

Student's Name

Department, Institutional Affiliation

Course Name and Number

Professor's Name

Due Date

Building a Secure User Authentication System with PHP and MySQL

Introduction

User authentication is a significant feature in many web applications for ensuring that people can access particular functionalities or resources. Building a secure user authentication system using PHP and MySQL comprises components like registration, login, password hashing session management, and security considerations.

Setting Up a Development Environment

Building a secure user authentication system with PHP and MySQL requires setting up a development environment. This environment should include PHP, MySQL, a web server, and a preferred code editor (Visual Studio code).

Requirements

- **PHP:** PHP should be installed on the web server or computer.
- **MySQL:** MySQL should be installed to store user data.
- **Web Server:** A server like Apache or Nginx is needed to host the php application.
- **Essential Understanding:** A basic understanding of PHP and MySQL is required.

PHP and MySQL can be installed with the web server by installing MAMP in the Mac operating system or XAMMP in Windows/Linux.

Creating a New Project

Once MAMP is installed and running, create a new project for building a secure user authentication system.

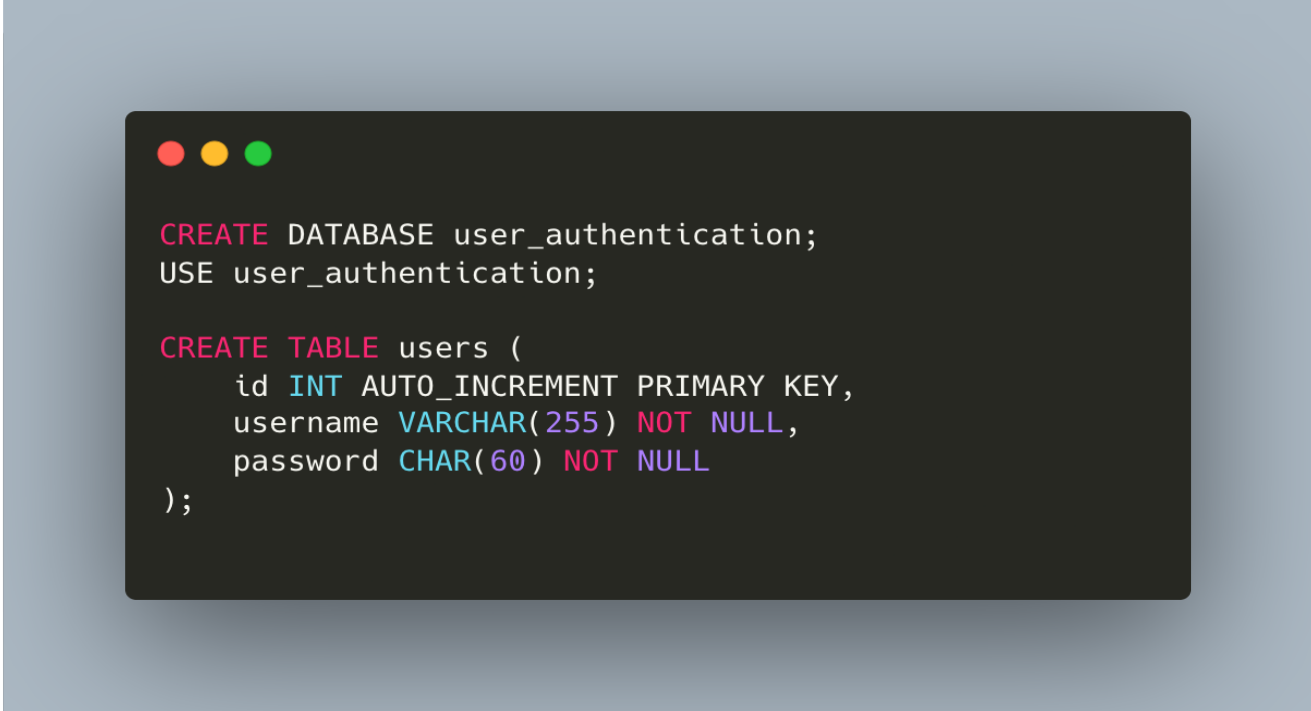
- Document Root: By default, MAMP uses the htdocs directory as the document root. It can be modified in the MAMP preferences, if needed.
- Project Directory: Create a new folder in the htdocs directory for the project. It is where your PHP files and assets will be stored.
- PHP Files: Open the preferred code editor to create PHP files for your user authentication system within your project directory.
- Database: For MySQL, phpMyAdmin is recommended and is included with MAMP.
- Open the web browser and go to <http://localhost/phpMyAdmin>. It provides a platform for creating a new database for the project and managing tables for user data.

By following these steps, it is easy to create a development environment with MAMP and create a new project to begin building your secure user authentication system. It is necessary to place PHP files in the project directory and use phpMyAdmin to manage your MySQL database. This environment will provide a local platform for development and testing before deploying the application to a live web server.

Setting up a Database

The initial step in creating any system is setting up a database to store user data. The database should have a table named “users” in which to keep the usernames and hashed

passwords. The user's table should have three columns: 'id,' 'username,' and 'password.' The 'id' field should be an auto-incrementing integer, the 'username' should be a string storing username, and the 'password' should be a string holding an encrypted password.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal displays the following SQL code:

```
CREATE DATABASE user_authentication;
USE user_authentication;

CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL,
  password CHAR(60) NOT NULL
);
```

Registration

The user should provide a username and password in the registration step and insert the data into the database after hashing the password.

