

ЛАБОРАТОРНА РОБОТА № 4

Модульна структура програми, зберігання даних у CSV та REST-подібний JSON API

Мета роботи: Навчитися розбивати програму на окремі логічно завершені файли та підключати їх засобами `require/include`; організувати зберігання й редагування даних програми у форматі CSV; реалізувати найпростіший JSON API на основі вже написаної логіки; побудувати два незалежні клієнти цього API — на JavaScript (у браузері) та на PHP (через cURL).

Теоретичні відомості

1. Підключення файлів: `include`, `require`, `require_once`

Коли програма розростається (тема 9), розміщувати всі класи в одному файлі стає незручно: файл важко читати, а логічно не пов'язані частини коду переплітаються. PHP дозволяє винести кожен клас чи логічний блок в окремий файл і підключати його інструкціями `include`, `require` або їхніми варіантами з `_once`.

Інструкція	Поведінка при відсутності файлу
<code>include</code>	Попередження (Warning), виконання скрипта триває далі
<code>require</code>	Фатальна помилка (Fatal Error), виконання скрипта зупиняється
<code>include_once</code>	Як <code>include</code> , але файл підключається не більше одного разу за весь запит
<code>require_once</code>	Як <code>require</code> , але файл підключається не більше одного разу за весь запит

Для підключення класів майже завжди використовують саме `require_once`: клас, від якого залежить решта програми, дійсно необхідний (тому `require`, а не `include`, який лише попереджає й продовжує роботу без потрібного класу), а повторне підключення того самого файлу класу спричинило б фатальну помилку «Cannot redeclare class» — тому `_once`.

2. Формат CSV: читання та запис

CSV (Comma-Separated Values) — простий текстовий формат для табличних даних: перший рядок зазвичай містить назви колонок (заголовки), кожен наступний рядок — один запис, значення розділені комою. PHP має вбудовані функції для роботи з CSV без ручного розбору рядків символ за символом.

Функція	Призначення
<code>fopen(\$file, \$mode)</code>	Відкриває файл; режим 'r' — читання, 'w' — запис (перезаписує вміст)
<code>fgetcsv(\$handle)</code>	Зчитує один рядок файлу й одразу повертає його як масив значень
<code>fputcsv(\$handle, \$row)</code>	Записує масив значень в один рядок файлу у форматі CSV

Функція	Призначення
<code>fclose(\$handle)</code>	Закриває файл, звільняючи ресурс

```
<?php
// Читання CSV-файлу
$handle = fopen('categories.csv', 'r');
$header = fgetcsv($handle); // перший рядок – заголовки колонок,
// зазвичай пропускається
while (($row = fgetcsv($handle)) !== false) {
    [$id, $name] = $row; // $row – звичайний індексований масив
    // значень
    echo "$id: $name\n";
}
fclose($handle);
```

Режим 'w' у `fopen()` повністю очищає наявний вміст файлу перед записом. Тому типовий підхід до редагування CSV — не дописувати окремий змінений рядок усередину файлу (що технічно складно для текстових форматів), а зчитати всі записи в пам'ять, змінити потрібний, і переписати файл повністю заново — так само, як це робить метод `save()` у прикладі цієї роботи.

3. Перетворення об'єктів у масив для JSON

Функція `json_encode()` (тема 5) вміє серіалізувати масиви й прості типи, але для об'єктів за замовчуванням серіалізує лише **public** властивості — а всі властивості наших класів **private** (тема 18, інкапсуляція). Прийнятий підхід — додати в клас метод `toArray()`, що вручну формує асоціативний масив із потрібними полями, і саме його передавати в `json_encode()`.

```
<?php
public function toArray(): array {
    return [
        'id' => $this->id,
        'name' => $this->name,
        'price' => $this->price,
    ];
}

echo json_encode($product->toArray(), JSON_UNESCAPED_UNICODE);
```

Прапорець `JSON_UNESCAPED_UNICODE` запобігає перетворенню кириличних символів на послідовності виду `\u0410\u0430...` — без нього текст лишається технічно коректним JSON, але нечитабельним для людини під час налагодження.

