https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 8 - 23

Creación de un sistema web de contenido nativo

Development of a native content web system

Información del reporte:

Para citar este reporte técnico:

Chamú Arias, J. O. y Ramírez Muñoz, D. A. (2026). Creación de un sistema web de contenido nativo. *Cuadernos Técnicos Universitarios de la DGTIC, 4*(3), páginas (8 - 23). https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182

José Othoniel Chamú Arias
Dirección General de Cómputo y de Tecnologías de Información y Comunicación
Universidad Nacional Autónoma de México
chamu_as@unam.mx
ORCID: 0009-0001-4949-3024

Diego Alfonso Ramírez Muñoz
Dirección General de Cómputo y de Tecnologías de Información y Comunicación
Universidad Nacional Autónoma de México
diegoramirez.code@gmail.com
ORCID: 0009-0009-1894-9074

Resumen

El desarrollo de un sistema a la medida que trabaja en dos servidores necesita de un alto grado de personalización para actualizar distintos servicios. Derivado de ello, surge el reto de actualizar ambos servidores: el principal, el cual sólo es accesible en la red interna y que, para funcionar, requiere únicamente que los archivos multimedia y tablas específicas sean actualizados; y el secundario, que aloja el sistema web y presenta la interfaz al público usuario. Para implementarlo, se consideraron dos opciones: el gestor de contenidos WordPress o código PHP puro. Para efectos de este proyecto, se optó por utilizar la segunda opción. En consecuencia de esta decisión, se absorbieron muchas actividades (como la seguridad, la cual estaba a cargo del Gestor de Contenido) y quedaron bajo responsabilidad del desarrollador. Para cumplir con esta tarea, se decidió usar Apache Firewall, que ofrece protección contra ataques comunes.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 9 - 23

Entre los retos a resolver, se encontraba la necesidad de sincronizar la comunicación entre ambos servidores; para ello, se utilizó la librería libssh2, que cuenta con funciones de ejecución de comandos remotos, transferencia de archivos seguros, autenticación y reenvío de puertos para comunicaciones seguras. Con esta herramienta, que permite ejecutar comandos remotos, se sorteó la labor de importar y exportar la base de datos completa de un servidor a otro y se logró operar consultas directas al servidor secundario para actualizar, insertar o borrar registros específicos; también se consiguió transferir archivos multimedia sin necesidad de saturar los servidores con procesos repetidos e innecesarios. Como resultado, esta técnica de comunicación permitió una conexión ágil entre los servidores, además de que coadyuvó en la protección de la información sensible del servidor principal, debido a que no comprometió en ningún momento el acceso, siendo una alternativa confiable para proyectos que requieren una sincronización selectiva y segura.

Palabras clave: Base de datos MySQL, firewall web Apache ModSecurity, libssh2, protocolo SSH, sincronización selectiva.

Abstract

The development of a custom system running on two servers requires a high degree of customization to update its various services. Consequently, a challenge arises when updating both servers: the main one, which can only be accessed via a local network and requires only updates to multimedia files and specific tables for its operation; and the secondary, which hosts the system that functions as the website and presents the user interface to the public. To implement this, two options were considered: the WordPress content management system or native PHP code. This project adopted the latter approach. In consequence of this decision, numerous activities (such as security, which fell under the responsibility of the Content Manager) were absorbed and placed under the developer's purview. To ensure compliance with security standards, it was decided to use Apache Firewall, which offers protection against common attacks.

Among the challenges to be resolved was the need to synchronize communication between both servers. To achieve this, the libssh2 library was utilized, which features functions for remote command execution, secure file transfer, authentication, and port forwarding for safe communications. This tool, which allows the execution of remote commands, bypassed the need to import and export the entire database from one server to another. It successfully enabled direct queries to the secondary server to update, insert, or delete specific records, as well as transfer multimedia files without overloading the servers with repeated and unnecessary processes. As a result, this communication technique allowed a fast connection between the servers, and also helped to protect the sensitive information on the main server, since it did not compromise access at any time. This proved to be a safe alternative for projects that require selective and secure synchronization.

Keywords: *MySQL database, Apache ModSecurity web firewall, libssh2, SSH protocol, selective synchronization.*

1. INTRODUCCIÓN

En el ámbito académico universitario, la disponibilidad y el flujo constante de datos son pilares fundamentales para la operación de servicios educativos y administrativos. Los sistemas de almacenamiento y transferencia de información se utilizan diariamente para gestionar desde contenidos didácticos y repositorios de investigación hasta trámites escolares y registros de personal.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 10 - 23

El almacenamiento y transferencia de información en los procesos académicos son elementos fundamentales que están presentes en todas las áreas de la universidad. Este proceso debe llevarse a cabo priorizando la seguridad de los datos durante la transferencia, almacenamiento y presentación en la página, dependiendo del tipo de información que se maneje (como un número de contacto de atención al público en general o los nombres, correos, fechas de nacimiento, CURP, direcciones de personal y alumnos, por mencionar algunos ejemplos).

El tratamiento de dicha información, siguiendo las buenas prácticas de seguridad, se separa en componentes públicos y privados, con la finalidad de establecer políticas diferenciadas de acceso y protección, además de facilitar las labores de mantenimiento. Para este caso, el sistema se dividió en dos grandes apartados: una parte que almacena la información sensible y otra donde se almacena información de consulta general. El sistema, dependiendo del usuario, se encarga de dividir el acceso a la información, lo cual garantiza la separación y protección de los datos sensibles y aseguran que los datos privados permanezcan aislados de cualquier perfil de acceso público.

