HIKING SPOTS CMS

COSC 213: Web Development Final Project Report

Team: Nery, Simon, Daniel

Date: November 29, 2025

Project: Content Management System (CMS)

1. EXECUTIVE SUMMARY

The following report describes the development of "Hiking Spots CMS," a dynamic web application using the LAMP stack: Linux, Apache, MySQL, and PHP. The platform will be used as a community hub for outdoor enthusiasts to discover, share, and discuss hiking trails. The system has secure user authentication, full CRUD operations for blog posts, and an interactive public commenting system.

The application successfully meets all the core requirements of Project 2, Content Management System, and implements two advanced features: a public commenting system and user-specific content management with role-based access control.

Our project implements a number of layers of security, following the best practices of the industry. We used PHP's official documentation regarding password hashing to implement bcrypt encryption on all user passwords. Following OWASP guidelines on SQL injection prevention, we utilized prepared statements with parameter binding throughout the application. XSS attacks are mitigated by sanitizing all user-generated content with htmlspecialchars(). Session management follows the recommendations from MDN Web Docs on how to maintain user authentication across multiple pages securely.

2. APPLICATION ARCHITECTURE

2.1 System Overview

The application follows a three-tier architecture design pattern:

• Presentation Layer (Client-Side): HTML provides the semantic structure, CSS does the styling with a customized design that complements the project's green color scheme, and minimal JavaScript for some basic interactions, whereas logic is mostly handled at the server-side via PHP.

• Application Layer (Server-Side): PHP performs all business logic, maintaining user sessions for authentication, and utilizes prepared statements to secure databases. This layer handles all form processing and validation of data.

• Data Layer: The MySQL database stores all application data in a normalized schema consisting of three tables: users, posts, comments. There are foreign key constraints to maintain referential integrity, and related data is automatically deleted using the CASCADE rules.

2.2 File Structure

Core PHP Files: config.php (database connection, session management), index.php, home page with listing of all posts, article.php: view single post with comments, login.php: User authentication, logout.php: kill session, new_article.php: create post, edit_article.php: edit post, dashboard.php: User manage his posts.

Database files: schema.sql contains a full database schema with CREATE TABLE statements, along with sample data INSERT statements. It includes two demo user accounts, an admin and a regular user, with pre-hashed passwords. (user123 for user, admin123 for admin)