4. Найпростіший JSON API на PHP

Розглянуті в темі 10 принципи REST API застосовні й до найпростішого одного PHP-файлу: він приймає HTTP-запит (можливо, із параметром у рядку запиту `$_GET`), звертається до вже написаної бізнес-логіки (клас-репозиторій), і повертає результат у форматі JSON із відповідним заголовком **Content-Type** та кодом статусу (тема 10).

```

<?php
header('Content-Type: application/json; charset=utf-8');

$id = $_GET['id'] ?? null;

if ($id !== null) {
    $item = $repository->findById((int) $id);
    if ($item === null) {
        http_response_code(404);
        echo json_encode(['error' => 'Не знайдено']);
        exit;
    }
    echo json_encode($item->toArray(), JSON_UNESCAPED_UNICODE);
    exit;
}

// Без параметра id – повернути список усіх записів
echo json_encode(array_map(
    fn($item) => $item->toArray(),
    array_values($repository->findAll())
), JSON_UNESCAPED_UNICODE);

```

5. Клієнт на JavaScript: fetch()

Функція `fetch()` у браузері надсилає HTTP-запит до вказаної адреси (тут — до нашого власного PHP-файлу з JSON API) і повертає результат асинхронно, через об'єкт Promise. Це найпростіший спосіб продемонструвати, що розроблений API можна використовувати з будь-якого клієнта — не лише з PHP.

```

<script>
fetch('api.php')
    .then(response => response.json())
    .then(data => {
        console.log(data);    // масив об'єктів, отриманий з API
    })
    .catch(error => console.error('Помилка:', error));
</script>

```

6. Клієнт на PHP: cURL

cURL (тема 5) дозволяє одному PHP-скрипту звертатися до іншого сервісу через HTTP — у цьому випадку до власного JSON API, який працює як окремий, незалежний від клієнта, вебресурс. Це той самий принцип, що лежить в основі міжсервісної взаємодії (тема 13): клієнт не знає внутрішньої реалізації API, а лише звертається до нього за URL-адресою.

```

<?php
$ch = curl_init('http://localhost:8000/api.php');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
curl_close($ch);

$data = json_decode($response, true);
foreach ($data as $item) {
    echo $item['name'] . "\n";
}

```

Завдання на лабораторну роботу

Продовжуйте працювати з предметною областю, обраною в лабораторній роботі № 3 (власний варіант курсової роботи з «Організації баз даних»). Виконайте послідовно шість кроків:

1. Розбийте файл лабораторної роботи № 3 на окремі фрагменти та підключіть необхідні функції з використанням `require`, `include` або `require_once`. Кожен клас — в окремому файлі; створіть файл-«точку збору» (`bootstrap`), що підключає всі класи одним місцем.
2. Створіть файли, де розмістіть у форматі CSV дані, якими має оперувати програма. Мінімум два CSV-файли: один для пов'язаної (довідникової) сутності, другий — для головної сутності з посиланням на першу через ідентифікатор.
3. Організуйте читання з файлів та запис редагованої інформації назад у файл. Клас-репозиторій під час створення повинен зчитувати дані з CSV; кожна операція, що змінює дані (створення, оновлення, видалення), повинна одразу зберігати оновлений стан назад у CSV-файл.
4. Зробіть базову API, яка буде віддавати дані із файлів у вигляді JSON. Один PHP-файл, що повертає або список усіх записів, або один запис за ідентифікатором (переданим через `$_GET`), коректно встановлюючи заголовок `Content-Type` та код статусу для випадку «не знайдено».
5. Створіть окрему html сторінку-демонстрацію, де розробіть скрипт, який читатиме дані з розробленого JSON API. Використайте `fetch()` для звернення до `api.php` і виведіть отримані дані на сторінку (список, таблиця тощо).
6. Створіть окремий PHP документ-демонстрацію, що буде читати дані з JSON API через CURL запити. Використайте `curl_init()/curl_exec()` для звернення до того самого `api.php` з боку іншого PHP-скрипта, розберіть отриманий JSON і виведіть дані на екран.

Вимоги до звіту

Звіт повинен містити: тему й мету роботи; структуру файлів проєкту (дерево тек і файлів); лістинг коду кожного файлу; вміст створених CSV-файлів до й після демонстрації операції редагування; знімок екрана роботи `api.php` (вивід JSON у браузері); знімок екрана html-сторінки з даними, отриманими через `fetch()`; знімок екрана виводу PHP-скрипта, що використовує cURL; висновок.

Приклад виконання: «Спортивні рюкзаки»

Приклад продовжує предметну область із лабораторної роботи № 3 — тепер класи розподілено по файлах, а дані зберігаються в CSV замість оперативної пам'яті.