En el área de trabajo se tenían que considerar las siguientes restricciones: el uso de un portal de gestión de contenidos no permitía crear roles para acceder a la información de manera jerárquica y su modificación era compleja. Por lo anterior, se evaluó usar herramientas que permitieran personalización total y que disminuyeran los archivos a vigilar y mantener.

El problema base que dio origen a este proyecto (implementado en 2023 – 2024) fue la necesidad de alimentar un sitio web alojado en un servidor secundario (público) sin comprometer la seguridad del sistema principal ni recurrir a procesos manuales lentos. Tras analizar las opciones, se identificó que el uso de gestores de contenido (CMS) presentaba restricciones importantes, ya que no permitían una integración directa con el sistema de gestión interna ni permitía la creación de roles jerárquicos personalizados para el acceso a la información. Debido a ello, el trabajo se dividió en dos partes: la gestión de seguridad del sistema y el desarrollo de la aplicación sin recurrir al uso de gestores de contenido.

El objetivo del proyecto fue desarrollar un módulo para el sistema de gestión interna que permitiera la transferencia segura y selectiva de archivos multimedia y tablas de bases de datos mediante el uso de la librería libssh2. A diferencia de una réplica total, este desarrollo buscó implementar una sincronización específica a través del protocolo SSH, permitiendo actualizar únicamente los componentes necesarios en el servidor secundario que es de acceso público. Con esto, se garantiza una comunicación ágil entre servidores, manteniendo el control total sobre la seguridad y optimizando los recursos de procesamiento.

2. DESARROLLO TÉCNICO

Se desarrolló un módulo que gestiona la carga de una imagen desde un formulario HTML, la guarda en el servidor primario, actualiza la base de datos con el nombre del archivo en la columna correspondiente, realiza una copia del mismo en el servidor secundario y ejecuta una consulta remota que actualiza la base de datos en el servidor secundario, que tiene únicamente datos de lectura y la información necesaria para la consulta. Para gestionar de manera eficiente este método, fue primordial configurar correctamente el entorno de desarrollo. Con este fin, se instalaron las extensiones mysqli (MySQL Improved), que permiten a los *scripts* gestionar la base de datos, y libssh2, para la ejecución remota de comandos y transferencia de archivos.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 11 - 23

Este proceso cubre el ciclo completo de carga de archivos necesarios para visualizar información en el sitio web, desde el guardado del archivo y la actualización de la base de datos en el servidor principal hasta la actualización remota del servidor secundario mediante funciones como Secure Copy (SCP) y consultas remotas.

2.1 METODOLOGÍA

La metodología que mejor se adaptó para el desarrollo de este proceso fue el modelo en cascada. Este enfoque, de acuerdo con González *et al*. (2021), "consiste en la ejecución de etapas unas tras otras, de arriba hacia abajo; esto requiere que antes de pasar a la siguiente fase se necesita que la primera fase haya concluido para proseguir con la siguiente etapa" (p. 188). Por ello, este acercamiento metodológico fue el apropiado para sistemas basados en transferencias y resultó fundamental para delimitar los momentos exactos en los que el método cambiaba de servidor. Debido a que ésta no permite avanzar a la siguiente fase si no se ha verificado la anterior, se garantizó una implementación segura y estructurada, a través de las siguientes etapas:

Diseño del proyecto

Se evaluaron las herramientas de terceros (*plugins*) que realizaran esta tarea de forma segura, y, debido a que no se encontró una opción que se integrara a las necesidades específicas de la institución, se decidió programar un módulo desde cero.

Diseño de flujo de datos

Se determinó de manera estricta qué archivos y datos debían permanecer aislados en el servidor primario y cuáles debían transferirse al servidor secundario para estructurar el recorrido de la información.

Selección de tecnologías

Con el flujo establecido, se definió el entorno de desarrollo y se seleccionó el lenguaje de programación, el motor de base de datos y la configuración del sistema. Además, se integraron las extensiones mysqli para la gestión de datos y libssh2 para la ejecución remota de comandos y transferencia de archivos (SCP).

Programación

Se desarrolló la estructura arquitectónica necesaria para integrar este nuevo módulo dentro del sistema de gestión interna ya existente. Se programó, en primera instancia, la lógica del servidor principal (carga de imágenes, almacenamiento local y actualización de la base de datos interna). Posteriormente, se estructuraron las versiones de la base de datos en el servidor secundario, creando únicamente las columnas estrictamente necesarias para el entorno público. Finalmente, se desarrolló el *front-end*, el cual, de acuerdo con Pressman y Maxim (2021), abarca la arquitectura de la interfaz de usuario y la lógica de presentación con la que interactúa de manera directa el público; en este proyecto, dicho entorno quedó encargado de consumir y mostrar esta información ya filtrada de forma segura.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 12 - 23

Instalación

Se configuró el entorno de producción en ambos servidores y se certificó que las librerías necesarias estuvieran habilitadas, así como que los permisos de red permitieran la comunicación por el puerto correspondiente para la ejecución de comandos de forma remota.

Pruebas

Para la validación técnica de cada etapa, se implementó un riguroso manejo de excepciones y verificación de resultados en el código con pruebas de funcionalidad. Según Rangel Cano (2021), las pruebas funcionales son la "actividad dedicada a la aplicación de las pruebas que analizarán el funcionamiento del *software* bajo un enfoque de usuario final... identificando interrupciones derivados de defectos funcionales" (p. 5). Además, estas pruebas se alinean con el modelo en cascada, garantizando que cada fase se complete antes de avanzar a la siguiente.

