

ЛАБОРАТОРНА РОБОТА № 8

MVC-архітектура, пошук та автентифікація/авторизація

Мета роботи: навчитися реорганізовувати наявний PHP-застосунок за архітектурним патерном MVC (Model-View-Controller), реалізовувати функцію пошуку для сутностей предметної області, а також впроваджувати базову автентифікацію та авторизацію користувачів на основі сесій.

Теоретичні відомості

1. Архітектурний патерн MVC

MVC (Model-View-Controller) — це архітектурний патерн, що розділяє застосунок на три взаємопов'язані, але незалежні компоненти. Мета такого розділення — зменшити зв'язність (coupling) між бізнес-логікою, поданням даних і обробкою запитів, що робить код більш тестованим, повторно використовуваним і зручним для командної розробки.

Model (Модель) — відповідає за дані та бізнес-логіку: сутності предметної області, правила валідації, взаємодію з базою даних через репозиторії. Модель нічого не знає про HTTP-запити чи HTML.

View (Подання) — відповідає виключно за візуальне представлення даних користувачу (HTML-розмітка). Подання отримує вже готові дані від контролера і не містить бізнес-логіки, лише мінімальні умови для відображення (наприклад, чи показувати форму додавання).

Controller (Контролер) — приймає HTTP-запит, викликає потрібні методи моделі/репозиторію, готує дані для подання та повертає результат (рендер сторінки або редирект). Контролер — це «диспетчер», який зв'язує модель і подання.

2. Front Controller та маршрутизація

У попередніх лабораторних роботах кожен PHP-файл був самостійною точкою входу (category_form.php, backpack_form.php тощо). У MVC-застосунку натомість використовується патерн Front Controller: усі запити проходять через єдину точку входу (зазвичай public/index.php), а спеціальний компонент — маршрутизатор (Router) — визначає, який контролер і метод повинні обробити конкретний запит, виходячи з URL та HTTP-методу.

Такий підхід дозволяє мати «красиві» URL-адреси (наприклад, /backpacks/5 замість backpack_view.php?id=5), централізовано підключати спільну логіку (автозавантаження класів, старт сесії, обробку помилок) та легко розширювати список маршрутів без створення нових фізичних файлів.

3. Автозавантаження класів (spl_autoload_register)

У міру зростання кількості класів (моделі, репозиторії, контролери) вручну підключати кожен файл через require стає незручно. Функція spl_autoload_register() дозволяє зареєструвати власну функцію автозавантаження, яка викликається автоматично, коли PHP зустрічає ім'я ще не підключеного класу. Поєднуючи простір імен (namespace) класу зі структурою тек проекту, можна реалізувати просту PSR-4-подібну систему автозавантаження без сторонніх бібліотек.

4. Функція пошуку

Пошук — одна з базових функцій будь-якого каталогу даних. На рівні бази даних пошук за частковим збігом рядка зазвичай реалізується оператором SQL LIKE із символами-масками % (наприклад,

WHERE name LIKE '%рюкзак%'). Параметр пошуку передається через GET-запит (наприклад, /backpacks?q=трекінг) і не повинен підставлятися в запит напряму — обов'язково використовуються підготовлені вирази (prepared statements), щоб уникнути SQL-ін'єкцій.

5. Автентифікація та авторизація

Автентифікація — це процес підтвердження особи користувача (відповідь на питання «хто ви?»), зазвичай через пару логін/пароль. **Авторизація** — це процес перевірки прав доступу вже автентифікованого користувача (відповідь на питання «що вам дозволено?»).

Паролі користувачів ніколи не зберігаються у відкритому вигляді. Для їх безпечного зберігання PHP надає функції password_hash() (створює криптографічно стійкий хеш із вбудованою «сіллю») та password_verify() (перевіряє введений пароль проти збереженого хешу). Стан автентифікованого користувача між запитамі зберігається в сесії (\$_SESSION). Після успішного входу рекомендується викликати session_regenerate_id(), щоб запобігти атакам фіксації сесії (session fixation).

Авторизація на рівні застосунку найчастіше реалізується як перевірка «чи є користувач у сесії» перед виконанням дії, що змінює дані (створення, редагування, видалення), тоді як перегляд даних може залишатися доступним і неавтентифікованим відвідувачам.

