📃 CIS 334 – Authentication with JSON Web Tokens (JWTs)

🎯 Learning Objectives

By the end of this lesson, you should be able to:

- Explain what a **JSON Web Token (JWT)** is and how it differs from PHP sessions
- Describe the structure of a JWT and how it is verified
- Generate and validate JWTs securely in PHP 8.3
- Store and send tokens using `localStorage` or `sessionStorage` in the browser
- Understand when to use JWTs instead of traditional session-based authentication

---

🔍 Overview

JSON Web Tokens (JWTs) provide a **stateless** way to handle authentication. Instead of storing login data on the server (like PHP sessions do), JWTs encode that data in a signed token that the **client stores** and sends back with each request.

JWTs are commonly used in **API-based** or **single-page applications (SPAs)**, where maintaining traditional PHP sessions isn't practical.

🧠 How JWT Authentication Works

[Slide diagram: Client ↔ Server ↔ Token validation]

1. User logs in with username and password.

2. PHP verifies credentials and generates a **JWT** containing user information.

3. The token is sent to the client.

4. The client stores the token (in `localStorage` or `sessionStorage`).

5. On each request, the client sends the token in the **Authorization header**:

   ```
   Authorization: Bearer <token>
   ```

6. The server verifies the token's signature and allows access if it's valid.

This approach eliminates the need for server-side session storage — the state travels with the token itself.

🔎 Structure of a JWT

A JWT consists of **three parts**, separated by dots (`.`):

```
header.payload.signature
```

Example:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VySWQiOjEsInVzZXJuYW1lIjoiYWxleCJ9.
kNqpxx-r5v8hO7K5aR6D6aRZ0hz5TJKk0INxX5sWQoI
```

1. **Header** – specifies algorithm and token type (e.g., HS256).
2. **Payload** – contains user claims (like ID, role, or expiration).
3. **Signature** – verifies that the token hasn't been altered.

⚙️ Step-by-Step Implementation in PHP 8.3

1️⃣ Setup and Secret Key

Create a `config.php` file for the signing key:

```php
<?php
const JWT_SECRET = 'ChangeThisToASecretRandomKey123!';
```

2️⃣ Generating a Token After Login

`login.php`

```php
<?php
declare(strict_types=1);
require_once 'config.php';
require_once 'database.php';

use Firebase\JWT\JWT;
use Firebase\JWT\Key;   // if using composer package "firebase/php-jwt"

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = trim($_POST['username'] ?? '');
    $password = $_POST['password'] ?? '';

    $pdo = Database::getConnection();
    $stmt = $pdo->prepare('SELECT id, password_hash, role FROM users WHERE
username = :username');
    $stmt->execute([':username' => $username]);
    $user = $stmt->fetch();
    if ($user && password_verify($password, $user['password_hash'])) {
        $payload = [
            'userId' => (int)$user['id'],
            'username' => $username,
            'role' => $user['role'],
            'exp' => time() + 3600  // expires in 1 hour
        ];

        $jwt = JWT::encode($payload, JWT_SECRET, 'HS256');

        // send the token as JSON
        header('Content-Type: application/json');
        echo json_encode(['token' => $jwt]);
        exit;
    }

    http_response_code(401);
    echo json_encode(['error' => 'Invalid credentials']);
}
```

---

3 Validating the Token (API Endpoint)

`dashboard.php`

```php
<?php
declare(strict_types=1);
require_once 'config.php';

use Firebase\JWT\JWT;
use Firebase\JWT\Key;

$headers = getallheaders();
$authHeader = $headers['Authorization'] ?? '';
if (!str_starts_with($authHeader, 'Bearer ')) {
    http_response_code(401);
    exit('Missing or invalid token');
}

$jwt = substr($authHeader, 7);

try {
    $decoded = JWT::decode($jwt, new Key(JWT_SECRET, 'HS256'));
    echo "Welcome, " . htmlspecialchars($decoded->username);
} catch (Exception $e) {
    http_response_code(401);
    exit('Invalid or expired token');
}
```

Here, the token is **verified using the shared secret key**. If the token is expired or altered, `JWT::decode()` throws an exception.

💾 Storing JWTs in the Browser

In **JavaScript**, store the token in browser storage after login:

```
// Example: storing token after a successful login
fetch('login.php', { method: 'POST', body: formData })
  .then(res => res.json())
  .then(data => {
    localStorage.setItem('jwt', data.token);
    window.location.href = 'dashboard.html';
  });
```

When making API requests:

```
const token = localStorage.getItem('jwt');

fetch('dashboard.php', {
  headers: { 'Authorization': 'Bearer ' + token }
});
```

Choosing storage:

- **localStorage:** persists even after the browser closes (useful for “Remember me”).
- **sessionStorage:** cleared when the tab closes (safer for short sessions).

🔐 Security Considerations

- Always use **HTTPS** — JWTs are as sensitive as passwords.
- Keep tokens short-lived (`exp` claim).
- Consider using **refresh tokens** for long-term sessions.
- Avoid storing tokens in cookies to prevent CSRF exposure.
- When the user logs out, remove the token from browser storage.

🧩 JWTs vs. PHP Sessions

Feature	JWT	PHP Session
Storage	Client (token)	Server (session data)
State	Stateless	Stateful
Best for	APIs, SPAs, mobile apps	Traditional PHP web apps

Feature	JWT	PHP Session
Scalability	Easy to scale across servers	Needs shared storage
Security	Signed but exposed if stolen	Server-controlled, safer for critical data

🧠 Quick Review

1. What are the three parts of a JWT?
2. How does the server verify that a token hasn't been altered?
3. Where should you store JWTs in a browser for short-term sessions?
4. What happens if the `exp` time has passed when verifying a token?
5. How is token-based authentication different from PHP's session-based approach?

🎯 Practice Exercise

1. Install the `firebase/php-jwt` package with Composer.
2. Implement a login endpoint that returns a JWT.
3. Build a simple HTML + JavaScript page that stores the token in `sessionStorage` and uses it for authenticated API calls.
4. Add expiration logic and verify that expired tokens are rejected.
5. Bonus: implement a "refresh token" that issues a new JWT when the old one expires.

✅ Key Takeaways

- **JWTs** allow stateless, scalable authentication without server-side sessions.
- Each token carries its own signed claims — verified by the server on every request.
- PHP 8.3 works seamlessly with modern JWT libraries like `firebase/php-jwt`.
- Store tokens securely and validate them carefully on every request.
- JWTs are ideal for **APIs**, **single-page apps**, and **mobile clients** — while sessions remain better suited for classic PHP websites.