Mantenimiento

Se establecieron lineamientos para la revisión de los registros (*logs*) de transferencia y errores de las funciones SSH, con el fin de garantizar que la sincronización selectiva se mantenga operativa a lo largo del tiempo ante posibles actualizaciones del servidor.

Seguridad

Se aplicó una estricta segmentación de la información. El uso del protocolo de seguridad SSH garantizó que tanto la copia de archivos como las consultas remotas viajaran de forma encriptada, cumpliendo el objetivo principal de mantener los datos privados completamente aislados del servidor secundario de acceso público. Cabe mencionar que se aplicó un proceso de *hardening* al servicio SSH del servidor mediante el cambio del puerto estándar, tal como lo recomienda Toapanta Rocha (2023) en su metodología de endurecimiento de sistemas operativos para mitigar accesos no autorizados.

2.1.1 ANÁLISIS

Debido a la estructura de un sistema integral de gestión interna, se identificó la necesidad de utilizar la información almacenada en él para actualizar automáticamente diferentes apartados de un sitio web alojado en un servidor independiente. Este proceso eliminó la intervención manual, la cual implicaba una inversión de tiempo innecesaria y el entrenamiento en conocimientos técnicos especializados para los usuarios administradores de esos módulos.

Después de analizar la problemática, se desechó el uso de gestores de contenidos y se concluyó que no resultaban adecuados, ya que no permitía la integración directa con el sistema de gestión interna ni la automatización del intercambio de datos entre servidores, limitándose a un entorno de administración propio y restringido.

Por ello, se decidió implementar la librería libssh2, gracias a que provee, a través de sus funciones, acceso a recursos sobre una máquina remota utilizando una vía de transporte criptográfica segura. Además, esta alternativa ofrece ventajas para optimizar el rendimiento —particularmente relevante dado que ambos servidores cuentan con recursos limitados y se priorizó minimizar el consumo de ancho de banda y procesamiento—, además de otros beneficios orientados a adaptar la seguridad a las necesidades

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 13 - 23

exactas del desarrollo, limitando las operaciones únicamente a los archivos y tablas involucradas en el proceso. Todo lo mencionado se refleja en la Tabla 1.

Tabla 1

Comparación entre libssh2 y migrar el sitio a un gestor de contenidos

Criterio	libssh2	Gestor de Contenidos (CMS)
Control sobre el flujo de trabajo	Control total sobre cada paso del proceso (carga, validación, almacenamiento, etc.).	Control limitado por la configuración predeterminada del CMS y las opciones disponibles en *plugins*.
Flexibilidad	Altamente flexible, permite cualquier personalización según las necesidades del proyecto.	Flexibilidad limitada a la funcionalidad básica y los *plugins* disponibles; personalizaciones avanzadas son difíciles.
Rendimiento	Rendimiento optimizado, ya que sólo se cargan las funcionalidades necesarias.	Generalmente más lento, debido a la sobrecarga causada por módulos y *plugins* innecesarios.
Seguridad	Se pueden implementar medidas de seguridad específicas, como la validación personalizada y control estricto.	Seguridad basada en *plugins* o módulos, que pueden ser vulnerables si no se actualizan adecuadamente.
Transferencia de archivos	Integración directa con SSH o SCP para transferir archivos a servidores remotos de forma segura.	Requiere complementos o configuraciones adicionales para manejar la transferencia entre servidores.
Gestión de bases de datos	Consultas SQL completamente personalizadas, optimizadas para el proyecto específico.	Uso de consultas estándar y limitadas a las funcionalidades del CMS; requiere *plugins* para personalización.
Escalabilidad	Escalable a medida que crece el proyecto, con posibilidad de optimización a nivel de código y base de datos.	Escalabilidad limitada, dependiente de la estructura interna del CMS y de los *plugins* instalados.
Facilidad de implementación	Más complicado de implementar, requiere conocimientos técnicos avanzados de PHP, bases de datos y servidores.	Fácil de implementar para proyectos pequeños o rápidos, con una curva de aprendizaje más baja.
Tiempo de desarrollo	Mayor tiempo de desarrollo debido a la personalización y pruebas necesarias.	Menor tiempo de desarrollo inicial debido a las funciones prediseñadas, pero la personalización puede ser compleja.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 14 - 23

Actualizaciones y mantenimiento	Mantenimiento gestionado por el equipo de desarrollo; control total sobre actualizaciones y cambios.	Actualizaciones dependientes del CMS y los *plugins*, con posibles incompatibilidades entre versiones.

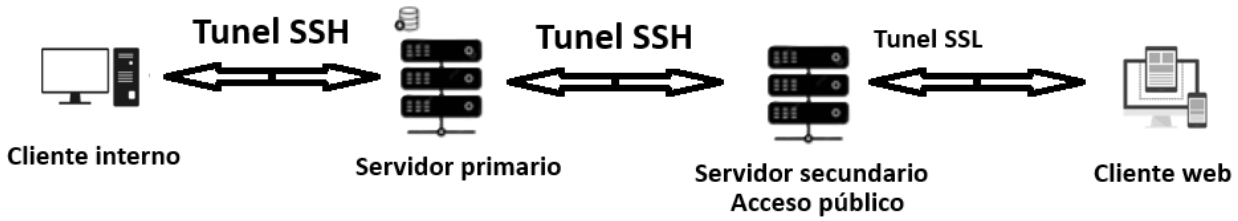
Como se observa en la tabla anterior, se tiene una comparativa de ventajas y desventajas de usar un método u otro, lo cual nos sirve de apoyo visual para seleccionar un estilo de desarrollo.

