

ЛАБОРАТОРНА РОБОТА № 6

JSON API з опрацюванням `php://input` та SPA-фронтенд на AJAX

Мета роботи: Перепроєктувати серверну частину застосунку так, щоб кожен запит на створення запису (крім завантаження файлу) приймав дані у форматі JSON через тіло запиту (`php://input`), а не через `$_POST`; побудувати єдину SPA-сторінку, що взаємодіє з цим API виключно через AJAX (`fetch`), оновлюючи вміст сторінки без жодного повного перезавантаження. По суті — розробити повноцінний JSON API для вже наявної предметної області.

Теоретичні відомості

1. SPA та AJAX: що змінюється порівняно з SSR

У лабораторній роботі № 5 кожна дія користувача (додавання категорії, рюкзака) призводила до повного перезавантаження сторінки: браузер надсилав форму, отримував нову HTML-сторінку від сервера й повністю її перемальовував. SPA (Single Page Application) — підхід, за якого браузер завантажує HTML-сторінку лише один раз, а всі подальші дії (додавання запису, оновлення списку) виконуються *асинхронно* через **AJAX**-запити (Asynchronous JavaScript and XML — історична назва; на практиці дані майже завжди передаються в форматі JSON, а не XML), без перезавантаження сторінки.

Ознака	SSR (лаб. № 5)	SPA + AJAX (ця робота)
Що повертає сервер	Готовий HTML на кожен запит	Лише дані у форматі JSON
Формат даних форми	<code>\$_POST</code> (пари ключ=значення)	Тіло запиту у форматі JSON
Оновлення сторінки	Повне перезавантаження (PRG, тема 9)	Частковий перемальовок через JavaScript
Роль PHP-файлу	Формує й HTML, і логіку водночас	Лише API — віддає й приймає дані

Це логічне продовження лабораторної роботи № 4, де вже був JSON API та демонстраційна сторінка з `fetch()` — але дані від клієнта туди не надсилались, лише зчитувались. Ця робота замикає коло: тепер клієнт і надсилає, і отримує дані виключно через API, а PHP-файли, що безпосередньо формують HTML для користувача (як у лабораторній роботі № 5), більше не потрібні.

2. Читання JSON з тіла запиту: `php://input`

Коли клієнт надсилає запит із тілом у форматі JSON (а не як заповнену HTML-форму), PHP **не** заповнює масив `$_POST` — цей масив призначений лише для класичних форм (тема 6). Натомість необроблене тіло будь-якого запиту доступне через потік `php://input`, який зчитують функцією `file_get_contents()` і перетворюють на масив PHP функцією `json_decode()`.

```
<?php
$raw = file_get_contents('php://input'); // '{"name":"Спортивні"}'
$data = json_decode($raw, true);       // ['name' => 'Спортивні']

$name = trim($data['name'] ?? '');
```

json_decode(\$raw, true) із другим параметром true повертає звичайний асоціативний масив PHP (як і очікують наші методи репозиторію), а не об'єкт stdClass, який повернувся б за замовчуванням. Якщо тіло запиту порожнє чи містить некоректний JSON, json_decode() поверне null — тому доступ до полів завжди супроводжують оператором ?? з порожнім значенням за замовчуванням.

3. Надсилання JSON з клієнта: fetch() із тілом запиту

У лабораторній роботі № 4 fetch() використовувався лише для отримання даних (GET-запит без тіла). Щоб надіслати JSON-дані на сервер, запит повинен: використовувати метод POST, мати заголовок Content-Type: application/json (щоб сервер знав, як інтерпретувати php://input), і містити серіалізований рядок JSON у параметрі body.

```
<script>
const payload = { name: 'Спортивні' };

fetch('api.php?resource=categories', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(payload),
})
  .then(response => response.json())
  .then(created => console.log('Створено:', created));
</script>
```

