Unit 4: Server-Side Programming and CRUD Operations

Unit 4 — Moodle Introduction

Unit 4 is the most technically intensive unit of this module. You will write server-side code that connects the HTML/CSS front end (Unit 2) to the MySQL database (Unit 3) to create a fully functional eLearning web application. By the end of this unit, your application must support complete CRUD operations. All code must be submitted to the Moodle Workshop for peer code review.

ID NOTE

Situated Learning (Lave & Wenger, 1991): Unit 4 activities are embedded in a realistic professional context—building a university eLearning system—rather than abstract exercises. This situated approach mirrors authentic software development practice, preparing students to contribute meaningfully to UR's actual eLearning infrastructure. The code inspection presentation (Activity 4D) further simulates the professional code review process common in industry.

Unit Aim

The aim of Unit 4 is to enable students to develop functional server-side code using PHP, Python, or Node.js that implements CRUD operations, user authentication, file upload, and automated communication for a web-based eLearning application.

Intended Learning Outcomes

- Implement full CRUD (Create, Read, Update, Delete) operations for user management and course management in a web-based eLearning application (Bloom's Level 3 — Apply).
- Apply secure coding practices including password hashing, prepared statements, input validation, and session management to protect the application against common web vulnerabilities (Bloom's Level 3 — Apply).
- Develop functionality for uploading and serving files (images, videos, PDFs, and SCORM packages) within an eLearning system (Bloom's Level 3 — Apply).
- Implement automated email or SMS notification functionality using a server-side email library or API (Bloom's Level 3 — Apply).

4.1 Choosing a Server-Side Language

Language / Framework	Characteristics and eLearning Relevance
PHP	Server-side scripting language designed specifically for web development. Powers approximately 77% of websites with a known server-side language, including Moodle, WordPress, and Facebook (originally). Excellent MySQL integration via PDO. Easy to deploy on shared cPanel hosting.
Node.js	JavaScript runtime environment that allows JavaScript to run on the server. Event-driven, non-blocking I/O makes it highly performant for real-time applications. Uses npm for package management. Ideal for

	RESTful APIs and real-time features (e.g., live chat in eLearning platforms).
Python (Django / Flask)	Python-based web frameworks. Django provides a complete MVC framework with ORM, authentication, and admin panel out of the box. Flask is a micro-framework for smaller applications. Both have excellent documentation and strong communities.

4.2 Secure Database Connectivity with PDO (PHP)

PHP Data Objects (PDO) is the recommended way to connect PHP applications to MySQL databases. PDO supports prepared statements—the primary defence against SQL injection, which is the number one web application security vulnerability (OWASP, 2021).

Definition: SQL Injection

A code injection attack in which malicious SQL code is inserted into an input field and executed by the database server. For example, entering admin'-- in a username field may bypass authentication if the query is not protected with prepared statements. SQL injection can expose, modify, or destroy all data in the database.

4.3 CRUD Operations in Practice

Operation	Implementation Guidance
CREATE (Register)	Validate input fields. Hash password using password_hash($pass, PASSWORD_BCRYPT). Insert record with prepared statement. Send welcome email.
READ (Login / Profile)	Retrieve user by email. Verify password with password_verify(). Store user ID and role in session. Display profile data.
UPDATE (Edit Profile)	Re-validate all changed fields. If password changed, re-hash. Update record with prepared statement. Do NOT allow users to change their own role.
DELETE (Deactivate)	Educational systems should rarely hard-delete users. Set is_active = 0 (soft delete) to preserve historical grade data and audit trails. Compliance with right to erasure requires complete deletion only on formal request.

4.4 File Uploads in eLearning Applications

eLearning systems must support file uploads: students submit assignments as PDFs or Word documents; instructors upload video lectures, slide decks, and SCORM packages. Secure file upload handling requires validation of file type, size, and naming.

UDL NOTE *Flexible Assessment Submission (UDL Guideline 5.2): The Moodle Assignment activity is configured to accept multiple file types (PDF, DOCX, MP4, ZIP) for each practical submission. This accommodates learners who may be more comfortable expressing their competence*

through different formats—for example, a student who prefers to submit an annotated video walkthrough of their code rather than a written report.

4.5 Sending Automated Emails and SMS

eLearning applications generate a variety of automated communications: registration confirmation emails, password reset links, assignment submission receipts, deadline reminders, and grade release notifications. In PHP, the PHPMailer library is the standard tool for sending HTML emails via SMTP. For SMS notifications, third-party APIs such as Africa's Talking (particularly relevant in Rwanda) are used.