2.1.2 DISEÑO

En esta fase, se identificó la ruta crítica del proceso, se definió la forma en que los archivos serían transferidos dentro del sistema y se determinó la manera en que serían enviados al servidor secundario. Para ello, el proceso se dividió en dos grandes bloques: el servidor primario, al cual sólo es posible ingresar desde redes locales y se encarga de guardar la información y prohibir su acceso al público; y el servidor secundario de acceso público, donde se coloca la página web. El esquema se muestra en la Figura 1.

Figura 1

Ruta de la información



En la figura se aprecia de manera gráfica cómo el archivo viaja desde la aplicación que se ejecuta en el servidor primario y se transfiere al servidor secundario mediante un túnel cifrado con SSH, para proteger su contenido durante el tránsito entre servidores y clientes.

2.1.3 IMPLEMENTACIÓN

En esta fase de la metodología, se desarrolló el modelo de carga y procesamiento de los archivos. Este proceso se dividió en diferentes etapas: primero, se implementó la captura del archivo cargado por el usuario, el cual se almacena en la ruta previamente definida en el servidor local, utilizando un identificador único y la extensión de cada uno de los archivos. Para ello, se empleó un módulo que permite subirlo mediante la función *move_upload_file()*, lo cual asegura que el archivo se mueva de la ubicación temporal a la ruta definida; después, si la operación resulta exitosa, se realiza la actualización de la base de datos local mediante una consulta SQL (*Structured Query Language*), que registra el nombre del archivo, la extensión y la ruta de su elemento correspondiente.

Las ocasiones en que se validó la actualización de la base de datos y se concretó poder subir el archivo en el servidor local, se desarrolló el procedimiento de transferencia del archivo al servidor secundario mediante la función *ssh_scp_send()*, que permite la transmisión de información mediante el protocolo de copia segura. De acuerdo con Effendi *et al.* (2021), este protocolo utiliza una conexión SSH en la que los datos se envían cifrados. La validación de la carga se realizó mediante la verificación del código de retorno de la función y la comprobación de la existencia y tamaño del archivo en el destino, asegurando

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 15 - 23

que la transferencia se completara exitosamente antes de continuar con el siguiente paso. De igual forma, se implementó la lógica para actualizar la base de datos en el servidor remoto. Cabe aclarar que esta base de datos fue diseñada únicamente con las tablas necesarias para la alimentación del sitio web; la actualización se efectuó mediante la ejecución de una consulta SQL similar a la realizada en el servidor local.

Para asegurar la robustez del módulo, fue necesario integrar estructuras de manejo de errores que permitieran identificar si existió algún problema durante la transferencia de archivos o durante la actualización remota de la base de datos. Posteriormente, se cerraron todas las conexiones abiertas que se comunicaran con el servidor remoto, lo que liberó recursos del servidor y aseguró que no quedaran conexiones activas que pusieran en peligro la seguridad de los servidores. Finalmente, la plataforma mostró mensajes de éxito, indicando que los datos fueron registrados correctamente y, después de un retraso de dos segundos, redirige al usuario a la página donde puede visualizar los archivos subidos.

2.1.3.1 MANEJO DEL ARCHIVO

Aquí se obtiene el nombre del archivo a subir desde el formulario y se extrae el nombre y la extensión que, en este caso, puede ser .jpg o .png. Cabe mencionar que la variable *$archivo* pasó por un proceso de limpieza.

Figura 2

Obtención de extensión del archivo

```
$extArchivo = pathinfo($archivo);
$extArchivo1 = $extArchivo['extension'];
```

La Figura 2 muestra un segmento de código donde se lleva a cabo la obtención de la extensión del archivo para poder clasificarlo y validarlo dentro del servidor. Cabe mencionar que éste es un paso indispensable para evitar que se procesen extensiones no válidas e impedir la inyección de código malicioso, una práctica fundamental en la mitigación de vulnerabilidades de seguridad web tal como recomienda Cajías (2020).

2.1.3.2 DEFINICIÓN DE LA RUTA LOCAL

Se definió la ruta local donde se almacenará el archivo. La ruta incluye el directorio, el identificador único del archivo y su extensión.

Figura 3

Ruta de servidor local

```
$rutaLocal = '/ruta/servidor/local/';
$fichero_subido = $rutaLocal.$idArchivo.'.'.$extArchivo1;
```

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 16 - 23

El código de la Figura 3 muestra cómo se lleva a cabo del proceso de creación de la ruta donde se va a depositar el archivo. Esto permite personalizar un directorio separado del sistema, de modo que, en caso de ser necesario, pueda redimensionarse su espacio sin afectar al sistema.

2.1.3.3 SUBIR EL ARCHIVO Y ACTUALIZAR LA BASE DE DATOS EN EL SERVIDOR LOCAL

En esta etapa, se realiza el traslado del archivo desde su ubicación temporal, asignada desde la carga, hasta la ubicación definida por *$fichero_subido*, mediante la instrucción mostrada en la Figura 4.

Figura 4

Subir el archivo

```
if (move_uploaded_file($_FILES['archivo']['tmp_name'], $fichero_subido))
```

Como se muestra en el código, se valida la acción de mover el archivo de los directorios temporales al directorio de destino final. Esto es importante para evitar inconsistencias de información, ya que guardar un estado de éxito en la base de datos sin asegurar que el archivo se cargue correctamente puede generar información imprecisa en la base.

Si no existe ningún error, el proceso continúa y se actualiza la base de datos. Se ejecuta una consulta de inserción que actualiza el campo “archivo” de la tabla correspondiente con el nombre y la extensión del fichero subido. La ejecución se muestra en la Figura 5.