4. Файли в SPA: чому вони не йдуть через JSON

Формат JSON — текстовий і не призначений для передавання довільних бінарних даних (вмісту зображення). Технічно вміст файлу можна закодувати в текст (Base64), але це збільшує розмір передавання приблизно на третину й ускладнює код без реальної потреби. Тому усталений підхід — надсилати файл окремим запитом у форматі multipart/form-data (тема 6, лабораторна робота № 5), через об'єкт FormData у JavaScript, а не через JSON.

```
<script>
const formData = new FormData();
formData.append('photo', fileInput.files[0]);

fetch('api.php?resource=backpacks&action=photo&id=5', {
  method: 'POST',
  body: formData, // без заголовка Content-Type – браузер виставить його сам,
}) // разом із межею (boundary) для multipart-повідомлення
```

Тому запис, що поєднує звичайні поля й файл (рюкзак і його фото), зберігається у два окремі запити: спершу JSON-запитом створюється сам запис (сервер повертає його `id`), а вже потім, окремим `multipart`-запитом із цим `id`, до нього прикріплюється файл. Це не обхідний прийом, а стандартна практика реальних API — так само влаштоване, наприклад, додавання вкладення до вже створеного запису в багатьох публічних API.

5. Проєктування маршрутів простого API

Принципи REST (тема 10) застосовуються й тут: кожен ресурс (категорії, рюкзаки, характеристики) має власний шлях, а дія визначається HTTP-методом, а не окремим "дієсловним" URL. У межах одного файлу `api.php` без додаткового налаштування переадресації URL (тема 9) шлях ресурсу зручно передавати через рядок запиту — параметр `resource`; такий підхід не змінює суті REST-маршрутизації, лише спрощує її технічну реалізацію для навчального проєкту.

Метод і шлях	Дія	Тіло запиту
GET <code>api.php?resource=categories</code>	Список категорій	—
POST <code>api.php?resource=categories</code>	Створити категорію	JSON: { name }
GET <code>api.php?resource=backpacks</code>	Список рюкзаків	—
POST <code>api.php?resource=backpacks</code>	Створити рюкзак (без фото)	JSON: { name, category_id, price, volume }
POST <code>api.php?resource=backpacks&action=photo&id=..</code>	Прикріпити фото до рюкзака	<code>multipart/form-data</code> : photo
POST <code>api.php?resource=characteristics</code>	Додати характеристику	JSON: { backpack_id, name, value }

6. Оновлення інтерфейсу без перезавантаження

Після успішного AJAX-запиту сторінка повинна відобразити зміну без `reload`. Найпростіший і найнадійніший спосіб для навчального проєкту — після кожної успішної операції *повторно* викликати ту саму функцію завантаження списку (яка й так виконується один раз при відкритті сторінки), а не намагатися вручну додати один новий елемент у DOM — це гарантує, що інтерфейс завжди відображає дані, реально підтверджені сервером.

```
async function loadCategories() {
  const response = await fetch('api.php?resource=categories');
  const categories = await response.json();
  document.getElementById('categoryList').innerHTML =
    categories.map(c => `<li>${c.name}</li>`).join('');
}

// Після успішного POST — просто викликати ще раз:
await loadCategories();
```

Завдання на лабораторну роботу

Продовжуйте працювати з предметною областю з лабораторних робіт № 3–5, замінивши SSR-форми на JSON API та SPA-клієнт.

1. Перепроєктуйте обробку кожної форми (крім завантаження файлу) так, щоб дані приймалися сервером у форматі JSON через `php://input`, а не через `$_POST`.
2. Реалізуйте окремий ендпоінт для завантаження файлу, що приймає `multipart/form-data` (оскільки вміст файлу не передається через JSON), і прикріплює завантажений файл до вже створеного запису за його ідентифікатором.
3. Створіть єдину SPA-сторінку (один HTML-файл із вбудованим JavaScript), що завантажується один раз і більше ніколи не перезавантажується під час роботи з даними.
4. Для кожної сутності реалізуйте на SPA-сторінці форму, обробник якої формує JSON-об'єкт із введених значень і надсилає його запитом `fetch()` з методом `POST` і заголовком `Content-Type: application/json`.
5. Для сутності з файлом реалізуйте дворівневе збереження: спочатку JSON-запит створення запису, потім — окремий запит на завантаження файлу з посиланням на отриманий ідентифікатор.
6. Після кожної успішної операції оновлюйте відображені списки записів, повторно запитуючи дані з API — без перезавантаження сторінки браузером.