Матеріал цієї роботи спирається на приклад із лабораторних робіт № 3–7 (предметна область «спортивні рюкзаки»: категорії, рюкзаки, характеристики) і продовжує розвивати той самий код, реорганізовуючи його за архітектурою MVC, додаючи пошук та автентифікацію/авторизацію на основі сутності User.

Приклад виконання

Структура проєкту

Після реорганізації за MVC-архітектурою проєкт має таку структуру тек і файлів:

```
project/
├── public/
│   ├── index.php          (Front Controller – єдина точка входу)
│   └── uploads/           (фотографії рюкзаків, без змін)
├── src/
│   ├── Router.php
│   ├── Auth.php
│   ├── Controllers/
│   │   ├── Controller.php    (абстрактний базовий контролер)
│   │   ├── BackpackController.php
│   │   ├── CategoryController.php
│   │   └── AuthController.php
│   ├── Models/
│   │   ├── Category.php      (без змін від лаб. 7)
│   │   ├── Backpack.php      (без змін від лаб. 7)
│   │   ├── Characteristic.php (без змін від лаб. 7)
│   │   ├── User.php          (новий клас)
│   │   └── Repositories/
│   │       ├── BackpackRepository.php (додано пошук)
│   │       └── UserRepository.php     (новий клас)
│   └── Views/
│       ├── backpacks/
│       │   ├── index.php
│       │   └── form.php
│       └── categories/
│           └── index.php
```

```

├── auth/
│   ├── login.php
│   └── register.php
├── database/
│   └── schema.sql
└── bootstrap.php

```

(додано таблицю users)
(PDO-підключення + автозавантаження, без змін логіки з'єднання)

1. Оновлена схема бази даних

До схеми з лабораторної роботи № 7 додається таблиця users для зберігання облікових записів. Таблиці категорій, рюкзаків і характеристик залишаються без змін.

```

-- database/schema.sql (фрагмент, що додається)

CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    login VARCHAR(50) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

```

2. bootstrap.php

Файл ініціалізації доповнюється реєстрацією автозавантажувача класів за простором імен App\ та стартом сесії, необхідної для роботи автентифікації.

```

<?php
// bootstrap.php
declare(strict_types=1);

session_start();

spl_autoload_register(function (string $class) {
    // App\Models\User -> src/Models/User.php
    $prefix = 'App\\';
    if (strncmp($prefix, $class, strlen($prefix)) !== 0) {
        return;
    }
    $relative = substr($class, strlen($prefix));
    $path = __DIR__ . '/src/' . str_replace('\\', '/', $relative) . '.php';
    if (is_file($path)) {
        require $path;
    }
});

function db(): PDO
{
    static $pdo = null;
    if ($pdo === null) {
        $pdo = new PDO(
            'sqlite:' . __DIR__ . '/database/backpacks.db',
            null, null,
            [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
        );
    }
    return $pdo;
}

```

3. Router.php — простий маршрутизатор

Маршрутизатор зіставляє метод запиту та шлях із зареєстрованим списком маршрутів. Шаблони маршрутів можуть містити параметри у фігурних дужках (наприклад, {id}), які вилучаються за допомогою регулярного виразу.

```
<?php
// src/Router.php
namespace App;

class Router
{
    private array $routes = [];

    public function add(string $method, string $pattern, callable $handler): void
    {
        $this->routes[] = [
            'method' => strtolower($method),
            'pattern' => $pattern,
            'handler' => $handler,
        ];
    }

    public function dispatch(string $method, string $uri): void
    {
        $path = parse_url($uri, PHP_URL_PATH) ?? '/';
        foreach ($this->routes as $route) {
            if ($route['method'] !== strtolower($method)) {
                continue;
            }
            $regex = preg_replace('#\{([a-zA-Z_+])\}#', '(?P<$1>[^/]+)', $route['pattern']);
            $regex = '#^' . $regex . '$#';
            if (preg_match($regex, $path, $matches)) {
                $params = array_filter(
                    $matches,
                    fn($k) => !is_int($k),
                    ARRAY_FILTER_USE_KEY
                );
                call_user_func($route['handler'], $params);
                return;
            }
        }
        http_response_code(404);
        echo '404 Not Found';
    }
}
```