Структура проєкту

```

lab4/
├── classes/
│   ├── Category.php
│   ├── Characteristic.php
│   ├── Backpack.php
│   └── BackpackRepository.php
├── data/
│   ├── categories.csv
│   └── backpacks.csv
├── bootstrap.php      (підключає всі класи та створює репозиторій)
├── api.php             (JSON API)
├── demo.html           (клієнт на JavaScript / fetch)
└── demo_curl.php       (клієнт на PHP / cURL)

```

1. Класи сутностей (classes/Category.php, Characteristic.php)

```

<?php
// classes/Category.php
class Category {
    public function __construct(
        private int $id,
        private string $name
    ) {}

    public function getId(): int { return $this->id; }
    public function getName(): string { return $this->name; }

    public function __toString(): string {
        return $this->name;
    }
}

```

```

<?php
// classes/Characteristic.php
class Characteristic {
    public function __construct(
        private string $name,
        private string $value
    ) {}

    public function getName(): string { return $this->name; }
    public function getValue(): string { return $this->value; }

    public function __toString(): string {
        return "{$this->name}: {$this->value}";
    }
}

```

2. Клас Backpack із методом toArray() (classes/Backpack.php)

```

<?php
// classes/Backpack.php
class Backpack {
    private array $characteristics = [];

    public function __construct(
        private int $id,
        private string $name,
        private Category $category,
        private float $price,
        private float $volumeLiters
    ) {}

    public function getId(): int { return $this->id; }
    public function getName(): string { return $this->name; }
    public function getCategory(): Category { return $this->category; }
    public function getPrice(): float { return $this->price; }
    public function getVolumeLiters(): float { return $this-
>volumeLiters; }
    public function getCharacteristics(): array { return $this-
>characteristics; }

    public function setName(string $name): void { $this->name =
$name; }
    public function setCategory(Category $category): void { $this-
>category = $category; }
    public function setPrice(float $price): void { $this->price =
$price; }
    public function setVolumeLiters(float $v): void { $this-
>volumeLiters = $v; }

    public function addCharacteristic(Characteristic $c): void {
        $this->characteristics[] = $c;
    }

    // Для JSON API (тема 4 цієї роботи, розділ 3)
    public function toArray(): array {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'category' => $this->category->getName(),
            'price' => $this->price,
            'volumeLiters' => $this->volumeLiters,
            'characteristics' => array_map(
                fn(Characteristic $c) => ['name' => $c->getName(),
'value' => $c->getValue()],
                $this->characteristics
            ),
        ];
    }

    public function __toString(): string {
        $chars = implode(' ', array_map(fn($c) => (string) $c, $this-
>characteristics));
        return sprintf(
            '#%d %s [%s] – %.1f л, %.2f грн (%s)',
            $this->id, $this->name, $this->category, $this-
>volumeLiters, $this->price, $chars
        );
    }
}

```

```
    }  
};
```

3. Вміст CSV-файлів (data/)

Файл `data/categories.csv`:

```
id,name  
1,Туристичні  
2,Спортивні  
3,Тактичні
```

Файл `data/backpacks.csv` (характеристики зберігаються в одному полі, елементи розділені «|», назва й значення — «:»):

```
id,name,category_id,price,volume_liters,characteristics  
1,TrailMaster 40,1,2400.00,40,Матеріал:Нейлон Cordura|Вага:1.2 кг  
2,RunLight 15,2,1200.00,15,Матеріал:Поліестер|Вага:0.6 кг  
3,TacticalPro 30,3,3200.00,30,Матеріал:Кордура 1000D|MOLLE-система:Так
```

Зберігання списку характеристик як одного текстового поля з власним розділювачем («|» між характеристиками, «:» між назвою й значенням) — типовий компромісний прийом для плоского CSV-формату, коли одна сутність пов'язана з кількома значеннями іншої. У повноцінній базі даних (тема 7) для цього використали б окрему таблицю й JOIN; тут, без бази даних, це поле відіграє ту саму роль у спрощеному вигляді.

