Prueba técnica: Controlador de almacenamiento seguro (PHP/JS)

Objetivo del proyecto

El objetivo es desarrollar una pequeña aplicación web para la gestión de archivos que permita a los usuarios subir documentos, al tiempo que aplica un conjunto de reglas de negocio para garantizar la seguridad y el uso justo del almacenamiento.

La aplicación debe contar con una interfaz de usuario sencilla y un panel de administración para gestionar las reglas.

Requerimientos no funcionales

- **Backend:** Cualquier framework de PHP o PHP puro (POO).
- **Frontend: Vanilla JavaScript** (ES6+). No se deben usar frameworks como React, Vue o Angular. Se permite el uso de librerías ligeras para estética (ej. Bootstrap, TailwindCSS) o peticiones (ej. Axios, aunque se prefiere `fetch`) solo si es necesario.
- **Base de datos:** MySQL o PostgreSQL.
- **Servidor:** Se puede usar el servidor de desarrollo incorporado de PHP o un entorno local como XAMPP/Docker.

Requerimientos funcionales

La aplicación debe implementar las siguientes funcionalidades:

1. Sistema de roles y grupos

- **Roles:** La aplicación debe tener al menos dos roles:
  - **Usuario:** Puede iniciar sesión y subir archivos.
  - **Administrador:** Puede gestionar usuarios, grupos y las configuraciones del sitio.
- **Grupos:**
  - Un administrador puede crear "Grupos" (ej. "Marketing", "Desarrolladores").
  - Un administrador puede asignar usuarios a un grupo.

2. Interfaz de usuario (UI)

- **Panel de usuario:** Una vista simple donde el usuario puede ver sus archivos subidos (nombre, tamaño) y un formulario para subir nuevos archivos.
- **Panel de administrador:** Un área protegida donde el administrador puede:
  - Gestionar usuarios y asignarlos a grupos.
  - Configurar los límites y restricciones descritos a continuación.
- **Estética:** La interfaz debe ser limpia, intuitiva y estéticamente agradable.

3. Lógica de subida de archivos

Esta es la parte más importante. Todas las validaciones deben ocurrir en el backend (PHP), y el frontend (JavaScript) debe gestionar la subida y mostrar las respuestas (éxito o error) sin recargar la página.

3.1. Límite de cuota total

- **Funcionalidad:** Bloquear la subida si el almacenamiento total usado por un usuario supera su cuota.
- **Configuración (admin):**
    1. Debe existir un límite global por defecto (ej. 10 MB).
    2. El admin debe poder establecer un límite específico por grupo.
    3. El admin debe poder establecer un límite específico por usuario (este tiene la mayor prioridad).
- **Lógica:** Antes de guardar un archivo, el sistema debe calcular el uso actual del usuario y verificar si `(uso_actual + tamaño_nuevo_archivo) > cuota_asignada`.
- **Notificación (JS):** Si se bloquea, el frontend debe mostrar un error claro (ej. "Error: Cuota de almacenamiento (10 MB) excedida").

3.2. Restricción de tipos de archivo

- **Funcionalidad:** Impedir la carga de archivos con extensiones peligrosas.
- **Configuración (admin):** El admin debe poder gestionar una lista global de extensiones prohibidas (ej. `exe`, `bat`, `js`, `php`, `sh`).
- **Notificación (JS):** Mostrar un error (ej. "Error: El tipo de archivo '.exe' no está permitido").

3.3. Análisis de archivos comprimidos (.zip)

- **Funcionalidad:** Inspeccionar el contenido de los archivos `.zip` antes de aprobar la subida.
- **Lógica (PHP):**
    1. Si el archivo subido es un `.zip`, el sistema debe abrirlo temporalmente (usando `ZipArchive` o una librería similar).
    2. Debe iterar sobre todos los archivos *dentro* del `.zip`.
    3. Si cualquier archivo *interno* tiene una extensión que está en la lista de prohibidas (ver punto 3.2), la subida del archivo `.zip` completo debe ser rechazada.
- **Notificación (JS):** Mostrar un error específico (ej. "Error: El archivo 'script_malicioso.js' dentro del .zip no está permitido").

🚚 Entregables

1. Un enlace a un repositorio Git (GitHub, GitLab) con el código fuente completo.
2. Un archivo `README.md` claro que explique:
    - Las decisiones de diseño tomadas.
    - Instrucciones detalladas de instalación y configuración (clonación, `composer install`, configuración de `.env`, migración de base de datos, etc.).
    - Credenciales de ejemplo para un `Administrador` y un `Usuario`.
3. Realizarás un video explicando cómo desarrollaste la prueba (máximo 5 minutos)

⚖️ Criterios de evaluación

- **Funcionalidad:** Todos los requisitos funcionales están implementados y funcionan correctamente.
- **Calidad del código (PHP):** Código limpio, legible (no funciones complejas escritas por IA), bien estructurado (POO) y que siga los principios SOLID. Uso correcto del framework (si se aplica).
- **Calidad del código (JS):** Uso de Vanilla JS moderno (ES6+), manipulación del DOM eficiente, y manejo de peticiones asíncronas (`fetch`) y errores.
- **Base de datos:** Diseño de esquema lógico y eficiente.
- **Experiencia de usuario (UX):** La aplicación es intuitiva y las notificaciones son claras y útiles.