



ТЕМА 8

Автентифікація та авторизація

Сесії · хешування паролів · ролі доступу · JWT · OAuth2 (оглядово)

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 8 з 18

7 питань сьогоднішньої теми

01

**Автентифікація та
авторизація: поняття**

02

Сесії в PHP

03

Хешування паролів

04

**Авторизація: ролі та
права**

05

Токени та JWT (огляд)

06

OAuth2 (огляд)

07

**Практичні поради з
безпеки**

Автентифікація vs авторизація

Автентифікація

Хто ти?

Вхід за логіном і паролем

Авторизація

Що тобі дозволено?

Перевірка ролі «адміністратор»

401 Unauthorized → не автентифікований. **403 Forbidden** → автентифікований, але не авторизований.

Як працюють сесії



Клієнт зберігає лише ID сесії; усі фактичні дані користувача залишаються на сервері.

Використання \$_SESSION

```
session_start();    // ОБОВ'ЯЗКОВО перед будь-яким виводом

// Збереження даних після успішного входу
$_SESSION['user_id'] = 42;
$_SESSION['username'] = 'oksana';

// Перевірка в наступних запитах
if (isset($_SESSION['user_id'])) {
    echo 'Вітаємо, ' . $_SESSION['username'];
} else {
    echo 'Будь ласка, увійдіть у систему';
}

// Вихід із системи
session_destroy();
```

password_hash(): реєстрація

```
$password = $_POST['password'];  
$hash = password_hash($password, PASSWORD_DEFAULT);  
  
$stmt = $pdo->prepare(  
    'INSERT INTO users (email, password_hash) VALUES (:email, :hash)'  
);  
$stmt->execute(['email' => $_POST['email'], 'hash' => $hash]);
```

Паролі НІКОЛИ не зберігають у відкритому вигляді. password_hash() автоматично використовує bcrypt і додає "сіль" проти rainbow-таблиць.

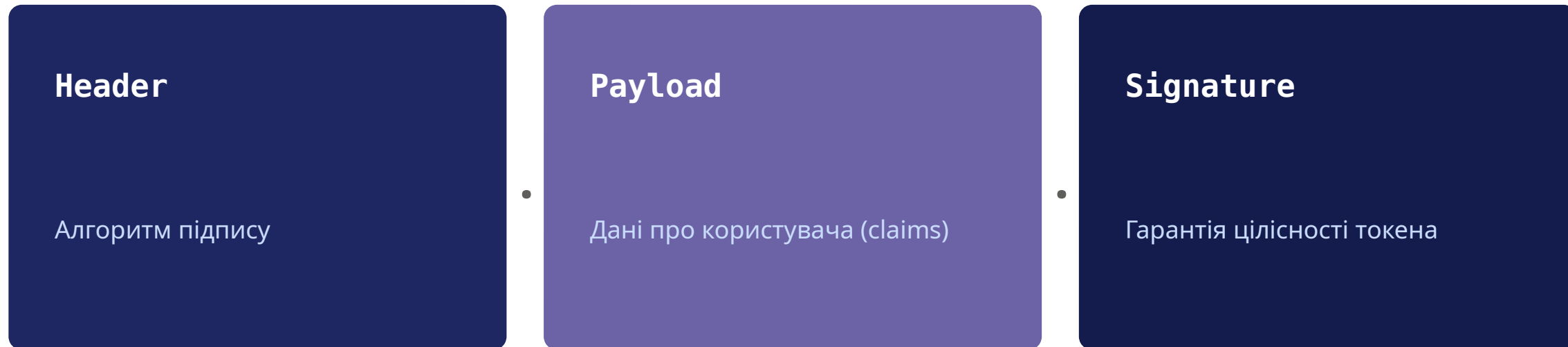
password_verify(): вхід

```
$stmt = $pdo->prepare('SELECT id, password_hash FROM users WHERE email = :email');  
$stmt->execute(['email' => $_POST['email']]);  
$user = $stmt->fetch(PDO::FETCH_ASSOC);  
  
if ($user && password_verify($_POST['password'], $user['password_hash'])) {  
    session_start();  
    $_SESSION['user_id'] = $user['id'];  
    echo 'Вхід виконано успішно';  
} else {  
    echo 'Невірний email або пароль';  
}
```

Перевірка ролі (RBAC)

```
function requireLogin(): void {  
    if (!isset($_SESSION['user_id'])) {  
        http_response_code(401);  
        exit('Необхідна автентифікація');  
    }  
}  
  
function requireRole(string $role): void {  
    requireLogin();  
    if (($_SESSION['role'] ?? null) !== $role) {  
        http_response_code(403);  
        exit('Недостатньо прав доступу');  
    }  
}  
  
requireRole('admin');  
echo 'Ласкаво просимо до адмін-панелі';
```

Структура JWT: три частини



Сервер перевіряє достовірність токена за підписом, не звертаючись до бази даних чи сховища сесій.

Видача й перевірка токена (концептуально)

```
// Видача токена після входу
$payload = ['sub' => $user['id'], 'exp' => time() + 3600];
$token = JWT::encode($payload, $secretKey, 'HS256');
echo json_encode(['token' => $token]);

// Перевірка: клієнт надсилає Authorization: Bearer <token>
$authHeader = $_SERVER['HTTP_AUTHORIZATION'] ?? '';
$token = str_replace('Bearer ', '', $authHeader);

try {
    $decoded = JWT::decode($token, new Key($secretKey, 'HS256'));
    $userId = $decoded->sub;
} catch (Exception $e) {
    http_response_code(401);
    exit('Недійсний або прострочений токен');
}
```

Основні ролі протоколу

Роль	Опис
Resource Owner	Користувач, якому належать дані
Client	Застосунок, що хоче отримати доступ
Authorization Server	Автентифікує користувача й видає токени
Resource Server	Зберігає захищені дані, приймає токен

OAuth2 — це протокол делегованої авторизації, а не автентифікація напряму: дозволяє доступ до даних без передачі пароля.

Потік Authorization Code

- 1 Користувач натискає «Увійти через Google»
- 2 Сайт перенаправляє на сторінку авторизації Google
- 3 Google повертає тимчасовий authorization code
- 4 Сайт обмінює код на access-токен (сервер-сервер)
- 5 Токен використовується для звернень до Resource Server

Контрольний список

Рекомендація	Чому
Завжди HTTPS	Без шифрування дані передаються відкритим текстом
HttpOnly + Secure для cookie	Захист від XSS та передачі лише через HTTPS
Обмеження спроб входу	Захист від brute-force атак
password_hash(), не власна реалізація	Враховує сучасні криптографічні практики
session_regenerate_id() після входу	Захист від session fixation

Підсумок теми 8

- Автентифікація відповідає "хто ти", авторизація — "що тобі дозволено"
- Сесії зберігають стан користувача на сервері; клієнт має лише ID
- Паролі завжди зберігаються як хеш (`password_hash`), ніколи у відкритому вигляді
- JWT дозволяє перевіряти автентичність без спільного сховища сесій
- OAuth2 — делегована авторизація без передачі пароля третій стороні
- HTTPS, `HttpOnly-cookie` та `session_regenerate_id()` — базовий мінімум безпеки