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["username"];
    $password = $_POST["password"];

    $hashed_password = password_hash($password, PASSWORD_BCRYPT);

    $conn = new mysqli("localhost", "username", "password", "user_authentication");

    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    $stmt = $conn->prepare("INSERT INTO users (username, password) VALUES (?, ?)");
    $stmt->bind_param("ss", $username, $hashed_password);

    if ($stmt->execute()) {
        echo "Registration successful!";
    } else {
        echo "Registration failed. Please try again.";
    }

    $stmt->close();
    $conn->close();
}
?>
```

The above code snippet checks if the request method is POST, indicating that the user has submitted the form. It gets the username and password from the form data and then hashes the password using the `password_hash` function with the `password_bcrypt` algorithm.

Login

Users will provide their username and password, which should be verified against the hash stored in the database.

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["username"];
    $password = $_POST["password"];

    $conn = new mysqli("localhost", "username", "password", "user_authentication");

    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    $stmt = $conn->prepare("SELECT id, password FROM users WHERE username = ?");
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $stmt->bind_result($user_id, $hashed_password);
    $stmt->fetch();
    $stmt->close();

    if (password_verify($password, $hashed_password)) {
        session_start();
        $_SESSION["user_id"] = $user_id;
        header("Location: dashboard.php");
    } else {
        echo "Login failed. Please check your username and password.";
    }

    $conn->close();
}
?>
```

The above code snippet checks if the request method is POST, indicating a login attempt. It retrieves the username and password from the form data. Then, it connects to the MySQL database and recovers the hashed password for the provided username. The password verify function compares the user's input password with the hashed password from the database. If they match, it creates a session, stores the user's ID, and redirects them to a dashboard or home page.

Password Hashing and Verification

In both the registration and login processes, the password hash function securely hashes the passwords before storing them in the database. Additionally, the 'password_verify' function

compares the user's input password with the stored hash for authentication. This ensures that the passwords are stored securely and are resistant to common attacks.

Session Management

Session management is essential for keeping users logged in and providing a personalized experience. PHP uses sessions to achieve this.



```
<?php
session_start();

if (isset($_SESSION["user_id"])) {
    $user_id = $_SESSION["user_id"];
} else {
    // User is not logged in, redirect to the login page
    header("Location: login.php");
}

?>
```

The dashboard.php file starts with the session and checks if the user_id is set in the session. If present, the user is considered logged in and is provided a personalized dashboard. If not present, the user is redirected to the login page.

Security Considerations

Building a secure user authentication system involves various security concerns. Here are some significant points to be considered:

- **Password Policies:** Implement password policies that encourage users to create strong passwords.

- **SQL Injection:** Use prepared statements to prevent SQL injection attacks when interacting with the database.
- **Session Security:** Protect session data and use HTTPS to secure data in transit.
- **Brute Force Protection:** Implement measures to limit the number of login attempts to prevent brute force attacks.
- **User Registration Verification:** Consider email verification or CAPTCHAs to prevent automated registration.
- **Password Reset:** Provide a secure method for users to reset their passwords.
- **User Data Sanitization:** Always sanitize and validate user input to prevent cross-site scripting (XSS) and other attacks.
- **Two-Factor Authentication (2FA):** Implement 2FA for added security, requiring users to enter a temporary code sent to their mobile device or email after entering their password.

Conclusion

Building a secure user authentication system with PHP and MySQL is an essential step in ensuring the security and integrity of web applications. The best practices outlined above assist in creating a robust authentication system that protects user data and maintains the trust of the system users. It covers the essential aspects of building a secure user authentication system, from setting up the database and implementing registration and login functionalities to addressing various security considerations. It is necessary to prioritize security during system development and remain informed about the latest security practices and threats to ensure that the system remains secure and that user data is protected. The security aspect is an ongoing process, and it is

essential to keep the system up to date with the latest security patches and practices to maintain a safe and reliable user authentication system.

References

- Aggarwal, P. K., Sharma, R., Khare, R., & Singh, S. (2023, May). E-commerce Application using PHP and Web Development: A Review. In *2023 International Conference on Disruptive Technologies (ICDT)* (pp. 755-758). IEEE.
- Hassan, M. A., Pavel, M. I., Muhtasim, D. A., Kamal Hussain Shahi, S. M., & Rumpa, F. I. (2022). Enhanced Security of User Authentication on Doctor E-Appointment System. In *Innovative Data Communication Technologies and Application: Proceedings of ICIDCA 2021* (pp. 47-63). Singapore: Springer Nature Singapore.
- Hussain, M. A., Hussien, Z. A., Abduljabbar, Z. A., Ma, J., Al Sibahee, M. A., Hussain, S. A., ... & Jiao, X. (2022). Provably throttling SQLI using an enciphering query and secure matching. *Egyptian Informatics Journal*, 23(4), 145-162.
- Kurien, A. T. J., Mathew, S. A., & Mana, S. C. (2022, April). Development of PHP and MySQL based Digital Asset Management System for Secure Organizations. In *2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 1859-1863). IEEE.
- Sotnik, S., Manakov, V., & Lyashenko, V. (2023). Overview: PHP and MySQL Features for Creating Modern Web Projects.