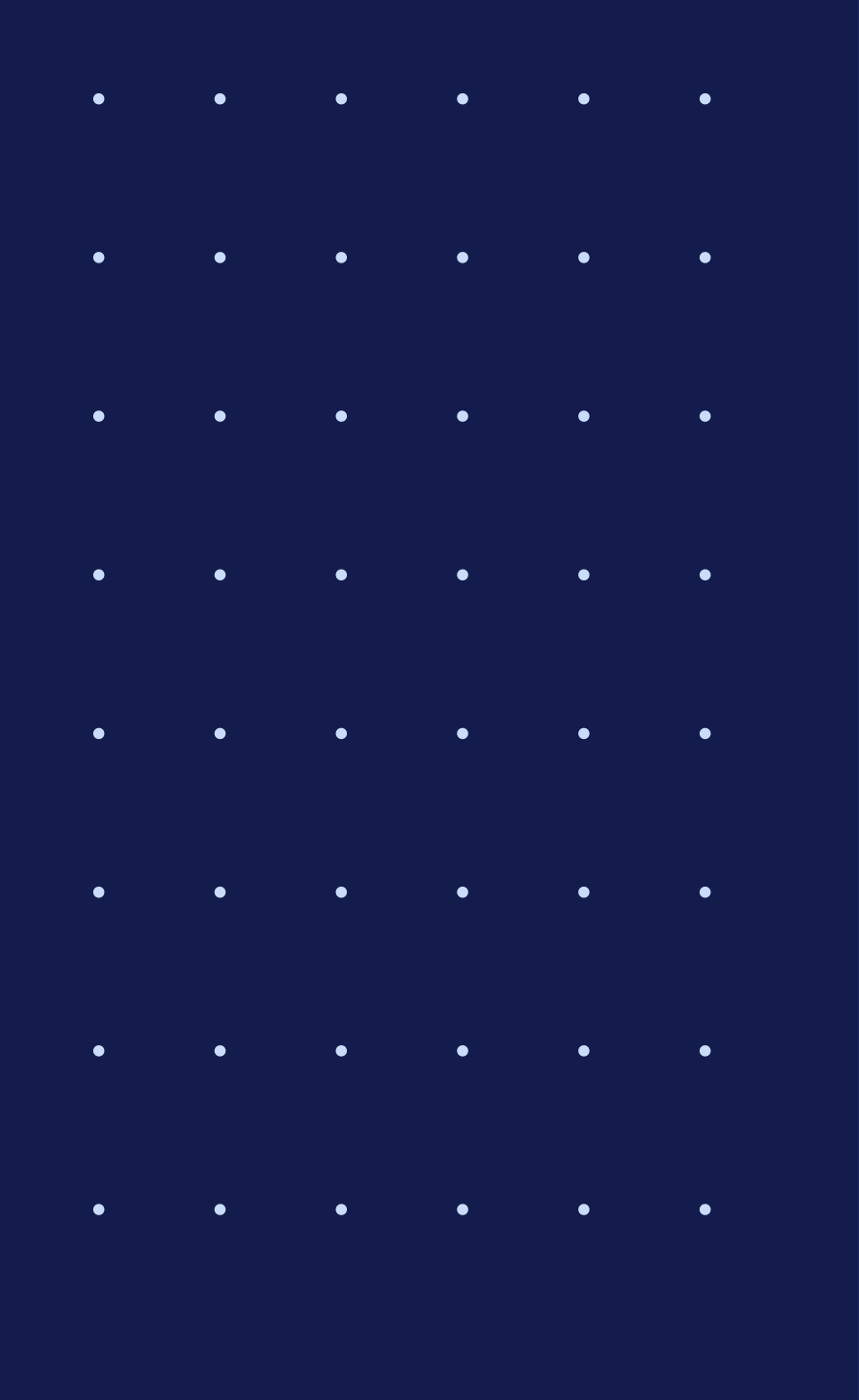




ТЕМА 10

REST API

Принципи · проектування ендпоінтів · коди відповідей · версіонування

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 10 з 18

7 питань сьогоднішньої теми

01

Що таке REST API

02

Принципи REST
архітектури

03

Проектування
ендпоінтів

04

HTTP-методи та CRUD

05

Коди відповідей HTTP

06

Версіонування API

07

Практична реалізація
на PHP

HTTP-запит → JSON-відповідь

```
GET /api/products/42 HTTP/1.1
```

```
Host: shop.example.com
```

```
Accept: application/json
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