Figura 5

Actualizar la base de datos

```
$Update = '
    update tabla
    set archivo = \''.$idArchivo.'.'.$extArchivo1.'\'
    where idArchivo = '.$idArchivo.'';
$mysqli->query($Update) or die ('Error en el query $Update1: '.$mysqli->error);
```

Como se ve en la Figura 5, ya que se validó que el archivo se cargó correctamente, se registra en la base para así tener información consistente en el sistema.

2.1.3.4 TRANSFERENCIA REMOTA

Las ocasiones en que subir el archivo y actualizar la base de datos local se completó de manera exitosa, se incluyó el archivo que contiene los parámetros de conexión para realizar la transferencia remota del archivo y la actualización de la base de datos, para lo cual es necesario primero definir las rutas de los servidores. Posteriormente, se utiliza la función *ssh2_scp_send()* para transferir el archivo del servidor local al remoto, seguido de los permisos para asegurar que esto suceda sin ningún problema.

Una vez que el archivo fue transferido de manera exitosa, se ejecutó una consulta SQL mediante una conexión SSH, que replica la misma actualización realizada en la base local.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 17 - 23

En la Figura 6 se muestra el código con rutas relativas, las cuales son útiles para poder migrar el código de manera más transparente, debido a que no está fijo en una estructura de directorios.

Figura 6

Conexión remota

```
include('../ruta/conexion.php');
$origen = '/ruta/servidor/local/'.$idArchivo.'.'.$extArchivo1;
$destino = "/ruta/servidor/remoto/".$idArchivo.'.'.$extArchivo1;
```

En esta imagen se muestra cómo se construyen las rutas origen y destino para poder transferir la información de manera dinámica dentro del sistema, con lo cual se evita que el proceso se lleve a cabo de forma manual.

La función que se muestra en la Figura 7 es útil para ejecutar el comando de sistema operativo de copiado seguro mediante un canal cifrado.

Figura 7

Transferencia del archivo al servidor remoto

```
ssh2_scp_send($connection, $origen, $destino, 0777);
```

Una vez mencionada la función de la instrucción, el segmento de código de la Figura 7 ejecuta la instrucción para conectar ambos servidores y transferir la información entre los mismos. Cabe mencionar que el último valor corresponde a los permisos otorgados en el servidor remoto e indica que todos los usuarios tienen permiso de lectura, escritura y ejecución dentro del servidor aislado (remoto).

La función de la Figura 8, en este caso *ssh2_exec*, tiene la utilidad de ejecutar comandos en servidores remotos mediante un canal cifrado.

Figura 8

Actualización de la base de datos remota

```
$consulta = '
    update tablaremota
    set archivo = \''.$idArchivo.'.'.$extArchivo1.'\'
    where idArchivo = '.$idArchivo.';';
$stream = ssh2_exec($connection, 'mysql -u user -p "baseremota" -sse "'.$consulta.'";'
```

Como se ve en el segmento de código mostrado arriba, se indica el proceso de cómo se ejecuta un comando en la base de datos de manera remota. De esta manera, se restringe el acceso directo a la base de datos desde exterior, de modo que las consultas únicamente pueden realizarse mediante la función *ssh2_exec* dentro del código PHP. Con esto, se evitan los ataques de fuerza bruta, los cuales se definen

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 18 - 23

como métodos de intrusión basados en el mecanismo de prueba y error. Tras cerrar el acceso en varias capas, se implementa el modelo de seguridad en profundidad. De acuerdo con Rodríguez Gahona (2020), "en esa estrategia de defensa, en lugar de colocar una única línea muy fuerte, se colocan varias líneas consecutivas" (p. 37), lo cual tiene como finalidad lograr que el atacante desista y cambie de objetivo.

2.1.3.5 MANEJO DE ERRORES Y NOTIFICACIONES

Para garantizar que durante el proceso no se produjeran errores inesperados, se implementaron estructuras de manejo de errores basados en la captura y gestión de errores en tiempo real. En primera instancia, se utilizó *or die()* en la ejecución de las consultas SQL y durante el proceso de transferencia de archivos, con el fin de mostrar un mensaje en caso de que ocurriera un problema en la ejecución. Lo anterior permitió obtener el error específico de la causa de la interrupción del proceso.

Para las transferencias remotas, se utilizaron las funciones *ssh_exec()* y *ssh_fetch_stream(),* con el objetivo de capturar los errores de una conexión SSH. Para esto, se configuraron los flujos como bloqueadores utilizando la función *stream_set_blocking(),* lo cual forzó al módulo a esperar una respuesta antes de continuar. Finalmente, los flujos fueron cerrados para liberar los recursos y evitar que se produjeran otro tipo de errores inesperados y, dependiendo del éxito o fallo de la tarea, se notificara al usuario a través de mensajes de error mostrados en pantalla.

Se implementaron las funciones de manejo de flujos (o *streams*) para controlar de manera rigurosa la conexión entre servidores.

Figura 9

Manejo de errores

```
$errorStream = ssh2_fetch_stream($stream, SSH2_STREAM_STDERR);
stream_set_blocking($errorStream, true);
stream_set_blocking($stream, true);
fclose($errorStream);
fclose($stream);
```

El código de la Figura 9 sirve para capturar los errores del servidor remoto de forma controlada y evita que el *script* de PHP se quede congelado.