Вимоги до звіту

Звіт повинен містити: тему й мету роботи; таблицю реалізованих маршрутів API (метод, ресурс, формат тіла запиту); лістинг коду `api.php` та SPA-сторінки; знімки екрана роботи застосунку (список до й після додавання запису без перезавантаження сторінки, вміст вкладки Network інструментів розробника браузера з видимими AJAX-запитами); висновок.

Приклад виконання: «Спортивні рюкзаки»

Приклад продовжує предметну область із попередніх лабораторних робіт. SSR-форми з лабораторної роботи № 5 замінено єдиною SPA-сторінкою, а `api.php` розширено до повноцінного маршрутизатора JSON API.

Структура проєкту

```

lab6/
├── classes/
│   ├── Category.php           (додано toArray())
│   ├── Characteristic.php
│   ├── Backpack.php           (оновлено toArray())
│   └── BackpackRepository.php (додано attachPhoto())
├── data/
│   ├── categories.csv
│   └── backpacks.csv
├── uploads/
├── bootstrap.php
├── api.php                     (маршрутизатор JSON API)
└── index.html                 (SPA: розмітка + fetch-логіка)

```

1. Доповнені класи: toArray() та attachPhoto()

```

<?php
// classes/Category.php – додано метод toArray()
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 toArray(): array {
        return ['id' => $this->id, 'name' => $this->name];
    }

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

```

```

<?php
// classes/Backpack.php – toArray() тепер повертає вкладену категорію й фото
public function toArray(): array {
    return [
        'id' => $this->id,
        'name' => $this->name,
        'category' => $this->category->toArray(),
        'price' => $this->price,
        'volumeLiters' => $this->volumeLiters,
        'photo' => $this->photo,
        'characteristics' => array_map(
            fn(Characteristic $c) => ['name' => $c->getName(), 'value' => $c->getValue()],
            $this->characteristics
        ),
    ];
}

```

```

<?php
// classes/BackpackRepository.php – новий метод для окремого кроку
завантаження фото
public function attachPhoto(int $backpackId, string $photoFilename): bool {
    $backpack = $this->findById($backpackId);
    if ($backpack === null) {
        return false;
    }
    $backpack->setPhoto($photoFilename);
    $this->save();
    return true;
}

```

create() тепер створює рюкзак без фото (сигнатура: name, category, price, volume) — фото більше не передається під час створення, а прикріплюється окремим викликом attachPhoto() після завантаження файлу. bootstrap.php лишається без змін порівняно з попередніми роботами.

2. Маршрутизатор JSON API (api.php)

```

<?php
require_once __DIR__ . '/bootstrap.php';

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

$method = $_SERVER['REQUEST_METHOD'];
$resource = $_GET['resource'] ?? '';
$action = $_GET['action'] ?? null;
$id = isset($_GET['id']) ? (int) $_GET['id'] : null;

function readJsonBody(): array {
    $data = json_decode(file_get_contents('php://input'), true);
    return is_array($data) ? $data : [];
}

function respond(mixed $data, int $status = 200): never {
    http_response_code($status);
    echo json_encode($data, JSON_UNESCAPED_UNICODE);
    exit;
}

```

```

<?php
// --- Категорії ---
if ($resource === 'categories' && $method === 'GET') {
    respond(array_map(
        fn($c) => $c->toArray(),
        array_values($repository->getCategories())
    ));
}

if ($resource === 'categories' && $method === 'POST') {
    $data = readJsonBody();
    $name = trim($data['name'] ?? '');

    if ($name === '') {
        respond(['error' => 'Поле name обов'язкове'], 400);
    }

    $category = $repository->createCategory($name);
    respond($category->toArray(), 201);
}