4. public/index.php — Front Controller

Усі запити спрямовуються на цей файл (наприклад, через .htaccess із mod_rewrite або вбудований сервер php -S localhost:8000 public/index.php). Тут реєструються всі маршрути застосунку.

```
<?php
// public/index.php
require __DIR__ . '/../bootstrap.php';

use App\Router;
use App\Controllers\BackpackController;
use App\Controllers\CategoryController;
use App\Controllers\AuthController;

$backpackController = new BackpackController();
$categoryController = new CategoryController();
$authController     = new AuthController();

$router = new Router();
```

```

$router->add('GET', '/', [$backpackController, 'index']);
$router->add('GET', '/backpacks', [$backpackController, 'index']);
$router->add('POST', '/backpacks', [$backpackController, 'store']);
$router->add('POST', '/backpacks/{id}/photo', [$backpackController, 'uploadPhoto']);

$router->add('GET', '/categories', [$categoryController, 'index']);
$router->add('POST', '/categories', [$categoryController, 'store']);

$router->add('GET', '/login', [$authController, 'showLogin']);
$router->add('POST', '/login', [$authController, 'login']);
$router->add('GET', '/register', [$authController, 'showRegister']);
$router->add('POST', '/register', [$authController, 'register']);
$router->add('POST', '/logout', [$authController, 'logout']);

$router->dispatch($_SERVER['REQUEST_METHOD'], $_SERVER['REQUEST_URI']);

```

5. Controller.php — абстрактний базовий контролер

Базовий контролер винесено спільну логіку рендерингу подань і редиректів, щоб не дублювати її в кожному конкретному контролері.

```

<?php
// src/Controllers/Controller.php
namespace App\Controllers;

abstract class Controller
{
    protected function render(string $view, array $params = []): void
    {
        extract($params);
        require __DIR__ . '/../Views/' . $view . '.php';
    }

    protected function redirect(string $path): void
    {
        header('Location: ' . $path);
        exit;
    }
}

```

6. Auth.php — сервіс автентифікації

Клас Auth інкапсулює всю роботу із сесією користувача: спробу входу, перевірку стану автентифікації, примусову вимогу авторизації та вихід.

```

<?php
// src/Auth.php
namespace App;

use App\Models\Repositories\UserRepository;

class Auth
{
    public static function attempt(string $login, string $password): bool
    {
        $repo = new UserRepository();
        $user = $repo->findByLogin($login);
        if ($user === null || !password_verify($password, $user->passwordHash)) {
            return false;
        }
        session_regenerate_id(true);
        $_SESSION['user_id'] = $user->id;
    }
}

```

```

        $_SESSION['login'] = $user->login;
        return true;
    }

    public static function check(): bool
    {
        return isset($_SESSION['user_id']);
    }

    public static function requireLogin(): void
    {
        if (!self::check()) {
            header('Location: /login');
            exit;
        }
    }

    public static function logout(): void
    {
        $_SESSION = [];
        session_destroy();
    }
}

```

7. User.php та UserRepository.php

Сутність User та відповідний репозиторій побудовані за тим самим принципом, що й Category/Backpack/Characteristic у лабораторній роботі № 7 — з тією різницею, що пароль зберігається виключно у вигляді хешу.

```

<?php
// src/Models/User.php
namespace App\Models;

class User
{
    public function __construct(
        public readonly ?int $id,
        public readonly string $login,
        public readonly string $passwordHash,
    ) {}
}

```

```

<?php
// src/Models/Repositories/UserRepository.php
namespace App\Models\Repositories;

use App\Models\User;

class UserRepository
{
    public function findByLogin(string $login): ?User
    {
        $stmt = db()->prepare('SELECT * FROM users WHERE login = :login');
        $stmt->execute(['login' => $login]);
        $row = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($row === false) {
            return null;
        }
        return new User((int) $row['id'], $row['login'], $row['password_hash']);
    }

    public function create(string $login, string $plainPassword): User