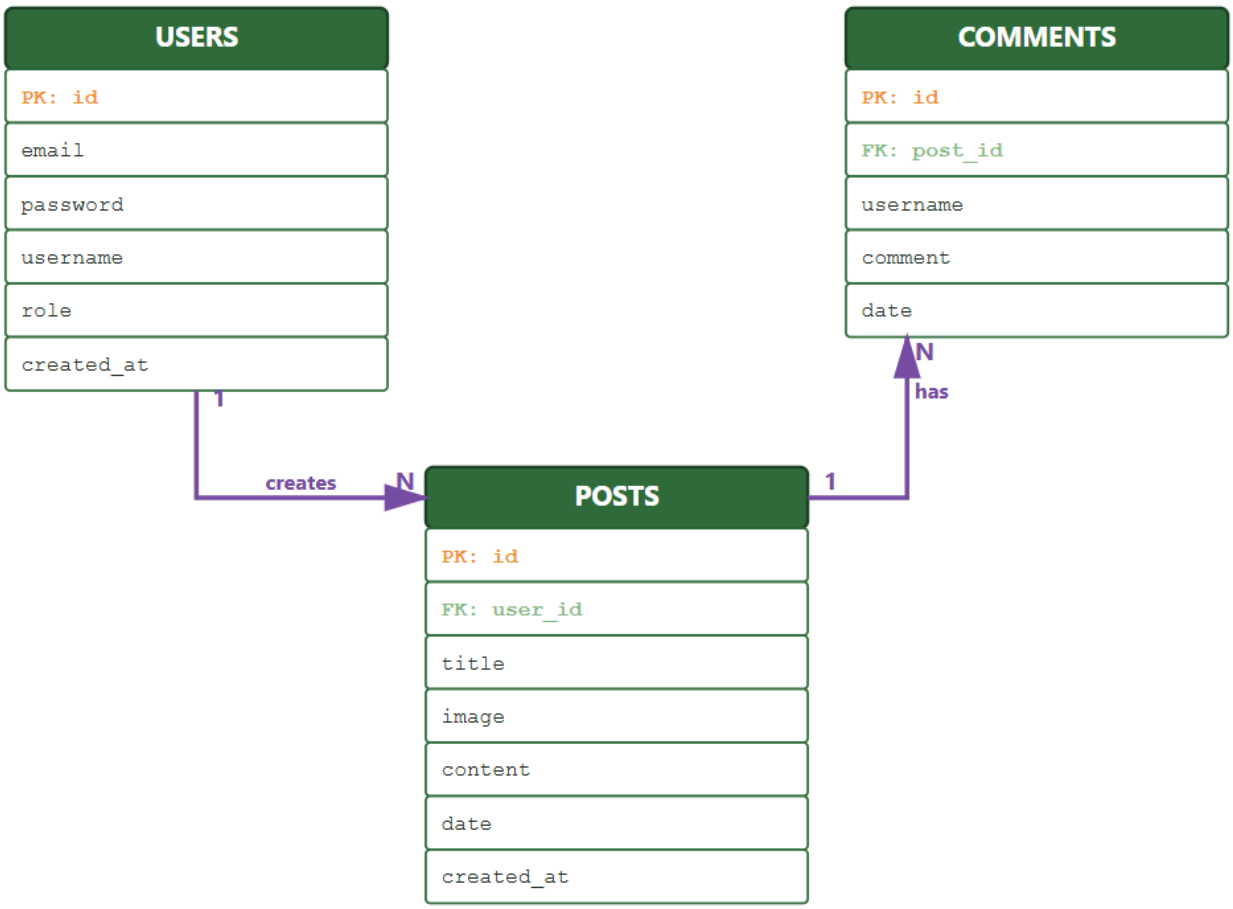
Assets: style.css includes all styling, including responsive design breakpoints; logo.png is for site branding.

2.3 Security Implementation

- The application implements several layers of security:
- Password Security: All passwords are hashed using PHP's password_hash() function with the bcrypt algorithm. Login verification uses password_verify() to compare submitted passwords against stored hashes. No plain-text passwords are ever stored in the database or transmitted.
- SQL Injection Prevention: Every database query utilizes prepared statements with parameter binding through bind_param(). User input will never be concatenated directly into the SQL strings; therefore, SQL injection is prevented.
- XSS Protection: All output is sanitized with htmlspecialchars() before rendering, which prevents any malicious script injection through user-submitted content.
- Authentication & Authorization: Session-based authentication keeps track of the users who have logged in. Protected routes automatically redirect unauthorized users to login. Ownership is validated while editing and deleting a post. Users can only edit or delete their posts, unless admins.

3. DATABASE SCHEMA

3.1 Entity-Relationship Design



3.2 Table Specifications & Key Decisions

Users Table: This table stores information about registered users with their bcrypt-hashed passwords. The email field has a UNIQUE constraint to forbid duplicate accounts creation. The role field is designed for role-based access. It may have values either 'admin' or 'author', with the default 'author'. An admin can manage all posts across the website, while an author is able to manage only their posts.

Table Posts: Contains all the articles about hiking spots, and it also has a foreign key relationship to the users table via user_id. The image field accepts optional URL. ON DELETE CASCADE ensures that if a user is deleted, all their posts will be deleted automatically. Posts are ordered by date in descending order (newest first).

Comments Table: Links to posts via foreign key post_id. Employs a username field instead of user_id to accommodate public comments by non-registered users (so anyone can participate without an account). ON DELETE CASCADE ensures comments are deleted upon removal of the parent post.

Normalization: The database is normalized to 3NF, which reduces redundancy and ensures data integrity. All attributes are atomic in value (1NF), there are no partial dependencies (2NF), and no transitive dependencies (3NF).