```
{"id": 42, "name": "Ноутбук", "price": 25000}
```

REST (Рой Філдінг, 2000) — архітектурний стиль побудови вебAPI. На практиці термін "REST API" здебільшого означає просто "API на основі ресурсів, HTTP-методів і JSON".

П'ять архітектурних обмежень

Принцип	Суть
Клієнт-серверна архітектура	Чіткий розподіл інтерфейсу й даних
Statelessness	Сервер не зберігає стан клієнта між запитам
Cacheability	Відповіді явно позначені як кешовані/некешовані
Уніфікований інтерфейс	Однакові принципи для будь-якого ресурсу
Шарувата система	Клієнт не знає про проміжні шари (проху тощо)

Statelessness: найважливіший принцип

Кожен запит містить УСЮ потрібну інформацію (включно з токеном автентифікації). Сервер не покладається на сесію, збережену з попереднього запиту.

Саме тому JWT (тема 8) так природно поєднується з REST: токен передається в кожному запиті через заголовок Authorization, без залежності від сесії на конкретному сервері.

Ресурс у шляху, дія — у методі

Правильно (REST)	Неправильно
GET /products	GET /getAllProducts.php
POST /products	GET /createProduct.php?name=...
GET /products/42	GET /getProduct.php?id=42
DELETE /products/42	GET /deleteProduct.php?id=42

URI ідентифікує РЕСУРС (іменник), а не дію над ним — дія визначається HTTP-методом.

Правила іменування

- 1 Іменники в множині для колекцій: /products, /users
- 2 Вкладеність відображає зв'язки: /users/5/orders
- 3 Без дієслів у шляху — дія вже в HTTP-методі
- 4 Дефіс, а не підкреслення: /product-categories
- 5 Уникати вкладеності глибше 2–3 рівнів

Один URI, різні методи

Метод	CRUD	Приклад	Ідемпотентний?
GET	Read	GET /products/42	Так
POST	Create	POST /products	Ні
PUT	Update (повна)	PUT /products/42	Так
PATCH	Update (часткова)	PATCH /products/42	Ні
DELETE	Delete	DELETE /products/42	Так

PUT (повна заміна) vs PATCH (часткова)

PUT

```
PUT /products/42
{
  "name": "Ноутбук",
  "price": 27000,
  "category": "electronics"
}
```

PATCH

```
PATCH /products/42
{
  "price": 27000
}
```

Найважливіші коди в REST API

Код	Ситуація
200 OK	Успішний GET, PUT, PATCH
201 Created	Успішний POST — ресурс створено
204 No Content	Успішний DELETE
400 Bad Request	Невалідний JSON, відсутнє поле
401 / 403	Не автентифікований / немає прав
404 Not Found	Ресурс за таким шляхом не існує
409 Conflict	Ресурс уже існує (напр. email зайнятий)
422 Unprocessable Entity	Дані не проходять валідацію

Версія в шляху URI

```
GET /api/v1/products/42
```

```
GET /api/v2/products/42
```

```
GET /api/products/42
```

```
Accept: application/vnd.shop.v2+json
```

Вгорі: версія в шляху (простіше). Внизу: версія в заголовку (URI лишається "чистим").

Порівняння підходів

Підхід	Переваги	Недоліки
У шляху (/v1/, /v2/)	Простий, легко тестувати	"Забруднює" URI
У заголовку	URI лишається чистим	Складніше дебажити

Головне правило: ніколи не вносити breaking changes у наявну версію API. Зміни структури — лише у новій версії; стара версія продовжує працювати для існуючих клієнтів.

Контролер: show() — GET-ендпоінт

```
class ProductApiController {  
    public function __construct(private ProductRepository $repository) {}  
  
    public function show(int $id): void {  
        $product = $this->repository->findById($id);  
        header('Content-Type: application/json');  
  
        if ($product === null) {  
            http_response_code(404);  
            echo json_encode(['error' => 'Товар не знайдено']);  
            return;  
        }  
  
        http_response_code(200);  
        echo json_encode($product);  
    }  
}
```

Контролер: store() — POST-ендпоінт

```
public function store(): void {
    header('Content-Type: application/json');
    $data = json_decode(file_get_contents('php://input'), true);

    if (empty($data['name']) || empty($data['price'])) {
        http_response_code(422);
        echo json_encode(['error' => 'Поля name і price обов\'язкові']);
        return;
    }

    $id = $this->repository->create($data['name'], $data['price']);
    http_response_code(201);
    echo json_encode(['id' => $id, 'name' => $data['name']]);
}
```

Маршрутизація з урахуванням методу

```
$method = $_SERVER['REQUEST_METHOD'];  
$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);  
  
if (preg_match('#^/products/(\d+)$#', $uri, $matches)) {  
    $id = (int) $matches[1];  
    match ($method) {  
        'GET' => $controller->show($id),  
        'PUT', 'PATCH' => $controller->update($id),  
        'DELETE' => $controller->destroy($id),  
        default => http_response_code(405),  
    };  
} elseif ($uri === '/products' && $method === 'POST') {  
    $controller->store();  
}
```

405 Method Not Allowed — URI впізнано, але метод не підтримується (відрізняється від 404).

Підсумок теми 10

- URI ідентифікує ресурс (іменник), дію визначає HTTP-метод
- Statelessness — кожен запит самодостатній, без залежності від сесії сервера
- PUT = повна заміна ресурсу, PATCH = часткове оновлення
- Код статусу — головний спосіб повідомити результат, а не поле success у JSON
- Версіонування (у шляху або заголовку) захищає існуючих клієнтів від breaking changes
- REST-контролер — той самий MVC-контролер, що повертає JSON замість HTML