MSCS-535: Assignment-14: Lab report: Secure Website using JavaScript and PHP with Database Connection

---

Sandesh Pokharel

University of the Cumberlands

MSCS-535 Secure Software Development

Brandon Bass

April 20, 2025

Introduction

The goal of this assignment is to provide a secure coding example of a web application that uses **JavaScript (frontend)** and **PHP (backend)** to interact with a **MySQL database**. The focus is on implementing strong security practices such as input validation, password hashing, protection against SQL injection, and secure database access.

This report explains the overall system architecture, folder structure, technologies used, and security techniques applied throughout the app. It also includes code walkthroughs, test user creation, and screenshots to demonstrate functionality.

Technology Stack

Component	Technology Used
Frontend	HTML, JavaScript
Styling	Bootstrap 5
Backend	PHP 8
Database	MySQL

Server	PHP built-in server

Folder Structure

secure-login-app/

├── index.html → Main login form with Bootstrap styling

├── login.php → Backend PHP script (secure login logic)

├── js/

│ └── validate.js → JavaScript input validation

└── assets/css/

└── style.css → Optional custom styles

Frontend: Secure Form with Validation

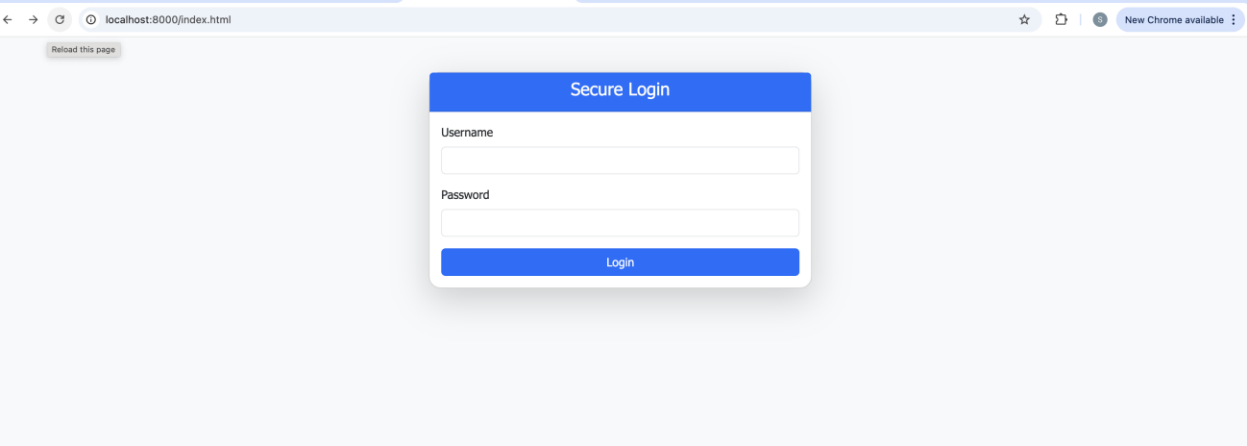
The login form is built using HTML and styled with Bootstrap 5 for a professional look.

JavaScript is used for real-time input validation before the form is submitted.

Key security practice:

- The form is validated via an **external JavaScript file** (validate.js) to prevent empty input fields and reduce unnecessary server requests.

📸 **Screenshot:** index.html displayed in the browser (showing the login form)



Backend: Secure PHP Logic

The login.php file contains all the backend logic for connecting to the database and authenticating users securely.

🔐 ***Security Features Implemented:***

1. **PDO with Prepared Statements**

Prevents SQL injection by using placeholders and binding user input safely.

2. **Password Hashing**

User passwords are stored using password_hash() and checked with password_verify().

