

Дисципліна

**«Бекенд-розробка, сервіси та мікросервіси»**

## **КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 8**

*Автентифікація та авторизація: сесії, токени, JWT, OAuth2 (оглядово)*

Спеціальність: Комп'ютерна інженерія, 3 курс

Обсяг: 1 год 20 хв (лекція 8 з 18)

## План лекції

1. Автентифікація та авторизація: базові поняття
2. Сесії в PHP: механізм і використання
3. Реєстрація та вхід користувача: хешування паролів
4. Авторизація: перевірка ролей і прав доступу
5. Токени та JWT (огляд)
6. OAuth2 (огляд)
7. Практичні поради з безпеки автентифікації

*Мета лекції: зрозуміти різницю між автентифікацією й авторизацією, навчитися реалізовувати вхід користувача через сесії PHP з безпечним зберіганням паролів, а також отримати оглядове уявлення про токен-базовану автентифікацію (JWT) і протокол OAuth2, які широко використовуються в сучасних API та мікросервісах.*

## 1. Автентифікація та авторизація: базові поняття

Автентифікація (authentication) відповідає на запитання «хто ти?» — це процес підтвердження особи користувача, зазвичай через пару логін/пароль, токен чи інший доказ ідентичності. Авторизація (authorization) відповідає на запитання «що тобі дозволено робити?» — це процес визначення прав доступу вже ідентифікованого користувача до конкретних ресурсів чи дій.

Ці два поняття часто плутають, хоча вони описують послідовні, але різні кроки: спочатку система встановлює, хто саме звертається (автентифікація), а вже потім перевіряє, чи має ця конкретна особа право виконати запитану дію (авторизація) — наприклад, звичайний користувач може бути успішно автентифікований, але не авторизований для видалення чужих дописів чи доступу до адміністративної панелі.

| Поняття        | Питання            | Приклад                                           |
|----------------|--------------------|---------------------------------------------------|
| Автентифікація | Хто ти?            | Вхід за логіном і паролем                         |
| Авторизація    | Що тобі дозволено? | Перевірка, чи має користувач роль «адміністратор» |

У HTTP-контексті також часто згадують код статусу 401 Unauthorized (насправді означає «не автентифікований» — токен чи сесія відсутні або недійсні) і 403 Forbidden («автентифікований, але не авторизований» — особу підтверджено, але доступ до ресурсу заборонено). Незважаючи на назву 401 (Unauthorized), він стосується саме автентифікації — це історична неточність у специфікації HTTP.

## 2. Сесії в PHP: механізм і використання

HTTP — протокол без стану (stateless): кожен запит обробляється незалежно, і сервер за замовчуванням «не пам'ятає» попередні запити того самого клієнта. Сесії — стандартний механізм PHP для збереження стану користувача (наприклад, факту успішного входу) між окремими HTTP-запитами.

### 2.1. Як працюють сесії

1. При виклику `session_start()` PHP перевіряє, чи прийшов від клієнта cookie з ідентифікатором сесії (за замовчуванням `PHPSESSID`).
2. Якщо cookie відсутній, PHP генерує новий унікальний ідентифікатор сесії та надсилає його клієнту через Set-Cookie.
3. Дані сесії зберігаються на сервері (за замовчуванням — у тимчасових файлах), а клієнт зберігає лише ідентифікатор.
4. У кожному наступному запиті браузер автоматично надсилає той самий cookie, і PHP підвантажує відповідні дані сесії в суперглобальний масив `$_SESSION`.

### 2.2. Використання `$_SESSION`

```
<?php
session_start();    // ОБОВ'ЯЗКОВО перед будь-яким виводом і роботою з
$_SESSION

// 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();
```

*`session_start()` повинна викликатись до будь-якого виводу HTML чи `echo` — вона встановлює HTTP-заголовок Set-Cookie, а заголовки не можна надсилати після того, як почався вивід тіла відповіді (та сама причина помилки «headers already sent», розглянутої в темі про вихідний буфер).*

### 3. Реєстрація та вхід користувача: хешування паролів

Паролі користувачів у базі даних ніколи не можна зберігати у відкритому вигляді (plain text): у разі витоку бази даних зломисник миттєво отримає доступ до всіх облікових записів. Замість пароля зберігається його хеш — результат односторонньої криптографічної функції, з якої практично неможливо відновити оригінальний пароль.