4. Читання й запис CSV у репозиторії (`classes/BackpackRepository.php`)

```

<?php
// classes/BackpackRepository.php
class BackpackRepository {
    private array $backpacks = [];
    private array $categories = [];
    private int $nextId = 1;

    public function __construct(
        private string $categoriesFile,
        private string $backpacksFile
    ) {
        $this->loadCategories();
        $this->loadBackpacks();
    }

    private function loadCategories(): void {
        $handle = fopen($this->categoriesFile, 'r');
        fgetcsv($handle); // пропустить заголовок
        while (($row = fgetcsv($handle)) !== false) {
            [$id, $name] = $row;
            $this->categories[(int) $id] = new Category((int) $id,
$name);
        }
        fclose($handle);
    }

    private function loadBackpacks(): void {
        $handle = fopen($this->backpacksFile, 'r');
        fgetcsv($handle);
        while (($row = fgetcsv($handle)) !== false) {
            [$id, $name, $categoryId, $price, $volume, $charsRaw] =
$row;

            $backpack = new Backpack(
                (int) $id, $name, $this->categories[(int) $categoryId],
                (float) $price, (float) $volume
            );
            if ($charsRaw !== '') {
                foreach (explode('|', $charsRaw) as $pair) {
                    [$cName, $cValue] = explode(':', $pair, 2);
                    $backpack->addCharacteristic(new
Characteristic($cName, $cValue));
                }
            }
            $this->backpacks[(int) $id] = $backpack;
            $this->nextId = max($this->nextId, (int) $id + 1);
        }
        fclose($handle);
    }

    public function getCategories(): array { return $this-
>categories; }
    public function findAll(): array { return $this->backpacks; }
    public function findById(int $id): ?Backpack { return $this-
>backpacks[$id] ?? null; }

    public function create(string $name, Category $category, float
$price, float $volume): Backpack {

```

```

        $backpack = new Backpack($this->nextId++, $name, $category,
$price, $volume);
        $this->backpacks[$backpack->getId()] = $backpack;
        $this->save();
        return $backpack;
    }

    public function update(int $id, array $data): bool {
        $backpack = $this->findById($id);
        if ($backpack === null) return false;
        if (isset($data['name'])) $backpack->setName($data['name']);
        if (isset($data['price'])) $backpack->setPrice((float)
$data['price']);
        if (isset($data['volumeLiters'])) $backpack-
>setVolumeLiters((float) $data['volumeLiters']);
        $this->save();
        return true;
    }

    public function delete(int $id): bool {
        if (!isset($this->backpacks[$id])) return false;
        unset($this->backpacks[$id]);
        $this->save();
        return true;
    }

    private function save(): void {
        $handle = fopen($this->backpacksFile, 'w');
        fputcsv($handle, ['id', 'name', 'category_id', 'price',
'volume_liters', 'characteristics']);
        foreach ($this->backpacks as $backpack) {
            $chars = implode('|', array_map(
                fn(Characteristic $c) => $c->getName() . ':' . $c-
>getValue(),
                $backpack->getCharacteristics()
            ));
            fputcsv($handle, [
                $backpack->getId(), $backpack->getName(),
                $backpack->getCategory()->getId(), $backpack-
>getPrice(),
                $backpack->getVolumeLiters(), $chars,
            ]);
        }
        fclose($handle);
    }
}

```

Методи `create()`, `update()` і `delete()` тепер викликають приватний метод `save()` наприкінці — на відміну від лабораторної роботи № 3, де зміни жили лише в оперативній пам'яті, тут кожна зміна одразу переписує `backpacks.csv` повністю, і дані залишаються доступними навіть після завершення роботи скрипта чи перезапуску сервера.

5. Підключення класів (`bootstrap.php`)

```
<?php
// bootstrap.php – єдина точка підключення всіх класів (крок 1
завдання)
require_once __DIR__ . '/classes/Category.php';
require_once __DIR__ . '/classes/Characteristic.php';
require_once __DIR__ . '/classes/Backpack.php';
require_once __DIR__ . '/classes/BackpackRepository.php';

$repository = new BackpackRepository(
    __DIR__ . '/data/categories.csv',
    __DIR__ . '/data/backpacks.csv'
);
```

Порядок require_once важливий: Backpack.php посилається на клас Category у сигнатурі конструктора, тому Category.php повинен підключатися раніше за Backpack.php. PHP не має вбудованого автозавантаження без додаткового налаштування (тема 9 розглядала spl_autoload_register() для більших проєктів) — для невеликої кількості файлів явний список require_once цілком достатній і зрозуміліший.

