GESTOR DE ARCHIVOS

PUBLICO

**Por Nemusco Project**

**INTRODUCCION**

Sencillo gestor de archivos para el almacenamiento público en la web. Realizado en php-7.3.0.

Enfocados en dificultar los ataques cibernéticos para así proporcionar una interfaz segura, el procesamiento de datos proporcionados del usuario fue abordado, queriendo así evitar en lo mayor posible que los ataques tengan éxito. Se prohíbe la subida de gran cantidad de archivos ejecutables tanto de rutinas dirigidas al servidor como de rutinas dirigidas al funcionamiento del sitio. Este sitio no cuenta con ningún formulario que requiera que el usuario proporcione información adicional más que su correo electrónico, no es necesaria más información para el buen funcionamiento de este sitio. El correo electrónico únicamente se usará para el aviso de restablecimiento de contraseña. Cada usuario cuenta con su propio directorio, este es creado con su registro, una vez la cuenta es eliminada, el directorio y todo su contenido es borrado del servidor, así como también toda la información vinculada al usuario. Los usuarios están divididos en 3 niveles donde el **Root**(únicamente habrá uno) está en el primer nivel, es quién podrá desempeñar varias operaciones incluyendo la de eliminar usuarios, seguido por el **Admin** quién puede hacer todas las funciones del **Root**, excepto eliminar usuarios y por último el **Client**, todo usuario que se registre a sí mismo será, por defecto, un **Client**, este sólo puede desempeñar las operaciones de autogestión y almacenar archivos.

Cuenta con una interfaz de usuario con operaciones para el almacenamiento de archivos y cómoda administración de la cuenta de usuario. Estás operaciones son

* **Operaciones para el manejo de archivos**

1. **Subir un archivo al servidor**
2. **Crear y eliminar categorías a las cuáles los archivos estarán vinculados.**
3. **Descargar un archivo**
4. **Ciertos tipos de archivos(Multimedia y Pdf) tendrán la opción de ser vistos previamente**
5. **Eliminar un archivo**

* **Operaciones de usuario**

1. **Nivel Root**

Añadir Usuarios

Cambiar la contraseña de un usuario

Eliminar algún usuario

Ver todos los ingresos al sistema

Eliminar todos los ingresos al sistema

Autogestión de cuenta

1. **Nivel Admin**

Añadir Usuarios

Cambiar la contraseña de un usuario

Ver los ingresos al sistema

Eliminar todos los ingresos al sistema

Autogestión de cuenta

1. **Nivel Client**

Autogestión de cuenta

Ver sus ingresos al sistema

Eliminar sus ingresos al sistema

Los siguientes frameworks son usados para el desarrollo del sistema: **Bootstrap-4.5, Jquery-3.5, Sweetalert, Popperjs, DataTable, Font-awesome-4.5.**

En la carpeta **“/Sources”** se encuentran todos los archivos del funcionamiento del sistema, clasificados en subcarpetas. Mientras que en la carpeta “**/Views**” se encuentran las vistas de la interfaz de usuario, la subcarpeta “**/tables**” tiene la vista de todas las tablas de la interfaz. Cada carpeta cuenta con un archivo .htaccess para prohibir la vista de sus archivos.

**HABLEMOS DE LA BASE DE DATOS**

La Base de Datos está hecha en Mysql, el esquema de esta se puede encontrar en la carpeta “**Sources/sql/schema.sql**”.

La estructura de la base de datos está formado por 4 tablas de registros: **Users, Files, AccessBooking, Categories…**

Para conocer mejor la estructura de la base de datos, vea el diagrama en formato PDF ubicado en la misma carpeta que schema.sql.

En la tabla Users se almacena la información de la cuenta de usuario. La tabla AccessBooking almacena los ingresos a una cuenta, almacenando el nombre del usuario de la cuenta a la que se ingresó y la fecha y hora del ingreso. La tabla Categories registra las categorías de archivos creada por cada usuario, la intención de esta tabla es la agrupar archivos para que en caso de desear eliminar algún grupo de archivos estos se eliminan por su categoría. La tabla Files registrará todos los archivos subidos al sistema, estos registros tienen la información para armar la ruta del archivo.

Casi todas las tablas están relacionadas con otra, la tabla **Users** está relacionada con las tablas **Categories** y **Files**, facilitando así borrar todo registro relacionado a algún usuario cuando este sea borrado. Por otro lado, la tabla **Categories** está relacionada con la tabla **Files**, con la finalidad de eliminar categorías y sus archivos relacionados. La tabla **AccessBooking** no tiene relación con otra tabla porque es necesario mantener la información de ingreso intacta hasta que el Root lo vea necesario.

**AHORA HABLEMOS DE LAS VISTAS**

El sistema de archivos cuenta con una interfaz sencilla, que cuenta con archivos **header, footer**, que muestran la barra de navegación y adjunta los archivos javascript necesarios para el funcionamiento del sitio, respectivamente. En la subcarpeta “**/tables**” están las vistas de todas las tablas (**registro de ingresos, usuarios, archivos**) y las vistas para la gestión de datos de usuario.