Africa's Talking SMS API

Africa's Talking provides a REST API for sending SMS messages to Rwandan mobile numbers. This is particularly valuable for reaching students who may have intermittent internet access but reliable mobile connectivity. Registration confirmation, deadline reminders, and grade notifications can all be sent as SMS using the AT API. Documentation: https://africastalking.com/sms

Unit 4 Assessments — ABC Learning Design Framework

Activity	ABC Type	Description	Bloom's Level	Weight
User Authentication System — E-tivity 1	Practice / Production	Build a secure registration and login system using PHP/PDO or Node.js/Express. Must include: password hashing (bcrypt), prepared statements, session management, CSRF protection on forms, and role-based page protection. Submit code via Moodle Workshop for peer code inspection.	Apply (L3)	15% (ICA)
Course Management Module — E-tivity 2	Production / Collaboration	Extend your application with instructor-facing CRUD operations for courses: create a new course, view course list, edit course details, manage enrolments, and soft-delete a course. Includes file upload for course banner image.	Apply / Create (L3–L6)	10% (ICA)
File Upload Feature — E-tivity 3	Practice / Production	Implement secure file upload functionality for student assignment submissions. The system must validate MIME type, restrict file size to 50 MB, generate a random storage	Apply (L3)	5% (ICA)

		filename, and store the upload path in the database.		
Code Inspection Presentation	Discussion / Collaboration	Present your developed code to a code inspection panel (lecturer and two peers). Explain and defend your implementation choices for authentication, SQL injection prevention, and role-based access control. Duration: 10 minutes presentation + 5 minutes Q&A.	Evaluate / Create (L5–L6)	10% (ICA)

End-Unit Assessment 1: Moodle Quiz — Server-Side Programming & Security

Ready-to-Use Moodle Quiz — Copy & Paste Questions

10 questions. Set: Attempts = 2, Grading = Highest, Shuffle = Yes, Feedback = Immediate.

Question & Answer	Notes
Q1. A PHP developer writes the following query: $query = "SELECT * FROM students WHERE email = '" . $_POST['email'] . "'"; What security vulnerability does this introduce? ✓ A) SQL Injection — user input is concatenated directly into the SQL string B) XSS — the output is not escaped before being displayed in HTML C) CSRF — the form does not include a CSRF token D) Insecure File Upload — the input is not validated as an email Feedback: Concatenating user input into SQL queries allows an attacker to inject malicious SQL (e.g., ' OR '1'='1). The correct approach is to use PDO prepared statements with parameterised queries.	
Q2. Which PHP function correctly verifies a bcrypt-hashed password against a user-supplied password string? ✓ A) password_verify($userInput, $storedHash) B) md5($userInput) === $storedHash C) sha1($userInput) === $storedHash D) hash_equals($userInput, $storedHash) Feedback: password_verify() is the correct function. It is timing-attack-safe and handles the salt embedded in the bcrypt hash. MD5 and SHA1 are cryptographically broken and must not be used for password storage.	
Q3. An instructor attempts to access a student management page directly via its URL, bypassing the login screen. What server-side mechanism should prevent this? ✓ A) Session-based role check at the top of every protected page B) Hiding the URL from the browser C) Using JavaScript to redirect unauthenticated users D) Setting the page file permissions to read-only Feedback: Client-side protections (JavaScript redirects, hidden URLs) can be bypassed. Every protected page must perform a server-side session check: if (!isset($_SESSION['user_id'])) { header('Location: /login.php'); exit; }	
Q4. Why must the MIME type of an uploaded file be validated using PHP's finfo extension rather than trusting the $_FILES['type'] value? ✓ A) $_FILES['type'] is sent by the browser and can be spoofed by an attacker B) $_FILES['type'] is	