4. IMPLEMENTED FEATURES

4.1 Core Requirements (All Met)

- Public Blog - index.php: Lists all hiking spots with images, title, author, date, and a summary of the whole text. Posts are loaded dynamically from the database and sorted chronologically, newest first.
- Full Article View: article.php displays full post content including complete descriptions, images, metadata, and their respective comments. Route based on URL parameters allows direct linking - article.php?id=X
- User Authentication; a secure login system with email and password verification. Sessions monitor active users through page changes. Password verification through Industry-standard bcrypt hashing.
- Admin Panel: dashboard.php allows for a dedicated admin panel that has all of the users' posts listed in table format, with buttons for edit and delete. Confirmation dialogs prevent accidental deletions.
- Article Creation/Editing: A new post is created by new_article.php with title, image URL, and content fields. Posts are edited through edit_article.php. The form validates required fields and auto-fills the existing data when editing.
- User Roles: The system differentiates between the 'admin' and 'author' roles. Admins will view all posts of all users on their dashboard and can edit or delete any post. Authors can see only their own posts and can only edit/delete posts they created.

4.2 Advanced Features Implemented

1. Comments: A full-featured, publicly commenting system allows users to comment on hiking spots without needing to log in. Comments show up with usernames and timestamps, comment counts are shown correctly, and comments persist in the database. Comments are deleted automatically when their parent post has been deleted using the CASCADE rules.

2. User-Specific Management: Advanced authorization system that ensures a user can only modify their content. SQL queries filter by user_id, employing WHERE statements. Ownership validation prevents different users manipulating other users data. Admin override allows admins to manage all content for moderation purposes.

5. TEAM CONTRIBUTIONS

Team Member	Overall Contributions
Nery	**Frontend Design & User Experience** Complete visual identity, including full green color scheme. Designed a responsive CSS layout system built on custom properties and media queries. The card-based interface for article preview is implemented as follows: header and footer styling using gradient backgrounds; ensured mobile-friendly design using breakpoints from most widespread screen sizes. Conducted user experience research and testing.
Simon	**Database Architecture & Interactions** Designed the complete database schema, including table structures, relationships, and constraints. Developed initial CRUD operations implementation logic. Created a localStorage prototype for initial testing, later replaced with MySQL. Performed extensive testing across all features to identify bugs. Fixed JavaScript integration issues. Contributed to debugging SQL query problems. Assisted with documentation and README creation.
Daniel	**Backend Development & Security** Wrote all the PHP server-side logic for every page: config.php, index.php, article.php, login.php, dashboard.php, new_article.php, edit_article.php, logout.php. Implemented the full authentication system, including bcrypt password hashing with password_hash() and password_verify(). Optimized all SQL queries for performance and security. Implemented prepared statements using parameter binding throughout the application. Created a session management system. Developed authorization logic with role-based access control. Wrote comprehensive project documentation. Integrated frontend HTML with backend PHP functionality.

6. CHALLENGES & SOLUTIONS

Challenge 1: Migrating from localStorage to MySQL

Problem: The very first prototype used client-side localStorage to store data about hiking spots. This had some critical limitations: data would not persist across browsers, multiple users could not share data, and there was no server-side validation/security.

Solution: Completely redesigned the application architecture to use a MySQL database backend. Created a normalized relational schema with proper foreign key relationships. Implemented PHP server-side logic to handle all data operations. This migration enabled multi-user functionality, data persistence, server-side validation, and proper security measures.

This will create a robust, scalable data layer that will efficiently support multiple concurrent users using reliable data persistence.

Challenge 2: Implementing Secure Password Storage

Problem: Password security is complex but critical. Passwords needed to be stored in a way that even database administrators couldn't read them, while still allowing login verification.

Solution: Researched PHP password security best practices and implemented the industry-standard bcrypt algorithm using PHP's built-in password_hash() function. During registration or password changes, passwords are hashed before database storage. During login, password_verify() compares the submitted password against the stored hash. Each hash automatically includes a unique salt to prevent rainbow table attacks. (A Rainbow Table Attack is a cryptographic attack method that uses precomputed tables of hash values to quickly reverse-engineer plaintext passwords from their hashed counterparts).