6. JSON API (api.php)

```
<?php
// api.php (крок 4 завдання)
require_once __DIR__ . '/bootstrap.php';

header('Content-Type: application/json; charset=utf-8');

$id = $_GET['id'] ?? null;

if ($id !== null) {
    $backpack = $repository->findById((int) $id);
    if ($backpack === null) {
        http_response_code(404);
        echo json_encode(['error' => 'Рюкзак не знайдено'],
JSON_UNESCAPED_UNICODE);
        exit;
    }
    echo json_encode($backpack->toArray(), JSON_UNESCAPED_UNICODE);
    exit;
}

$all = array_map(
    fn(Backpack $b) => $b->toArray(),
    array_values($repository->findAll())
);
echo json_encode($all, JSON_UNESCAPED_UNICODE);
```

Очікувана відповідь api.php (без параметрів)

```
[
  {
    "id": 1, "name": "TrailMaster 40", "category": "Туристичні",
    "price": 2400, "volumeLiters": 40,
    "characteristics": [
      {"name": "Матеріал", "value": "Нейлон Cordura"},
      {"name": "Вага", "value": "1.2 кг"}
    ]
  },
  { "id": 2, "name": "RunLight 15", ... },
  { "id": 3, "name": "TacticalPro 30", ... }
]
```

7. HTML-демонстрація на JavaScript (demo.html)

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Демонстрація JSON API – Рюкзаки</title>
</head>
<body>
  <h1>Каталог рюкзаків</h1>
  <ul id="list">Завантаження...</ul>

  <script>
    fetch('api.php')
      .then(response => response.json())
      .then(backpacks => {
        const list = document.getElementById('list');
        list.innerHTML = '';
        backpacks.forEach(b => {
          const li = document.createElement('li');
          li.textContent =
            `${b.name} [${b.category}] – ${b.volumeLiters}
л, ${b.price} грн`;
          list.appendChild(li);
        });
      })
      .catch(error => {
        document.getElementById('list').textContent = 'Помилка
завантаження даних';
        console.error(error);
      });
  </script>
</body>
</html>
```

8. PHP-демонстрація через cURL (demo_curl.php)

```

<?php
// demo_curl.php (крок 6 завдання) – окремий, незалежний клієнт API
$ch = curl_init('http://localhost:8000/api.php');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);

if (curl_errno($ch)) {
    echo 'Помилка cURL: ' . curl_error($ch);
    exit;
}
curl_close($ch);

$backpacks = json_decode($response, true);

echo "=== Каталог рюкзаків (отримано через cURL) ===\n";
foreach ($backpacks as $b) {
    printf(
        "%#d %s [%s] – %.1f л, %.2f грн\n",
        $b['id'], $b['name'], $b['category'], $b['volumeLiters'],
        $b['price']
    );
}

```

Очікуваний вивід demo_curl.php

```

=== Каталог рюкзаків (отримано через cURL) ===
#1 TrailMaster 40 [Туристичні] – 40.0 л, 2400.00 грн
#2 RunLight 15 [Спортивні] – 15.0 л, 1200.00 грн
#3 TacticalPro 30 [Тактичні] – 30.0 л, 3200.00 грн

```

Щоб demo_curl.php зміг звернутися до api.php через http://localhost:8000/, обидва файли повинні працювати під одним запущеним сервером (наприклад, php -S localhost:8000 у теці проєкту) — cURL надсилає справжній мережевий HTTP-запит, а не просто підключає інший файл, тому без запущеного сервера отримаємо помилку з'єднання (тема 13).

Контрольні запитання

- Чим require_once відрізняється від include за поведінкою при відсутності файлу й при повторному підключенні?
- Чому порядок require_once файлів класів у bootstrap.php має значення саме для Category.php і Backpack.php?
- Навіщо перед записом у CSV-файл (режим 'w') потрібно спершу зчитати весь наявний вміст у пам'ять?
- Для чого потрібен метод toArray(), якщо властивості класу вже зберігають усі необхідні дані?
- Який HTTP-заголовок і навіщо встановлює api.php перед виведенням JSON?
- Чим клієнт на fetch() (demo.html) принципово відрізняється від клієнта на cURL (demo_curl.php) з погляду того, де виконується код?

- Чому `demo_curl.php` не може просто `require_once api.php` замість звернення через `cURL` — у чому практична різниця цих двох підходів?
- Як API з цієї роботи (крок 4) співвідноситься з принципами REST, розглянутими в темі 10?