always NULL for binary files C) finfo is required by the SCORM standard for file uploads D) The browser does not send a MIME type for ZIP files Feedback: $_FILES['type'] reflects what the client's browser reports, which can be manipulated. finfo::file() inspects the actual file bytes on the server, making it a reliable server-side MIME type check.	
Q5. What is a CSRF (Cross-Site Request Forgery) attack and how does a CSRF token prevent it? ✓ A) An attack where a malicious website tricks a logged-in user's browser into making an unwanted request to the application; a CSRF token is a secret, session-tied value submitted with every form that the server validates before processing the request B) An attack that injects malicious JavaScript into the page; a CSRF token encrypts the JavaScript C) An attack that intercepts network traffic; a CSRF token encrypts the request D) An attack that guesses session IDs; a CSRF token replaces the session ID Feedback: CSRF exploits the browser's automatic inclusion of cookies. A synchroniser token embedded in the form and validated server-side ensures that requests originate from the legitimate application, not a third-party site.	
Q6. In PHP session management, which session configuration setting should be applied to prevent JavaScript from accessing the session cookie (mitigating XSS-based session hijacking)? ✓ A) session.cookie_httponly = 1 B) session.cookie_secure = 1 C) session.use_strict_mode = 1 D) session.gc_maxlifetime = 0 Feedback: The HttpOnly flag (session.cookie_httponly) prevents JavaScript from reading the session cookie, blocking the most common XSS-based session hijacking vector. session.cookie_secure should also be set to restrict the cookie to HTTPS connections.	
Q7. A student submits an assignment file. The PHP upload handler stores the file using its original filename. What security risk does this create? ✓ A) Path traversal and PHP code execution if the filename is malicious (e.g., ../../config.php or shell.php) B) The file will not be accessible in the Moodle interface C) The database will reject the file path if it contains spaces D) It violates SCORM file naming conventions Feedback: Filenames supplied by users can include path traversal sequences (../) or malicious extensions (.php). Always generate a cryptographically random storage filename (e.g., bin2hex(random_bytes(16)) . '.pdf') and never use the original filename for storage.	
Q8. Why should educational systems perform a 'soft delete' (setting is_active = 0) rather than a hard DELETE when removing a student account? ✓ A) To preserve historical grade and submission data linked to the student's foreign keys, and to comply with audit trail requirements B) Because MySQL prevents deletion of rows referenced by foreign keys C) To comply with SCORM 1.2 data retention requirements D) Because the PHP unset() function does not remove database rows Feedback: Hard-deleting a student row would cause referential integrity failures (or cascading deletions) in linked tables (submissions, grades, enrolments). Soft deletion preserves data integrity and audit trails while removing the account from active use.	
Q9. Which Africa's Talking API endpoint should a UR developer call to send an SMS deadline reminder to students? ✓ A) POST https://api.africastalking.com/version1/messaging B) GET https://api.africastalking.com/version1/messaging/send C) POST https://api.twilio.com/2010-04-01/Messages D) GET https://api.africastalking.com/sms Feedback: Africa's Talking SMS API uses a POST request to the /version1/messaging endpoint with parameters: username, to, message, and from. A GET request cannot send message body data securely.	

Q10. [Short Answer] Explain the difference between a session and a cookie in web applications, and describe how PHP sessions are used to maintain authenticated user state across multiple page requests. Model Answer: A cookie is a small piece of data stored in the user's browser and sent with every HTTP request to the server. A session is a server-side data store identified by a unique session ID, which is stored in a cookie on the client side. When a user logs in, PHP creates a session (session_start()), stores the user's ID and role in the $_SESSION superglobal (a server-side array), and sends the session ID as a cookie to the browser. On every subsequent page request, the browser sends the session ID cookie back to the server, which retrieves the corresponding session data. The application checks $_SESSION['user_id'] and $_SESSION['role'] to verify authentication and authorisation without requiring the user to log in on every page.	

End-Unit Assessment 2: Code Inspection Presentation Rubric

Criterion	Distinction (75–100%)	Merit (60–74%)	Pass (50–59%)	Fail (<50%)
Technical Accuracy	All implementation choices are technically correct and demonstrate a thorough understanding of secure coding principles.	Most choices technically correct; one or two minor inaccuracies.	Core functionality present but with identifiable security or logic flaws.	Significant technical errors; core CRUD or security features absent or non-functional.
Security Implementation	bcrypt hashing, PDO prepared statements, CSRF tokens, session management, and role checks all correctly implemented and explained.	Most security features correctly implemented; one security control missing or incorrectly explained.	Some security features present; SQL injection or session management inadequately addressed.	Critical security features (prepared statements, password hashing) absent or broken.
Clarity of Explanation	Explanations are clear, structured, and use accurate technical vocabulary throughout the presentation.	Mostly clear with minor unclear passages; vocabulary generally accurate.	Explanation is partially clear; some technical terms misused.	Explanation is unclear or inaccurate; cannot defend implementation choices.
Response to Questions	Responds to all questions accurately and confidently; demonstrates understanding beyond presented code.	Answers most questions correctly; struggles with one or two deeper questions.	Answers some questions; significant difficulty with follow-up questions.	Unable to answer most questions about own code.