```

```

    {
        $hash = password_hash($plainPassword, PASSWORD_DEFAULT);
        $stmt = db()->prepare(
            'INSERT INTO users (login, password_hash) VALUES (:login, :hash)'
        );
        $stmt->execute(['login' => $login, 'hash' => $hash]);
        return new User((int) db()->lastInsertId(), $login, $hash);
    }
}

```

8. Пошук у BackpackRepository

До репозиторію рюкзаків (перенесеного без змін логіки з лабораторної роботи № 7) додаються методи пошуку, що використовують оператор LIKE із параметром, переданим через підготовлений вираз.

```

<?php
// фрагмент src/Models/Repositories/BackpackRepository.php

public function searchCategories(string $query): array
{
    $stmt = db()->prepare(
        'SELECT * FROM categories WHERE name LIKE :q ORDER BY name'
    );
    $stmt->execute(['q' => '%' . $query . '%']);
    return array_map(
        fn($row) => $this->hydrateCategory($row),
        $stmt->fetchAll(PDO::FETCH_ASSOC)
    );
}

public function searchBackpacks(string $query): array
{
    $stmt = db()->prepare(
        'SELECT * FROM backpacks WHERE name LIKE :q ORDER BY name'
    );
    $stmt->execute(['q' => '%' . $query . '%']);
    return array_map(
        fn($row) => $this->hydrateBackpack($row),
        $stmt->fetchAll(PDO::FETCH_ASSOC)
    );
}

```

9. BackpackController.php

Контролер поєднує читання параметра пошуку з рядка запиту, перевірку прав доступу через Auth::requireLogin() перед діями, що змінюють дані, та рендеринг відповідного подання.

```

<?php
// src/Controllers/BackpackController.php
namespace App\Controllers;

use App\Auth;
use App\Models\Repositories\BackpackRepository;

class BackpackController extends Controller
{
    private BackpackRepository $repo;

    public function __construct()
    {
        $this->repo = new BackpackRepository();
    }
}

```

```

public function index(array $params): void
{
    $q = $_GET['q'] ?? '';
    $backpacks = $q !== ''
        ? $this->repo->searchBackpacks($q)
        : $this->repo->findAll();

    $this->render('backpacks/index', [
        'backpacks' => $backpacks,
        'query'      => $q,
        'isLoggedIn' => Auth::check(),
    ]);
}

public function store(array $params): void
{
    Auth::requireLogin();
    $this->repo->create(
        (int) $_POST['category_id'],
        $_POST['name'],
        (float) $_POST['price']
    );
    $this->redirect('/backpacks');
}

public function uploadPhoto(array $params): void
{
    Auth::requireLogin();
    $id = (int) $params['id'];
    // логіка перевірки MIME-типу та збереження файлу – без змін від лаб. 5
    $this->repo->attachPhoto($id, $_FILES['photo']);
    $this->redirect('/backpacks');
}
}

```

10. Подання src/Views/backpacks/index.php

Подання відповідає лише за розмітку: воно виводить навігацію з умовним посиланням «Вхід» або «Вийти» залежно від isLoggedIn, форму пошуку та форму додавання, яка показується тільки автентифікованим користувачам.

```

<!-- src/Views/backpacks/index.php -->
<!doctype html>
<html lang="uk">
<head><meta charset="utf-8"><title>Рюкзаки</title></head>
<body>
    <nav>
        <a href="/backpacks">Рюкзаки</a> | <a href="/categories">Категорії</a>
        <?php if ($isLoggedIn): ?>
            <form method="post" action="/logout" style="display:inline">
                <button type="submit">Вийти</button>
            </form>
        <?php else: ?>
            <a href="/login">Вхід</a>
        <?php endif; ?>
    </nav>

    <form method="get" action="/backpacks">
        <input type="text" name="q" value="<?= htmlspecialchars($query) ?>"
            placeholder="Пошук за назвою...">
        <button type="submit">Знайти</button>
    </form>

    <ul>
        <?php foreach ($backpacks as $b): ?>

```

```

        <li><?= htmlspecialchars($b->name) ?> – <?= $b->price ?> грн</li>
    <?php endforeach; ?>
</ul>

<?php if ($isLoggedIn): ?>
    <form method="post" action="/backpacks">
        <!-- поля category_id, name, price -->
        <button type="submit">Додати рюкзак</button>
    </form>
<?php endif; ?>
</body>
</html>