2.1.4 PRUEBAS

La fase de pruebas consistió en ejecutar diversos escenarios para validar el correcto funcionamiento del módulo. Esto también condujo a definir el tipo de archivo y el tamaño máximo para optimizar la transferencia de información entre servidores, estableciendo sólo la transferencia de imágenes en formato .jpg de máximo 2.5MB y .pdf de máximo 10MB.

A lo largo de cada prueba, se observó el comportamiento del sistema para mover los archivos desde su ubicación temporal hasta el servidor remoto; se probaron también las consultas en ambas bases de datos, verificando que la información sobre los datos cargados fueran los correctos, y que se hubieran registrado en las columnas correspondientes. También se realizaron pruebas para evaluar el proceso

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 19 - 23

de transferencia de archivos vía SSH, que posibilitaron realizar ajustes a las conexiones y personalizar los permisos para reducir el riesgo en la escritura, o dejar abierta alguna conexión que pudiera resultar en el mal funcionamiento del módulo. Cada error fue monitoreado para gestionar adecuadamente las conexiones fallidas o permisos incorrectos.

2.1.5 MANTENIMIENTO

A partir de los resultados obtenidos de las pruebas y después de obtener una versión estable de este módulo, se liberó al usuario la versión que se encargaría del registro de la información, pero se mantuvo una comunicación constante para adaptar el módulo a las necesidades de éste. El intercambio de información permitió realizar ajustes personalizados a la transferencia de archivos, tales como agregar nuevos formatos (.epub y .png) solicitados por el usuario. Estas mejoras permitieron al modelo ser escalable y habilitaron la retroalimentación por parte de los usuarios para conducir los ajustes conforme se avanzara en el desarrollo. Sin embargo, se pudo observar que, en archivos mayores a 25MB, el rendimiento del sistema se veía afectado considerablemente. Por ende, fue necesario cargar los archivos manualmente en el servidor remoto; este hallazgo condujo a empezar a desarrollar nuevas técnicas en el proceso de transferencia que ofrecieran un mejor manejo de archivos de gran volumen.

2.1.6 SEGURIDAD

Una vez finalizados los módulos de carga de archivos y actualizaciones de base de datos, se identificó la necesidad de agregar una capa de seguridad para mitigar ataques, por lo que se llevaron a cabo las siguientes acciones:

Se autorizó el acceso a los servicios solamente a un grupo cerrado de direcciones IP mediante un *firewall* perimetral. Este mecanismo, el cual es descrito por Caiza Chipantaci (2025) como un sistema de seguridad esencial diseñado para segmentar y separar las redes, controla estrictamente el tráfico hacia la red interna y entre los servicios, evitando así la exposición de la infraestructura al público general.

Para mitigar el impacto de los *sniffers*, los cuales se definen como herramientas especializadas en interceptar, escuchar y capturar el tráfico de datos dentro de una red informática, tal como lo señalan Lescano y Quiros (2021), se implementaron certificados digitales en los servidores web. Asimismo, como se mencionó en el desarrollo, se utilizó la librería libssh2 para cifrar la comunicación en la actualización de las bases de datos y en la transferencia de archivos, garantizando la confidencialidad y la secrecía en el tránsito de la información.

Para mitigar algunos ataques que pudieran venir de equipos internos, tales como: SQL Injection, el cual consiste en manipular las instrucciones web para ejecutar consultas maliciosas en la base de datos; *path transversal*, el cual permite navegar entre directorios del sistema operativo manipulando los datos web que se mandan al servidor; *command injection*, que permite ejecutar comandos de sistema operativo en el servidor web manipulando datos que no son debidamente filtrados; *cross site scripting*, que permite ejecutar instrucciones de JavaScript en equipos remotos para secuestrar sesiones a través de robar identificadores de sesión; se delegó la protección a un *firewall* web de Apache, el cual está implementado en el módulo Modsecurity y cuenta con una serie de reglas para mitigar los ataques mencionados y muchos más, ya que basa muchas de sus reglas en el Open Web Application Security Project (OWASP), que es una colección de recomendaciones de seguridad para proteger aplicaciones.

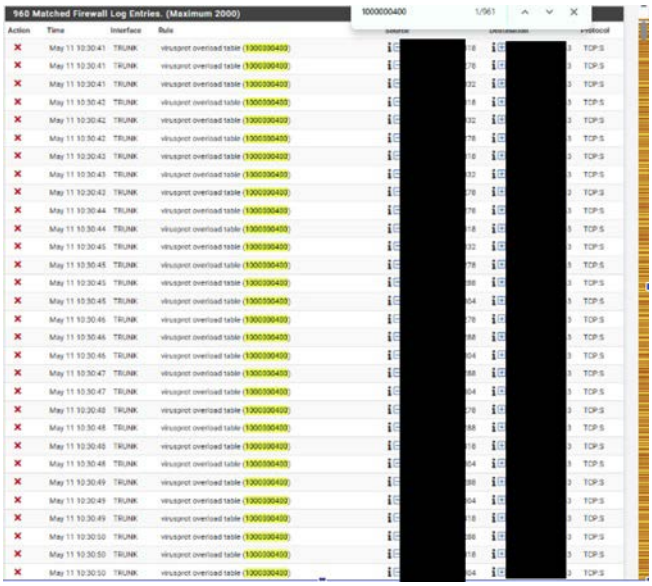
https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 20 - 23

Se utilizó el *firewall* web de Apache porque es gratuito y menos agresivo y complicado de usar comparado con Suricata (Sistema de Detección de Intrusos), de acuerdo con nuestra experiencia. En este contexto, se considera que este último tiene un gran rango de instrucciones de bloqueo de ataques, sin embargo, excede a las necesidades de la aplicación web y eleva la complejidad del despliegue de la infraestructura. Además, en ocasiones previas, se ha observado que puede generar falsos positivos, los cuales expulsan del sistema a usuarios legítimos, así como se debe instalar en una capa externa al servidor web para disminuir la carga a éste.

