front Controller спрямовує всі запити через одну точку входу — `index.php`
- Router визначає, який контролер обробить конкретний URL
- Хороший контролер тонкий: логіка — у моделі, розмітка — у представленні
- Post/Redirect/Get запобігає повторній відправці форми
- `spl_autoload_register()` — основа автозавантаження, як у Laravel і Symfony