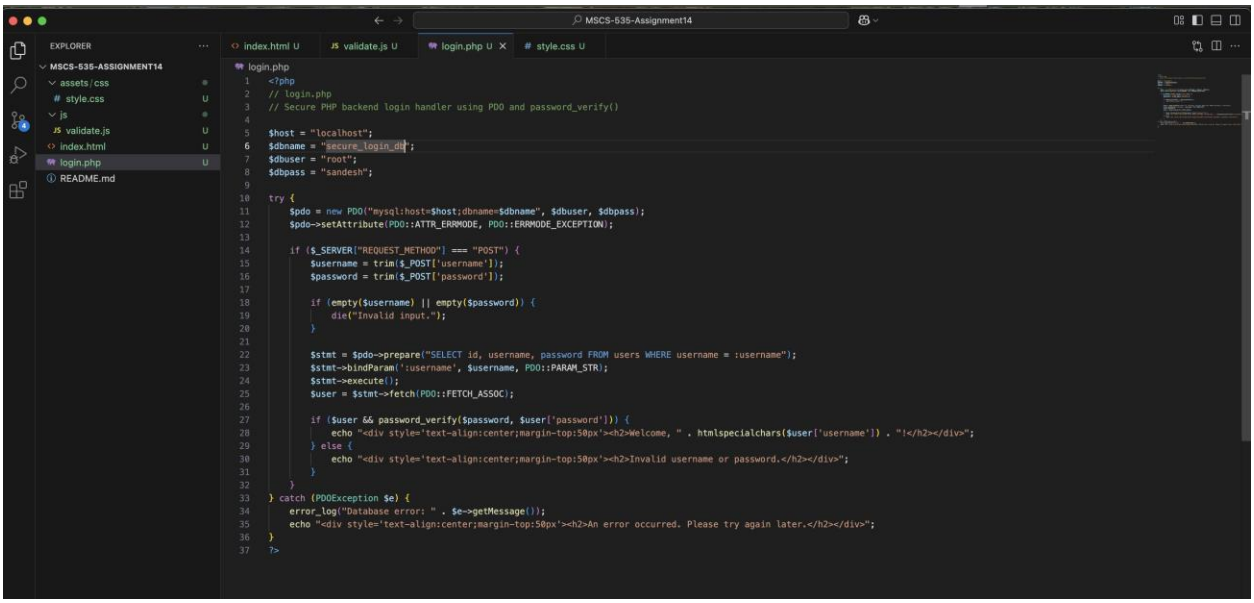
3. **Input Sanitization**

All input is sanitized using trim() and htmlspecialchars() to prevent XSS.

4. **Error Handling**

Errors are logged using error_log() instead of being shown to the user.

📸 **Screenshot:** PHP code from login.php and sample successful login message



Database Setup and Test User

The application uses a MySQL database named secure_login_db with a table called users.

SQL to create the table:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

A test user was inserted with a securely hashed password using PHP's interactive shell (php -a):

echo password_hash("test1234", PASSWORD_DEFAULT);

SQL to insert:

```
INSERT INTO users (username, password) VALUES (
  'admin',
  '$2y$10$Niv6QOSdGlZqAMTyMbkmKuWEm9rpnMNsZLS5UoIkMi5ETsKML7rHS'
);
```

📸 **Screenshot:** Terminal output of MySQL database with select * from users

```
mysql> CREATE DATABASE secure_login_db;
Query OK, 1 row affected (0.01 sec)

mysql> USE secure_login_db;
Database changed
mysql> CREATE TABLE users (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     username VARCHAR(100) NOT NULL UNIQUE,
    ->     password VARCHAR(255) NOT NULL
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO users (username, password) VALUES (
    ->   'admin',
    ->   '$2y$10$Niv6QOSdGlZqAMTyMbkmKuWEm9rpnMNsZLS5UoIkMi5ETsKML7rHS'
    ->
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> show * from users;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '* from users' at line 1
mysql> show tables;
+---------------------------+
| Tables_in_secure_login_db |
+---------------------------+
| users                     |
+---------------------------+
1 row in set (0.01 sec)

mysql> show * from users
    -> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '* from users' at line 1
mysql> select * from users;
+----+----------+--------------------------------------------------------------+
| id | username | password                                                     |
+----+----------+--------------------------------------------------------------+
|  1 | admin    | $2y$10$Niv6QOSdGlZqAMTyMbkmKuWEm9rpnMNsZLS5UoIkMi5ETsKML7rHS |
+----+----------+--------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> update users set password = '$2y$12$UHIsNKH.ZvQzrtZBbzaZi.soJBEV06YaY3yFzCxxgl/pqesqAB7Fa' where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> clear
mysql>
```