En la Figura 10 se muestra una salida correspondiente a la implementación del sistema en diversos proyectos. La siguiente imagen muestra el detector de intrusos en entornos de mayor complejidad. En este caso, se aprecia un intento de ataque de virus hacia un servidor, el cual es bloqueado efectivamente por el detector de intrusos al analizar el comportamiento de las conexiones.

Figura 10

Suricata



El uso de esta herramienta no debe ser descartado si se cuenta con los recursos de infraestructura necesarios para implementarla, ya que Suricata optimiza su rendimiento y trabaja mejor en entornos de redes de alta velocidad, tal como lo indican Cabrera Laguapillo y Larco Andrade (2018) en su análisis de sistemas de detección de intrusos periféricos. Para efectos de este proyecto, el *firewall* web de Apache cubrió las necesidades de protección y cumplió con el objetivo de optimizar la carga del servidor web.

3. RESULTADOS

En el desarrollo del módulo, se logró implementar de manera exitosa un mecanismo de sincronización entre el servidor local y el servidor remoto, mediante funciones diseñadas específicamente para la transferencia, registro y validación de archivos. Durante las pruebas realizadas, tanto en entornos

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 21 - 23

controlados como con usuarios finales, se comprobó que los archivos cargados son correctamente almacenados en las rutas definidas y registrados de forma consistente en ambas bases de datos. Este proceso incluye validaciones que garantizan la integridad de la información, evitando inconsistencias entre los servidores. Asimismo, se incorporaron mecanismos de control que detectan errores en la comunicación, tales como fallos en la conexión a bases de datos o interrupciones en la conexión SSH, permitiendo detener el proceso y notificar al usuario con mensajes claros, lo que fortalece la confiabilidad del sistema.

Por otra parte, se consiguió que la seguridad del sistema recayera directamente en la lógica implementada por el desarrollador, por la acción deliberada de no utilizar *frameworks* o sistemas de gestión de contenido. Esta decisión implicó el desarrollo manual de controles de validación, limpieza de datos, manejo de sesiones y verificación de accesos, lo que facilitó el entendimiento profundo de los mecanismos de seguridad involucrados. Entre las ventajas observadas, destacan las siguientes: una mayor flexibilidad en la implementación de políticas específicas de seguridad, una reducción en la dependencia de terceros y un mejor control sobre el comportamiento del sistema. Además, se minimizó la sobrecarga de componentes innecesarios, lo cual optimizó el desempeño general y facilitó la personalización del módulo conforme a los requerimientos del entorno.

Finalmente, a través de las pruebas de desempeño, se logró identificar que ciertas problemáticas relacionadas con el tiempo de carga y la estabilidad en la transferencia de archivos no corresponden directamente a deficiencias del *software*, sino a limitaciones del *hardware* y de la infraestructura de red. En particular, en el proceso de trabajar con archivos de gran tamaño, se observó un incremento considerable en los tiempos de transferencia y, en algunos casos, interrupciones en la conexión. Este comportamiento fue clave para distinguir entre cuellos de botella asociados al sistema y aquellos derivados de recursos físicos, lo cual resulta clave para futuras optimizaciones.

4. CONCLUSIONES

Los resultados obtenidos demuestran que la solución propuesta cumple de manera satisfactoria con los objetivos planteados. Se logró una sincronización efectiva entre servidores, un manejo adecuado de errores y una implementación robusta de medidas de seguridad, además de identificar correctamente las limitaciones asociadas al *hardware*. En conjunto, estos resultados resolvieron el problema inicial gracias a que proporcionaron un sistema funcional, confiable y adaptable, capaz de operar dentro de los parámetros esperados y responder adecuadamente ante escenarios de fallo.

Es importante destacar que el desarrollo prescindió del uso de gestores de contenido y *frameworks* no por una desventaja inherente a estas herramientas, sino por una decisión basada en los requerimientos específicos del usuario. Se reconoce que los gestores de contenido representan soluciones sólidas, ampliamente probadas y con múltiples ventajas en términos de rapidez de desarrollo, mantenimiento y escalabilidad. Sin embargo, para este proyecto en particular, no se requería dicha capa de abstracción, por lo cual se optó por una implementación más ligera y completamente controlada. Esta decisión favoreció la flexibilidad del sistema y la adaptación precisa a las necesidades planteadas; ello no implica una postura en contra del uso de este tipo de herramientas en otros contextos.

El principal aporte de este trabajo es la implementación de mecanismos de optimización en la transferencia de archivos de gran tamaño, como la fragmentación de archivos, el uso de colas de procesamiento o

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 **Vol. 4, Núm. 3.** julio-septiembre 2026, págs. 22 - 23

la compresión previa a la transferencia. Recomendamos para trabajos futuros la exploración de la integración de herramientas híbridas que mantengan el control del desarrollador, pero aprovechando ciertas ventajas de *frameworks* ligeros en aspectos específicos como seguridad o manejo de sesiones. Este tipo de proyectos puede evolucionar hacia sistemas más complejos de gestión distribuida de archivos o plataformas escalables que integren monitoreo en tiempo real, tolerancia a fallos y balanceo de carga, ampliando así su alcance y aplicabilidad en entornos de mayor exigencia.

