🗒️ CIS 334 – Authorization in PHP (PHP 8.3)

🎯 Learning Objectives

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

- Explain what authorization means and how it differs from authentication
- Implement basic role-based authorization in PHP
- Restrict access to specific pages or actions based on user roles
- Store and verify user permissions securely
- Use PHP 8.3 language features to simplify authorization logic

---

🔍 Overview

Authentication verifies *who* a user is. **Authorization** determines *what* that user can do once authenticated.

Common examples of authorization include:

- Only administrators can view user lists
- Interns can edit their own applications but not others'
- Logged-in users can view opportunities, but only staff can approve them

In PHP applications, authorization logic typically runs **after authentication** — once the user's identity has been confirmed and stored in a session.

🧠 Understanding Authorization Flow

[Diagram idea for slides: "Authentication → Role Check → Page Access"]

1. **Authenticate the user** (via login).
2. **Retrieve the user's role** or permission level from the database.
3. **Store that role** in the user's session (e.g., `$_SESSION['role']`).
4. **Check the role** when loading restricted pages or performing sensitive actions.

This simple model is often called **role-based access control (RBAC)**.

⚙️ Step-by-Step Example

1️⃣ Database Structure

Extend the `users` table to include a `role` column:

```sql
ALTER TABLE users ADD COLUMN role ENUM('intern', 'staff', 'admin') NOT NULL
DEFAULT 'intern';
```

Now each user has a defined role that determines what they’re allowed to access.

---

2 Storing Role Information on Login

`login.php`

```php
<?php
session_start();
require_once 'database.php';

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'])) {
        $_SESSION['userId'] = (int)$user['id'];
        $_SESSION['username'] = $username;
        $_SESSION['role'] = $user['role'];
        header('Location: dashboard.php');
        exit;
    } else {
        echo "<p>Invalid username or password.</p>";
    }
}
?>
```

---

3 Restricting Access to Pages

`admin-dashboard.php`

```php
<?php
session_start();

// Redirect to login if not authenticated
if (!isset($_SESSION['userId'])) {
    header('Location: login.php');
    exit;
}

// Redirect if not an admin
if (($_SESSION['role'] ?? '') !== 'admin') {
    http_response_code(403);
```

```
        echo "<h2>Access Denied</h2>";
        echo "<p>You do not have permission to view this page.</p>";
        exit;
    }
    ?>
    <h2>Admin Dashboard</h2>
    <p>Welcome, <?php echo htmlspecialchars($_SESSION['username']); ?>.</p>
    <p>You have administrator access.</p>
```

This script:

1. Confirms the user is logged in.
2. Confirms that the user's role is `admin`.
3. Returns a **403 Forbidden** response if access is denied.

---

4 Conditional Access in the Interface

Sometimes you'll want to **show or hide links** depending on the user's role.

```
<?php if (($_SESSION['role'] ?? '') === 'admin'): ?>
  <p><a href="admin-dashboard.php">Admin Dashboard</a></p>
<?php endif; ?>
```

This ensures that non-admins don't even see links to pages they can't open.

Role-Based vs. Permission-Based Access

Approach	Description	Example
Role-Based Access (RBAC)	Assigns each user a predefined role with known permissions.	`role = 'admin'`
Permission-Based Access	Assigns specific permissions or flags per action.	`canApprove = true`, `canDelete = false`

Most small to mid-size PHP applications use **role-based** checks — simple and effective.

For more granular control, permissions can be stored in a separate table and checked per feature.

PHP 8.3 Features That Help

- **Strict typing** ensures roles are handled consistently as strings.
- **Match expressions** simplify conditional checks:

```php
$permissions = match ($_SESSION['role'] ?? '') {
    'admin' => include 'admin-permissions.php',
    'staff' => include 'staff-permissions.php',
    default => include 'intern-permissions.php'
};
```

- **Named arguments** and **exceptions** make database calls clearer and safer.

🧠 Quick Review

1. How does authorization differ from authentication?
2. Why is storing a user's role in `$_SESSION` helpful?
3. What HTTP status code should you use when access is denied?
4. What PHP 8.3 feature simplifies multiple role checks?
5. Describe a case where permission-based access might be better than role-based.

🎯 Practice Exercise

1. Add a `role` column to your user table if it doesn't exist.
2. Update your login code to save that role in the session.
3. Create one restricted page that only "admin" users can view.
4. Add logic to show or hide navigation links based on user role.
5. Test with at least two user accounts (intern and admin).

✅ Key Takeaways

- **Authorization** defines what an authenticated user can access.
- Roles and permissions make it easy to control visibility and access.
- Always check roles **server-side**, not just with hidden links.
- PHP 8.3 provides clean, simple tools — `match`, exceptions, and typed logic — for safe and clear access control.