Result: Industry-standard password security-no plain-text passwords ever stored or transmitted.

Challenge 3: Preventing SQL Injection Attacks

Problem: Concatenation of User Input Directly into SQL Query Strings leads to critical security vulnerabilities. A malicious attacker can inject malicious SQL code through form inputs, which could target any data within the database for access, modification, or deletion.

Solution: Converted all database queries in the application to MySQLi prepared statements. Lastly, bind user input to queries using bind_param() with correct type specification: i for integers and s for strings. Bound parameter values are treated by the database engine as literal input data, not as executable code, and thus, SQL injection is entirely prevented.

Code Example: $stmt = $conn->prepare('SELECT * FROM posts WHERE id = ?'); $stmt->bind_param('i', $post_id);

Result: Complete protection against SQL injection attacks throughout the application.

Challenge 4: Session Management Across Pages

Problem: HTTP is stateless, and each page request is entirely independent. Maintaining the user login state across multiple pages required a reliable session system to keep track of who is authenticated.

Solution: Centralize session handling in config.php, included on every page. Automatically start session on page load, if session is not active. On login, store credentials into session variables:$_SESSION['user_id'], $_SESSION['username'],$_SESSION['role']. Check the existence of these session variables on protected pages before displaying content. Logout.php destroys session data and removes cookies.

Result: Seamless authentication state management throughout the application.

Challenge 5: User Authorization & Content Ownership

Problem: After authentication, users were able to edit or delete other users' posts with simply altering URL parameters - for example, changing edit_article.php?id=5 to id=6.

Solution: I added a user_id foreign key to the posts table to connect each post with its owner. I have added a role field to the users table to differentiate between 'admin' and 'author' roles. All SQL queries are updated to check for ownership. Queries executed by regular users include 'WHERE

user_id = ?' clauses while the admin queries select all posts. Edit and delete functions check the ownership before editing is allowed.

Code Example: $stmt = $conn->prepare('SELECT * FROM posts WHERE id = ? AND user_id = ?'); $stmt->bind_param('ii', $post_id, $user_id);

Result: Users can only edit their own content, with admin override for moderation.

7. CONCLUSION & ACHIEVEMENTS

Hiking Spots CMS successfully fulfills all core requirements for Project 2 (CMS) and implements two advanced features. The application demonstrates a solid understanding of the LAMP stack, secure web development practices, and database design principles.

Key Achievements:

• Fully functional LAMP stack web application; all required features are functional

• Secure user authentication system with industry-standard bcrypt password hashing

• Comprehensive CRUD for blog posts management

• Role-based access control differentiating between admin and author permissions

• Protection against common web vulnerabilities (SQL injection, XSS attacks)

• Clean, responsive user interface optimized for multiple device sizes

• Normalized database design eliminating data redundancy

• Well-documented code base, with comments and structure.

Skills Developed Through This Project:

• PHP server-side programming and business logic implementation

MySQL database design, normalization, and query optimization

• User authentication and session management systems

• Security best practices including password hashing and SQL injection prevention

• Frontend-backend integration and full-stack development

• Git version control and team collaboration workflows

• Technical documentation and project reporting

This project provided hands-on experience in all layers of web application development, from database design through server-side logic to the implementation of user interface, with an emphasis on security best practices throughout.

8. REFERENCES

1. PHP Documentation: Password Hashing - https://www.php.net/manual/en/function.password-hash.php

2. PHP Documentation: MySQLi Prepared Statements - https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php

3. OWASP: SQL Injection Prevention - https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

4. OWASP: Cross-Site Scripting (XSS) - https://owasp.org/www-community/attacks/xss/

5. W3Schools: PHP MySQL Tutorial - https://www.w3schools.com/php/php_mysql_intro.asp

6. MDN Web Docs: HTTP Sessions - https://developer.mozilla.org/en-US/docs/Web/HTTP/Session