📃 CIS 334 — State Management & Security in PHP

🎯 Module Goals

After completing this two-week module, you should be able to:

- Explain how HTTP's stateless nature affects web applications
- Use PHP tools to maintain state securely
- Implement login and authorization systems
- Protect forms from CSRF attacks
- Understand when to use server-side sessions vs. client-side tokens

---

🌐 1️⃣ The Stateless Web

HTTP doesn't remember users between requests — every page load is independent. PHP provides tools to "bridge" that gap:

Technique	Where Data Lives	Typical Use
Hidden Fields	Form body	Passing small state info between form submissions
Query Strings	URL	Filtering, paging, linking
Cookies	Browser	Remembering simple preferences or identifiers
Sessions	Server	Secure, temporary user data (login, carts)

🍪 2️⃣ Cookies

- Stored **in the browser** and sent with each request.
- Created using `setcookie(name, value, expires, path, domain, secure, httponly)`.
- Best for lightweight, non-sensitive data.

✅ Use for "Remember me" options or interface settings. 🚫 Don't store passwords or IDs directly — use session IDs instead.

🧠 3️⃣ Sessions

- Data stored **on the server**; client only keeps a small session ID cookie.
- Started with `session_start()` and accessed via `$_SESSION`.
- Ends with `session_destroy()`.

✅ Secure because the user can't modify server-side data. ⚙️ Configurable via `session_set_cookie_params()` for extra security.

🔐 4️⃣ Authentication

- Confirms **who the user is**.
- Typically uses a login form → database check → session storage.
- PHP 8.3's `password_hash()` and `password_verify()` handle secure password storage.

✅ Hash all passwords — never store them as plain text. ⚙️ Use `password_needs_rehash()` to upgrade old hashes automatically.

🧩 5️⃣ Authorization

- Controls **what the authenticated user can do**.
- Commonly role-based (`intern`, `staff`, `admin`).
- Checked using session data before granting access.

✅ Always verify roles server-side. ⚙️ Return `403 Forbidden` for unauthorized access. 🧱 PHP 8.3's `match` expression simplifies role checks.

🧷 6️⃣ CSRF Protection

- Prevents **Cross-Site Request Forgery** attacks on authenticated users.
- Requires a **CSRF token** stored in the session and validated on form submission.

✅ Use `random_bytes()` to generate tokens. ✅ Validate with `hash_equals()` for timing-safe comparison. 🚫 Reject requests missing or mismatching tokens.

🔑 7️⃣ JSON Web Tokens (JWTs)

- Stateless authentication alternative to sessions.
- Encodes user info + signature; verified by the server on each request.
- Ideal for APIs, SPAs, and mobile clients.

Feature	JWT	Session
Storage	Client	Server
State	Stateless	Stateful
Best for	APIs / SPAs	Classic PHP apps
Validation	Signature	Server lookup

✅ Use libraries like `firebase/php-jwt`. ⚙️ Send tokens via `Authorization: Bearer <token>` header. 🚫 Always use HTTPS; store short-term tokens in `sessionStorage`.

🧭 Putting It All Together

Secure PHP workflow example:

1. User registers — password hashed with `password_hash()`.

2. User logs in — credentials verified; session started.
3. Authorization rules determine accessible pages.
4. CSRF tokens protect all form submissions.
5. (For APIs) JWTs replace sessions for stateless authentication.

🧠 Quick Recap Checklist

✅ I can explain the difference between **cookies**, **sessions**, and **tokens**. ✅ I can use **password_hash()** and **password_verify()** for secure logins. ✅ I can restrict pages based on **roles or permissions**. ✅ I can generate and validate **CSRF tokens**. ✅ I can implement **JWT authentication** for stateless clients.

🧩 Practice Ideas

- Convert your existing PHP login system to include **roles**.
- Add CSRF protection to all POST forms.
- Create a simple **JWT API** for user info retrieval.
- Compare the behavior of `localStorage` vs. `sessionStorage` for token storage.

✅ Key Takeaways

- **State management** is essential for dynamic, interactive web applications.
- **Security** depends on where and how that state is stored.
- PHP 8.3 offers strong, modern APIs for encryption, hashing, and randomness.
- Combining cookies, sessions, tokens, and secure coding practices builds trustworthy, maintainable web systems.