```

11. AuthController.php та форма входу

Контролер автентифікації відповідає за показ форм входу/реєстрації та обробку відповідних POST-запитів за принципом Post/Redirect/Get.

```

<?php
// src/Controllers/AuthController.php
namespace App\Controllers;

use App\Auth;
use App\Models\Repositories\UserRepository;

class AuthController extends Controller
{
    public function showLogin(array $params): void
    {
        $this->render('auth/login', ['error' => null]);
    }

    public function login(array $params): void
    {
        $ok = Auth::attempt($_POST['login'], $_POST['password']);
        if (!$ok) {
            $this->render('auth/login', ['error' => 'Невірний логін або пароль']);
            return;
        }
        $this->redirect('/backpacks');
    }

    public function showRegister(array $params): void
    {
        $this->render('auth/register', ['error' => null]);
    }

    public function register(array $params): void
    {
        $repo = new UserRepository();
        if ($repo->findByLogin($_POST['login']) != null) {
            $this->render('auth/register', ['error' => 'Логін вже зайнятий']);
            return;
        }
        $repo->create($_POST['login'], $_POST['password']);
        $this->redirect('/login');
    }

    public function logout(array $params): void
    {
        Auth::logout();
        $this->redirect('/backpacks');
    }
}

```

```

<!-- src/Views/auth/login.php -->
<!doctype html>
<html lang="uk">
<head><meta charset="utf-8"><title>Вхід</title></head>
<body>
  <h1>Вхід</h1>
  <?php if ($error): ?><p style="color:red"><?= htmlspecialchars($error) ?></p><?php endif;
?>
  <form method="post" action="/login">
    <input type="text" name="login" placeholder="Логін" required>
    <input type="password" name="password" placeholder="Пароль" required>
    <button type="submit">Увійти</button>
  </form>
  <p><a href="/register">Зареєструватися</a></p>
</body>
</html>

```

Що залишилося без змін порівняно з лабораторною роботою № 7: підключення до бази даних (PDO), структура таблиць categories/backpacks/characteristics, методи знаходження/створення сутностей, логіка завантаження та перевірки фотографій. Реорганізація за MVC змінює лише те, «де» розташований код і «як» викликаються ці методи — сама бізнес-логіка перенесена практично без змін.

Завдання на лабораторну роботу

1. Реорганізуйте проєкт із лабораторної роботи № 7 за MVC-архітектурою: винесіть спільну логіку в базовий контролер, створіть Front Controller (єдину точку входу) та маршрутизатор.
2. Додайте функцію пошуку для кожної сутності предметної області (за назвою або іншим текстовим полем), реалізавши її через SQL LIKE з підготовленими виразами та параметр запиту в URL.
3. Реалізуйте автентифікацію та авторизацію: додайте сутність User і відповідну таблицю в базі даних, форми реєстрації та входу, збереження стану сесії, а також обмежте дії, що змінюють дані (додавання/редагування/видалення), лише автентифікованим користувачам.
4. Переконайтеся, що паролі зберігаються виключно у вигляді хешу (password_hash/password_verify), а не у відкритому вигляді.
5. Перевірте, що весь функціонал, реалізований у попередніх лабораторних роботах (перегляд, додавання, завантаження фотографій), продовжує коректно працювати після реорганізації.
6. Підготуйте короткий звіт зі скриншотами роботи застосунку (форма входу, результат пошуку, сторінка зі списком рюкзаків для автентифікованого та неавтентифікованого користувача).

Контрольні запитання

1. У чому полягає призначення кожного з трьох компонентів патерну MVC?
2. Що таке Front Controller і навіщо він потрібен у MVC-застосунку?
3. Як працює spl_autoload_register() і чим він зручніший за ручне підключення файлів через require?
4. Чому пошук за LIKE слід реалізовувати через підготовлений вираз, а не шляхом конкатенації рядка запиту?
5. Чим відрізняються поняття автентифікації та авторизації?

6. Навіщо потрібна функція `session_regenerate_id()` після успішного входу користувача?
7. Чому паролі не можна зберігати у відкритому вигляді або з використанням оборотного шифрування, а лише як хеш?