#### 3.1. password\_hash() та password\_verify()

PHP надає вбудовані функції для безпечного хешування паролів, які автоматично використовують сучасний алгоритм (bcrypt за замовчуванням) і додають «сіль» (salt) — випадкові дані, що унеможливляють атаки за готовими таблицями хешів (rainbow tables).

```
<?php
// Реєстрація: хешування пароля перед збереженням у БД
$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]);
```

```
<?php
// Вхід: перевірка введеного пароля проти збереженого хешу
$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';
}
```

*Повідомлення про помилку входу навмисно однакове для випадків «такого email не існує» і «пароль невірний» — якщо повідомляти про них окремо, зломисник міг би перебором визначити, які email-адреси зареєстровані в системі (це називається enumeration attack).*

## 4. Авторизація: перевірка ролей і прав доступу

Після автентифікації система зазвичай має визначити, до яких дій чи ресурсів має доступ саме цей користувач. Найпростіший підхід — рольова модель (Role-Based Access Control, RBAC): кожному користувачу призначається одна чи кілька ролей (наприклад, «користувач», «модератор», «адміністратор»), і доступ до функціоналу перевіряється за наявністю потрібної ролі.

```
<?php
session_start();

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');
    }
}

// Використання на сторінці адмін-панелі
requireRole('admin');
echo 'Ласкаво просимо до адмін-панелі';
```

*Перевірку прав доступу необхідно виконувати на сервері для кожного запиту, що торкається захищеного ресурсу чи дії — приховування кнопки «Видалити» в інтерфейсі для звичайних користувачів (лише на клієнті) не є авторизацією: досвідчений користувач може напряму звернутися до відповідного URL чи API-ендпоінту, минаючи інтерфейс.*

## 5. Токени та JWT (огляд)

Сесії добре працюють для класичних вебзастосунків, де клієнт і сервер підтримують cookies, але погано підходять для API, які обслуговують мобільні застосунки чи інші сервери — там немає браузера, який автоматично керує cookies, а сама модель «сервер пам'ятає стан кожного клієнта» погано масштабується на кілька серверів. Токен-базована автентифікація вирішує цю проблему: після входу сервер видає клієнту токен, який той сам зберігає й надсилає з кожним наступним запитом (найчастіше в заголовку Authorization).

### 5.1. Структура JWT

JWT (JSON Web Token) — популярний формат токена, що складається з трьох частин, розділених крапками, кожна з яких закодована в Base64: заголовок (header) з інформацією про алгоритм підпису, корисне навантаження (payload) з даними про користувача (claims), і підпис (signature), який гарантує, що вміст токена не було змінено.

```
// Приклад структури JWT (три частини, розділені крапками)
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI0MiIsIm5hbWUiOiJPa3NhbmEifQ.4f8a2b...
└── header ─┘ └── payload ─┘ └── signature ─┘
```

Ключова властивість JWT — сервер може перевірити достовірність токена (за підписом), не звертаючись до бази даних чи сховища сесій: уся потрібна інформація вже міститься у самому токені. Це і робить JWT зручним для розподілених систем і мікросервісів, де кілька незалежних серверів повинні перевіряти автентичність запитів без спільного сховища сесій.

### 5.2. Спрощений приклад використання (концептуально)

```
<?php
// Видача токена після успішного входу (потрібна бібліотека, напр.
firebase/php-jwt)
$payload = ['sub' => $user['id'], 'exp' => time() + 3600];
$token = JWT::encode($payload, $secretKey, 'HS256');

echo json_encode(['token' => $token]);

// Перевірка токена в наступних запитах (клієнт надсилає його в заголовку)
// 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');
}
```

На відміну від сесій, JWT неможливо достроково «відкликати» (*server-side revoke*) без додаткової інфраструктури (наприклад, чорного списку токенів), оскільки сервер не зберігає стан токена — він лише перевіряє підпис і термін дії (*exp*). Тому JWT зазвичай видають на короткий термін і оновлюють через окремий *refresh*-токен.

## 6. OAuth2 (огляд)

OAuth2 — це не спосіб автентифікації користувача застосунком напряду, а протокол делегованої авторизації: він дозволяє одному застосунку отримати обмежений доступ до ресурсів користувача на іншому сервісі без передачі власного пароля цього сервісу. Класичний приклад — кнопка «Увійти через Google» чи надання стороннього застосунку доступу до списку контактів у Google без розкриття пароля від облікового запису Google.