```

```

<?php
// --- Рюкзаки ---
if ($resource === 'backpacks' && $method === 'GET') {
    respond(array_map(
        fn($b) => $b->toArray(),
        array_values($repository->findAll())
    ));
}

if ($resource === 'backpacks' && $method === 'POST' && $action === null) {
    $data = readJsonBody();
    $name = trim($data['name'] ?? '');
    $category = $repository->findCategoryById((int) ($data['category_id'] ?? 0));
    $price = (float) ($data['price'] ?? 0);
    $volume = (float) ($data['volume'] ?? 0);

    if ($name === '' || $category === null || $price <= 0 || $volume <= 0) {
        respond(['error' => 'Некоректні дані рюкзака'], 400);
    }

    $backpack = $repository->create($name, $category, $price, $volume);
    respond($backpack->toArray(), 201);
}

```

```

<?php
// --- Завантаження фото: окремий ендпоінт, тіло HE у форматі JSON ---
if ($resource === 'backpacks' && $action === 'photo' && $method === 'POST') {
    if ($id === null || $repository->findById($id) === null) {
        respond(['error' => 'Рюкзак не знайдено'], 404);
    }
    if (empty($_FILES['photo']['name']) || $_FILES['photo']['error'] !==
    UPLOAD_ERR_OK) {
        respond(['error' => 'Файл не надіслано або сталася помилка
завантаження'], 400);
    }

    $allowedTypes = ['image/jpeg', 'image/png', 'image/webp'];
    $mimeType = mime_content_type($_FILES['photo']['tmp_name']);

    if (!in_array($mimeType, $allowedTypes, true)) {
        respond(['error' => 'Дозволені лише зображення JPEG, PNG чи WEBP'],
        400);
    }

    $extension = pathinfo($_FILES['photo']['name'], PATHINFO_EXTENSION);
    $filename = uniqid('backpack_', true) . '.' . $extension;
    move_uploaded_file($_FILES['photo']['tmp_name'], __DIR__ . '/uploads/' .
    $filename);

    $repository->attachPhoto($id, $filename);
    respond($repository->findById($id)->toArray());
}

```

```

<?php
// --- Характеристики ---
if ($resource === 'characteristics' && $method === 'POST') {
    $data = readJsonBody();
    $backpackId = (int) ($data['backpack_id'] ?? 0);
    $name = trim($data['name'] ?? '');
    $value = trim($data['value'] ?? '');

    if ($name === '' || $value === '' || $repository->findById($backpackId)
    === null) {
        respond(['error' => 'Некоректні дані характеристики'], 400);
    }

    $repository->addCharacteristicToBackpack($backpackId, $name, $value);
    respond($repository->findById($backpackId)->toArray(), 201);
}

respond(['error' => 'Невідомий маршрут'], 404);

```

Функція `respond()` з типом повернення `never` (PHP 8.1+) повідомляє й людині, що читає код, і самому PHP, що функція ніколи не повертає керування назад — вона завжди завершує скрипт викликом `exit`. Це усуває потребу писати `return` після кожного виклику `respond()` для запобігання подальшому виконанню коду нижче.

3. SPA-сторінка (index.html)

Один файл містить розмітку всіх трьох форм зі списками та всю логіку взаємодії з API. Файл не звертається до жодного файлу PHP напряду (крім самого api.php через fetch) — сервер лише віддає JSON.

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Спортивні рюкзаки – SPA</title>
</head>
<body>
  <h1>Категорії</h1>
  <ul id="categoryList"></ul>
  <form id="categoryForm">
    <input type="text" id="categoryName" placeholder="Назва категорії"
required>
    <button type="submit">Додати категорію</button>
  </form>

  <h1>Рюкзаки</h1>
  <ul id="backpackList"></ul>
  <form id="backpackForm">
    <input type="text" id="backpackName" placeholder="Назва" required>
    <select id="backpackCategory" required></select>
    <input type="number" step="0.01" id="backpackPrice"
placeholder="Ціна" required>
    <input type="number" step="0.1" id="backpackVolume"
placeholder="Об'єм" required>
    <input type="file" id="backpackPhoto" accept="image/*">
    <button type="submit">Додати рюкзак</button>
  </form>

  <h1>Характеристики</h1>
  <form id="characteristicForm">
    <select id="charBackpack" required></select>
    <input type="text" id="charName" placeholder="Назва характеристики"
required>
    <input type="text" id="charValue" placeholder="Значення" required>
    <button type="submit">Додати характеристику</button>
  </form>
```

4. Завантаження й відображення списків

