**APEX PLANET SOFTWARE PVT LTD**

**Security Enhancements Documentation:**

* **Objective**

Strengthen application security by implementing best practices to protect against common web vulnerabilities, including:

* SQL Injection
* Malicious Input
* Unauthorized Access
* **Application Security Upgrades**

The following key security features were successfully implemented to protect against common web vulnerabilities :

| **Feature** | **Description** |
| --- | --- |
| * **Prepared Statements** | Protects against SQL Injection by using parameterized queries |
| * **Form Validation** | Validates user input on server and client side |
| * **User Roles & Access Control** | Restricts access to sensitive actions based on user roles |

**1. Prepared Statements :**

**What?**

All SQL queries now use **PDO Prepared Statements**.

**Why?**

Prevents **SQL Injection** by separating user inputs from SQL logic.

**Where?**

Implemented in all critical files:

* login.php
* register.php
* create.php
* edit.php
* delete.php

**Example:**

$stmt = $conn->prepare("SELECT \* FROM users WHERE username = ?");

$stmt->execute([$username]);

**2. Form Validation :**

**Server-Side Validation**

**What?**

All incoming form data is validated in **PHP** before processing.

**Why?**

Ensures **data integrity** and blocks malicious or malformed input.

**Where?**

Validation included in:

* register.php
* login.php
* create.php
* edit.php

**Example:**

if (empty($title)) { $errors[] = "Title is required."; }

elseif (mb\_strlen($title) > 255) { $errors[] = "Title must be less than 255 characters.";}

**Client-Side Validation**

**What?**

All forms use **HTML5 attributes** and **JavaScript** for real-time validation.

**Why?**

Improves user experience with immediate feedback.

**Where?**

Used in:

* All frontend forms

**Example:**

JavaScript -->

document.getElementById("formId").addEventListener("submit", function(e) {if

(!this.checkValidity()) {e.preventDefault();alert("Please fill out the form correctly.");}});

**3. User Roles & Access Control :**

**User Table Extension**

**What?**

Added a role column to the users table.

**Why?**

Enables **Role-Based Access Control (RBAC)**.

**SQL Example:**

ALTER TABLE users ADD COLUMN role ENUM('admin', 'editor', 'user') NOT NULL DEFAULT 'user';

**Role-Based Access Control**

**What?**

Custom helper function requireRole() checks the logged-in user's role.

**Why?**

Restricts sensitive operations like create, edit, and delete to authorized users only.

**Where?**

Used at the top of:

* create.php
* edit.php
* delete.php

**💡 Helper Function (utils/session.php):**

function requireRole($roles) { if (!isset($\_SESSION['user\_role']) || !in\_array($\_SESSION['user\_role'], (array)$roles)) { header("HTTP/1.1 403 Forbidden");

exit("Access denied");

}

}

**4. Summary of Security Measures :**

| **Measure** | **Description** |
| --- | --- |
| ✔ Prepared Statements → | SQL queries now use PDO to prevent SQL injection |
| ✔ Server-Side Validation → | All forms validate data before processing |
| ✔ Client-Side Validation → | HTML5 + JavaScript for improved UX and error prevention |
| ✔ User Roles → | Added role column in users table |
| ✔ Access Control → | Protected routes and actions based on user roles |

Prepared by:

CHINTAKULA SAI SURYA