### 6.1. Основні ролі в OAuth2

| Роль                 | Опис                                                                     |
|----------------------|--------------------------------------------------------------------------|
| Resource Owner       | Користувач, якому належать дані (напр. власник акаунту Google)           |
| Client               | Застосунок, що хоче отримати доступ до даних (напр. ваш сайт)            |
| Authorization Server | Сервер, що автентифікує користувача й видає токени (напр. сервер Google) |
| Resource Server      | Сервер, що зберігає захищені дані та приймає токен для доступу до них    |

### 6.2. Спрощений потік Authorization Code

1. Користувач натискає «Увійти через Google» на сайті-клієнті.
2. Сайт перенаправляє користувача на сторінку авторизації Google, де користувач вводить свої облікові дані Google (сайт-клієнт їх ніколи не бачить).
3. Після згоди користувача Google перенаправляє його назад на сайт-клієнт із тимчасовим кодом авторизації (authorization code) у параметрі URL.
4. Сайт-клієнт обмінює цей код на access-токен, звертаючись напряду до сервера Google (цей обмін відбувається "за лаштунками", між серверами).
5. Отриманий токен сайт-клієнт використовує для звернень до Resource Server Google (наприклад, щоб отримати ім'я та email користувача).

У межах цього курсу OAuth2 розглядається лише оглядово — реалізація повноцінного Authorization Server із нуля виходить за рамки початкового курсу бекенд-розробки. На практиці для інтеграції з Google, Facebook чи іншими провайдерами використовують готові бібліотеки (наприклад, *league/oauth2-client*), які інкапсують усі деталі протоколу.

## 7. Практичні поради з безпеки автентифікації

| Рекомендація                                                      | Чому це важливо                                                                                              |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Завжди використовувати HTTPS                                      | Без шифрування пароль, сесійний cookie чи токен передаються відкритим текстом і можуть бути перехоплені      |
| Встановлювати прапорці HttpOnly і Secure для сесійних cookies     | HttpOnly забороняє доступ до cookie з JavaScript (захист від XSS); Secure дозволяє передачу лише через HTTPS |
| Обмежувати кількість спроб входу                                  | Захист від brute-force атак — перебору паролів                                                               |
| Використовувати password_hash(), а не власні реалізації хешування | Вбудована функція автоматично враховує сучасні криптографічні практики                                       |
| Не розкривати деталі помилки входу                                | Запобігає enumeration-атакам на існуючі облікові записи                                                      |
| Регенерувати ID сесії після входу (session_regenerate_id())       | Захист від session fixation — коли зломисник підставляє відомий йому ID сесії жертві                         |

```
<?php
session_start();

// Після успішної автентифікації
session_regenerate_id(true); // новий ID сесії, старий видаляється
$_SESSION['user_id'] = $user['id'];
```

## Питання для самоперевірки

- Поясніть різницю між автентифікацією та авторизацією на власному прикладі.
- Чому HTTP-код 401 Unauthorized стосується автентифікації, а не авторизації, попри свою назву?
- Опишіть, як PHP пов'язує конкретний браузер з даними його сесії на сервері.
- Чому паролі зберігають як хеш, а не у зашифрованому вигляді (з можливістю розшифрування)?
- Для чого повідомлення про помилку входу навмисно роблять однаковим для неіснуючого email і невірного пароля?
- Чим токен-базована автентифікація (JWT) відрізняється від сесійної з погляду масштабованості на кілька серверів?
- Яку роль виконує Authorization Server у протоколі OAuth2?
- Для чого потрібно викликати `session_regenerate_id()` одразу після успішного входу користувача?

## Короткий глосарій

| Термін               | Означення                                                             |
|----------------------|-----------------------------------------------------------------------|
| Автентифікація       | Підтвердження особи користувача                                       |
| Авторизація          | Визначення прав доступу автентифікованого користувача                 |
| Сесія                | Механізм збереження стану користувача між HTTP-запитами на сервері    |
| Хешування пароля     | Одностороннє незворотне перетворення пароля перед збереженням         |
| JWT (JSON Web Token) | Самодостатній токен із підписом для передачі даних про автентифікацію |
| OAuth2               | Протокол делегованої авторизації без передачі пароля третій стороні   |
| RBAC                 | Role-Based Access Control — модель авторизації на основі ролей        |
| Session fixation     | Атака з підстановкою відомого злоумиснику ID сесії жертві             |