Testing and Results

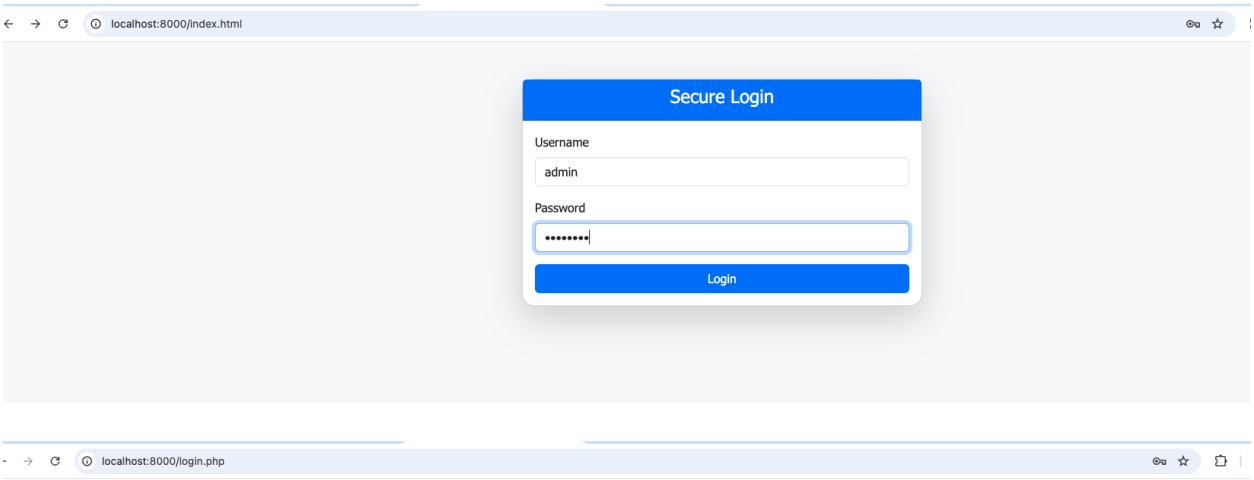
Once the PHP server was started using:

php -S localhost:8000

The login form was tested by submitting correct and incorrect credentials.

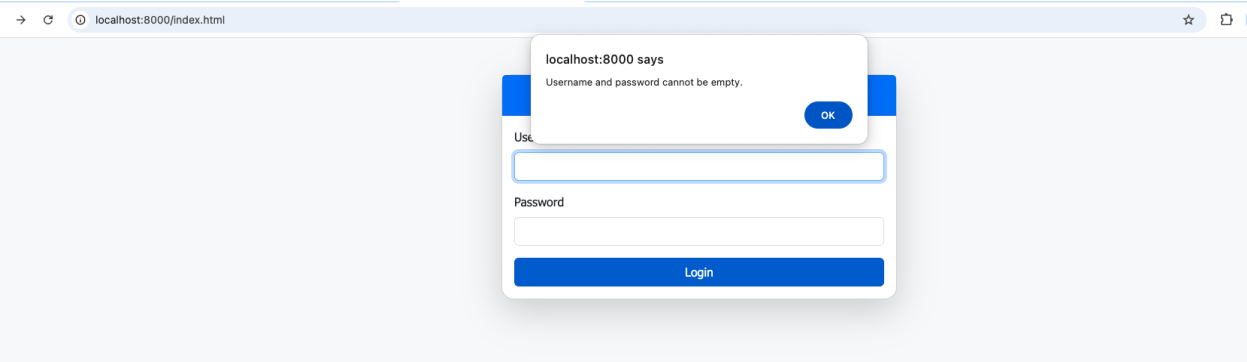
- ✅ Correct username and password → Login successful
- ❌ Incorrect password → Error message shown
- ❌ Empty fields → Caught by frontend validation

📸 **Screenshot Placeholder:** Successful login response



📸 **Screenshot Placeholder:** Login error due to incorrect password

📸 **Screenshot Placeholder:** JavaScript alert for empty input



Security Summary

Security Measure	Implementation Detail
SQL Injection Prevention	PDO + prepared statements in PHP
Password Protection	password_hash() + password_verify()
XSS Mitigation	Input sanitized with htmlspecialchars()
Input Validation	JavaScript client-side validation before form submission
Error Handling	No raw errors exposed to users; errors logged with error_log()

Conclusion

This assignment demonstrates how even a simple web application can be built with solid security foundations by applying secure coding practices throughout the stack. Using JavaScript for frontend validation, PHP for secure database access, and MySQL for data storage, we built a complete mini login system that avoids common vulnerabilities like SQL injection and insecure password storage.

This structure and implementation can be scaled into more complex applications while maintaining these best practices.