```

<script>
async function loadCategories() {
  const response = await fetch('api.php?resource=categories');
  const categories = await response.json();

  document.getElementById('categoryList').innerHTML = categories
    .map(c => `<li>${c.name}</li>`).join('');

  document.getElementById('backpackCategory').innerHTML = categories
    .map(c => `<option value="${c.id}">${c.name}</option>`).join('');
}

async function loadBackpacks() {
  const response = await fetch('api.php?resource=backpacks');
  const backpacks = await response.json();

  document.getElementById('backpackList').innerHTML = backpacks.map(b => `
    <li>
      ${b.photo ? `` :
''}
      ${b.name} [${b.category.name}] – ${b.volumeLiters} л, ${b.price}
грн
      (${b.characteristics.map(c => `${c.name}: ${c.value}`).join(',
')})
    </li>
  `).join('');

  document.getElementById('charBackpack').innerHTML = backpacks
    .map(b => `<option value="${b.id}">${b.name}</option>`).join('');
}

```

5. Обробники форм: JSON-запити й дворівневе завантаження фото

```

document.getElementById('categoryForm').addEventListener('submit', async
(event) => {
  event.preventDefault();

  await fetch('api.php?resource=categories', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name: document.getElementById('categoryName').value,
    }),
  });

  event.target.reset();
  loadCategories();
});

```

```

document.getElementById('backpackForm').addEventListener('submit', async
(event) => {
    event.preventDefault();

    const payload = {
        name: document.getElementById('backpackName').value,
        category_id:
Number(document.getElementById('backpackCategory').value),
        price: Number(document.getElementById('backpackPrice').value),
        volume: Number(document.getElementById('backpackVolume').value),
    };

    // Крок 1: створення запису JSON-запитом
    const response = await fetch('api.php?resource=backpacks', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload),
    });
    const backpack = await response.json();

    // Крок 2: якщо обрано файл – окремий multipart-запит із отриманим id
    const photoInput = document.getElementById('backpackPhoto');
    if (photoInput.files.length > 0) {
        const formData = new FormData();
        formData.append('photo', photoInput.files[0]);

        await
fetch(`api.php?resource=backpacks&action=photo&id=${backpack.id}`, {
            method: 'POST',
            body: formData,
        });
    }

    event.target.reset();
    loadBackpacks();
});

```

```

document.getElementById('characteristicForm').addEventListener('submit',
async (event) => {
    event.preventDefault();

    await fetch('api.php?resource=characteristics', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            backpack_id:
Number(document.getElementById('charBackpack').value),
            name: document.getElementById('charName').value,
            value: document.getElementById('charValue').value,
        }),
    });

    event.target.reset();
    loadBackpacks();
});

loadCategories();
loadBackpacks();
</script>
</body>
</html>

```

event.preventDefault() на самому початку кожного обробника — критична деталь: без неї браузер виконає стандартну дію HTML-форми (повне перезавантаження сторінки на action-адресу), і вся SPA-поведінка миттєво зламається, повертаючи застосунок до звичайної SSR-логіки лабораторної роботи № 5.

Контрольні запитання

- Чому \$_POST лишається порожнім, коли клієнт надсилає JSON у тілі запиту, і що використовують замість нього?
- Для чого потрібен параметр true у виклику json_decode(\$raw, true) і що повернеться без нього?
- Який заголовок обов'язково додає клієнт при надсиланні JSON через fetch() і навіщо він серверу?
- Чому фото рюкзака не можна включити в той самий JSON-запит, що й решту полів, і як це вирішено в прикладі?
- Опишіть, які саме два окремі HTTP-запити виконує браузер при доданні рюкзака з фото, і в якому порядку.
- Що станеться з поведінкою SPA-сторінки, якщо прибрати виклик event.preventDefault() з обробника форми?
- Чому після успішного POST-запиту сторінка повторно викликає loadBackpacks(), а не додає новий елемент у список вручну через JavaScript?

- Порівняйте маршрут `api.php?resource=backpacks` (метод POST) і `api.php?resource=backpacks&action=photo` (метод POST) — чим принципово відрізняється тіло їхніх запитів?