AGRADECIMIENTOS

Agradecemos al Equipo editorial de CTUD por acompañar a nuestro texto en todas sus etapas y por sus valiosas recomendaciones que hicieron posible el presente trabajo. En especial valoramos el tiempo dedicado por Eric Villar Juárez y Juan Manuel Campos Coto para asesorarnos sobre la estructuración y redacción de este reporte técnico.

Declaración de contribución de autoría

Diego Alfonso Ramírez Muñoz: Especialista en desarrollo de *software* y análisis de soluciones tecnológicas. Rol dentro del proyecto: desarrollador. Participó en la identificación de las necesidades del cliente/usuario y del sistema, con el propósito de definir una solución tecnológica que respondiera de manera adecuada a los requerimientos funcionales y operativos del proyecto. Su intervención se centró en analizar el flujo de trabajo esperado, las condiciones de uso de la aplicación y los elementos técnicos necesarios para construir una solución clara, funcional y mantenible. A partir de esta revisión, consideró aspectos relacionados con la facilidad de uso, la disminución de la complejidad para el usuario final y la correcta separación entre los componentes de infraestructura y aplicación. Asimismo, colaboró en la definición de medidas orientadas a proteger el tránsito de la información a través de la red, procurando que la solución mantuviera criterios de seguridad, organización y eficiencia durante su operación.

José Othoniel Chamú Arias: Especialista en tecnologías de la información y seguridad informática. Rol dentro del proyecto: *pentester* y administrador de servidores. Participó en la identificación de herramientas, configuraciones y medidas técnicas necesarias para proteger la aplicación una vez desplegada en el servidor web. Su contribución se orientó al fortalecimiento de la seguridad de la solución mediante la revisión de aspectos del código y la detección de oportunidades de mejora para reducir la exposición ante ataques conocidos. Asimismo, colaboró en el despliegue de la aplicación dentro de los servidores correspondientes, realizando actividades de instalación y configuración de los ambientes requeridos para su operación. Estas actividades incluyeron la configuración de la base de datos, servidor web, sistema operativo, creación de cuentas, apertura y restricción de puertos, así como la configuración de módulos necesarios para el funcionamiento de la aplicación. También intervino en la incorporación de herramientas de protección, tales como el *firewall* web, con el fin de mejorar la seguridad y estabilidad del servicio.

REFERENCIAS

Cabrera Laguapillo, M. S. y Larco Andrade, Y. C. (2018). *Análisis de las técnicas de ocultación de código malicioso para evasión de sistemas de protección de usuario final y NIDS* [trabajo de titulación de ingeniería, Escuela Politécnica Nacional]. Repositorio Institucional de la EPN. https://bibdigital.epn.

https://doi.org/10.22201/dgtic.30618096e.2026.4.3.182 Vol. 4, Núm. 3. julio-septiembre 2026, págs. 23 - 23

edu.ec/handle/15000/19514

Caiza Chipantaci, E. M. (2025). *Implementación de ciberseguridad a través de pfSense con SIEM* [trabajo de integración curricular, Escuela Politécnica Nacional]. Repositorio Institucional de la EPN. https://bibdigital.epn.edu.ec/handle/15000/26460

Cajías, E. F. (2020). *Evaluación de la eficiencia de ataques de tipo clickjacking y touchjacking en entornos controlados* [trabajo de titulación de maestría, Escuela Politécnica Nacional]. Repositorio Institucional de la EPN. https://bibdigital.epn.edu.ec/handle/15000/21498

Effendi, M. R., Al-Falah, R. S., y Ismail, N. (2021, 19-20 de agosto). *IoT-Based Battery Monitoring System in Solar Power Plants with Secure Copy Protocol (SCP)* [ponencia]. 2021 7th International Conference on Wireless and Telematics (ICWT), Bandung, Indonesia. https://doi.org/10.1109/ICWT52862.2021.9678210

Lescano Andrade, B. L., y Quiros Chulca, C. A. (2021). *Implementación de dos sniffers inalámbricos en un sistema Raspberry Pi para el componente práctico de comunicaciones inalámbricas de la ESFOT* [trabajo de titulación de tecnólogo, Escuela Politécnica Nacional]. Repositorio Institucional EPN. https://bibdigital.epn.edu.ec/handle/15000/21812

González, Y., MC lenan, J., y Abrego, A. (2022). Los proyectos y las metodologías. Modelos de procesos: Ciclos de vida de desarrollo de software. *Revista Semilla Científica*, *3*(3), 185–194. https://revistas.umecit.edu.pa/index.php/sc/article/view/1090

Pressman, R. S., & Maxim, B. R. (2021). *Ingeniería de software: Un enfoque práctico* (9a ed.). McGraw-Hill Interamericana.

Rangel Cano, L. L. (2021). *Metodología para aplicar pruebas funcionales como parte de una auditoría de sistemas de información*. Dirección General de Cómputo y de Tecnologías de Información y Comunicación, UNAM. https://www.red-tic.unam.mx/recursos/2021/metodologia-pruebas-funcionales.pdf

Rodríguez Gahona, G. (2020). *Análisis comparativo de los modelos defensa en profundidad y MSPI, para la implementación de la seguridad informática en el sector privado del país* [trabajo monográfico de especialización, Universidad Nacional Abierta y a Distancia]. https://repository.unad.edu.co/handle/10596/38717

Toapanta Rocha, J. C. (2023). *Implementación de hardening en un sistema operativo de servidor Linux de base Red Hat* [trabajo de integración curricular, Escuela Politécnica Nacional]. Repositorio Institucional de la EPN. https://bibdigital.epn.edu.ec/handle/15000/25203