**AHORA LE TOCA A LA CARPETA SOURCES**

Esta carpeta contiene todos los archivos de funcionamiento del sistema. Se divide en carpetas:

* **CSS:** Los recursos que definen el diseño del sitio, desde las carpetas de **bootstrap-4.5**, las recursos para **DataTables** y un archivo **.css** extra para algún cambio extra que se quiera hacer al diseño del sitio, por otro lado, todos los iconos son añadidos con **font-awesome-4.5.**
* **IMG:** Almacena todas las imágenes que se vayan a usar en el sitio.
* **JS:** Contiene los recursos de los frameworks requeridos en el sistema: **DataTables, Jquery, Sweetalert**, además de una subcarpeta “**/process**” que contiene los archivos de las funciones **javascript** usadas en cada vista.
* **PHP:** Contiene dos subcarpetas **“/controllers**” y **“/process”.** En la primera subcarpeta están las clases requeridas en el sistema, las transacciones de cada tabla de la base de datos se encuentran en una de estas clases, clases que se extienden de la clase **“ModelConnection”,** además, la clase **“FileManager”,** con métodos para el manejo de archivos, cabe destacar que no todos los métodos de esta clase son usados en este sistema. Por otro lado, la subcarpeta **“/process”** contiene todos los archivos para el procesamiento de las operaciones del sistema gestor de archivos. Los datos de las peticiones son enviados a estos archivos para ser procesados por medio de instancias de **AJAX** creadas en las funciones de los archivos de la subcarpeta **“/process”** de la subcarpeta **“/JS”.** En los archivos .php son procesados y una vez devuelta una respuesta, la procesa la función .js que envió los datos.
* **SQL:** Contiene el archivo .sql que construye la base de datos y un archivo en formato .pdf que describe la estructura de esta.

**SIGAMOS CON LA CARPETA FILES**

En esta carpeta se subirán los archivos procesados, cada usuario cuenta con una subcarpeta cuyo nombre es generado aleatoriamente con las funciones de php **bin2hex()** y **random\_bytes()** para evitar nombres de directorio obvios.

**EXPLICANDO EL FUNCIONAMIENTO DE LAS OPERACIONES DE MANEJO DE ARCHIVOS.**

* **Subir un archivo:** Subir un archivo crea dos nuevos elementos a la vez: Un nuevo registro en la tabla Files y un nuevo archivo que se puede encontrar en el directorio del usuario. El nuevo registro en la tabla Files es insertado si el archivo es subido de manera exitosa al servidor, para que eso ocurra el archivo es previamente procesado por el método testing() ( que verifica que no sea de un tamaño mayor al deseado ni sea de algunos de los tipos de archivos especificados en el método disallowed\_format() de la misma clase) de la clase FileManager, si es válido, se procede a subir al servidor con el método upload(), también de la clase FileManager, upload() verifica que el directorio al que quiero subir el archivo exista sino existe lo crea y luego vuelve a subir el archivo. Una vez subido el archivo al servidor es el turno de la clase ModelFiles que inserta el registro de este nuevo archivo con el método save(), si lo logra, devuelve una respuesta de verdadero, caso contrario devuelve falso.
* **Crear y eliminar categorías a las cuales los archivos estarán vinculados:** Las categorías sirven para agrupar archivos, de este modo si se desea eliminar una categoría se puede eliminar todos los archivos agrupados a esta categoría, ahora bien, para agregar una nueva categoría, método add() de la clase ModelCategories es usado, este inserta un registro en la tabla Categories. Al momento de eliminar una categoría se usa el método delete() de la misma clase, al estar relacionadas las tablas Files y Categories, si se elimina un registro de la tabla Categories, todos los registros relacionados a ese registro en la tabla Files serán borrados, el detalle es que los archivos guardados en el servidor no se habrán borrados, es por ello, que se usa el método find\_by\_categories() de la clase ModelFiles para enlistar todos los archivos asociados a la categoría, esto es previo a eliminar la categoría. Una vez obtenidos todos los archivos relacionados, un bucle elimina todos los archivos con el método delete() de la clase FileManager. Luego se procede a eliminar la categoría.
* **Descargar un archivo:** Se descarga pasando la ruta del archivo a un elemento <a></a>.
* **Vista previa de un documento:** Se busca obtener la ruta del archivo con el método find() de la clase ModelFiles para luego mostrar un elemento html que lo procese (imágenes, vídeos, audios, documentos en formato .pdf), este elemento es agregado a un modal para ser visualizado. Por defecto, solo se pueden visualizar archivos .jpg, .png, .mp3, .mp4 y .pdf, pero se pueden permitir visualizar más agregando la extensión del tipo de archivo deseado al arreglo $visibles en la vista Files.
* **Eliminar un archivo:** Para eliminar un archivo se llama al método find() de la clase ModelFiles que devuelve la ruta de archivo, esta ruta se pasa al método delete() de la clase FileManager para eliminar al archivo del servidor, por último se llama al método delete\_by\_id() de la clase ModelFiles.

**EXPLICANDO EL FUNCIONAMIENTO DE LAS OPERACIONES DE USUARIO.**

* **Añadir Usuarios:** Se hace llamando al método new\_user() de la clase ModelUsers, este método inserta un nuevo registro en la tabla Users si no se encontraron registros con esos mismos datos, para esto se usa el método find() de la clase ya mencionada.
* **Cambiar la contraseña de un usuario:** Se crea una contraseña aleatoria con las funciones php random\_bytes() y bin2hex(), una vez creada, se inserta un arreglo que contiene la nueva contraseña como parámetro para el método change\_info() de la clase ModelUsers, si se realizan los cambios de manera exitosa, se envía un correo electrónico al usuario al que se le aplicaron los cambios con la nueva contraseña.
* **Eliminar un usuario:** Se inicia obteniendo con el método find() de la clase ModelUsers el directorio del usuario que se va a eliminar, luego se llama al método delete() de la clase FileManager para eliminar de manera recursiva todos los archivos que se encuentren en el directorio del usuario y el directorio mismo. Por último se usa el método delete() de la clase ModelUsers para eliminar el registro de ese usuario.
* **Eliminar todos los ingresos al sistema:** El método delete() de la clase ModelBooking elimina todos los registros de la tabla AccessBooking si no se especifica el parámetro username;
* **Autogestión de cuenta:** Se actualizan los datos del usuario llamando al método change\_info() de la clase ModelUsers.