**Documentación Proyecto FireBITE**

**Realizado por Lissette Ocampo.**

**Se ha construido un sistema de gestión y monitoreo de cámaras de seguridad IP, con funcionalidades de:**

* **Visualización en tiempo real de cámaras mediante streaming HLS.**
* **Captura y almacenamiento de imágenes.**
* **Control PTZ (pan, tilt, zoom) de cámaras compatibles (aunque falta configurar el hardware).**
* **Gestión de cámaras (agregar/eliminar).**
* **Gestión de usuario mediante archivo credentials.json, api.php, dashboard.php y index.php**
* **Notificaciones visuales en la interfaz.**

**El sistema está compuesto por un frontend (HTML, CSS, JS) y un backend en PHP que interactúa con JSON para persistencia de datos, también se incluyó un sistema de gestión de usuarios con el objetivo de permitir la administración eficiente de los usuarios del sistema de cámaras PTZ. Se ofrece una interfaz sencilla que incluye las funcionalidades de crear, leer, actualizar y eliminar usuarios (CRUD), así como la asignación de roles y permisos.**

**Index.php:**

**Archivo de entrada al sistema**

1. Redirige a dashboard.html tras autenticación exitosa.
2. Verifica las credenciales y si son incorrectas muestra un mensaje de error.
3. Luego se inserta un HTML:

* Si hay errores lo muestra
* Formulario con métodos

**Elementos Clave**

* Formulario de login (username, password).
* Scripts de validación de login.
* Redirección a dashboard tras autenticación.
* Posible integración con credentials.json para validar credenciales.

**Dashboard.php:**

Archivo principal del frontend.

Proporciona la estructura HTML del sistema, incluyendo:

* Sidebar de navegación.
* Contenedor principal con secciones:
* cameraGridContainer: Grilla de cámaras, En vivo: Miniatura de cámara con las siguientes acciones:

1. Movimiento
2. Zoom
3. Capturas (Solo funciona esta)

* single-camera-view: Vista de cámara individual.
* capturesGridContainer: Galería de capturas, Checkbox y Botones: Eliminar e descargar
* settings-view: Formulario de cambio de contraseña.

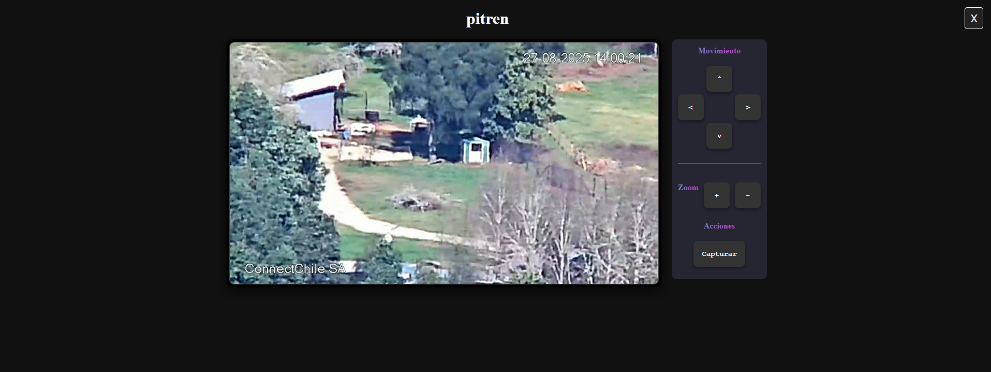
Modales:

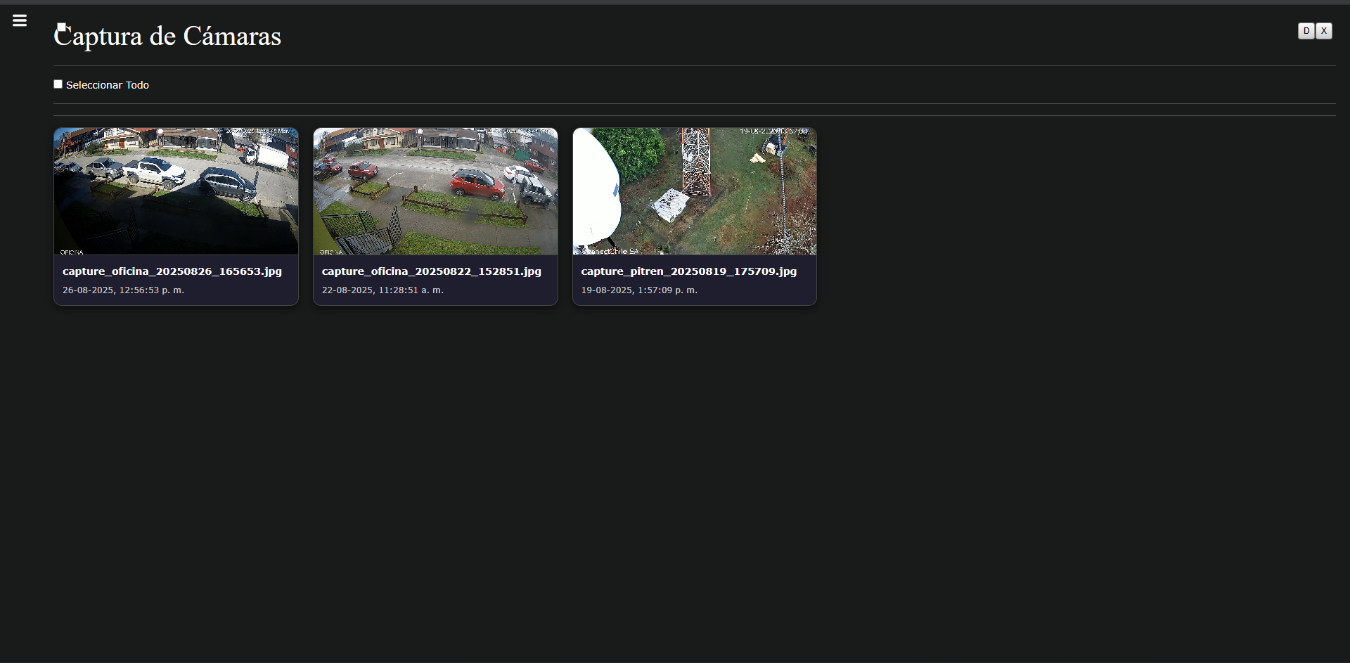
* Confirmación de cambio de contraseña.

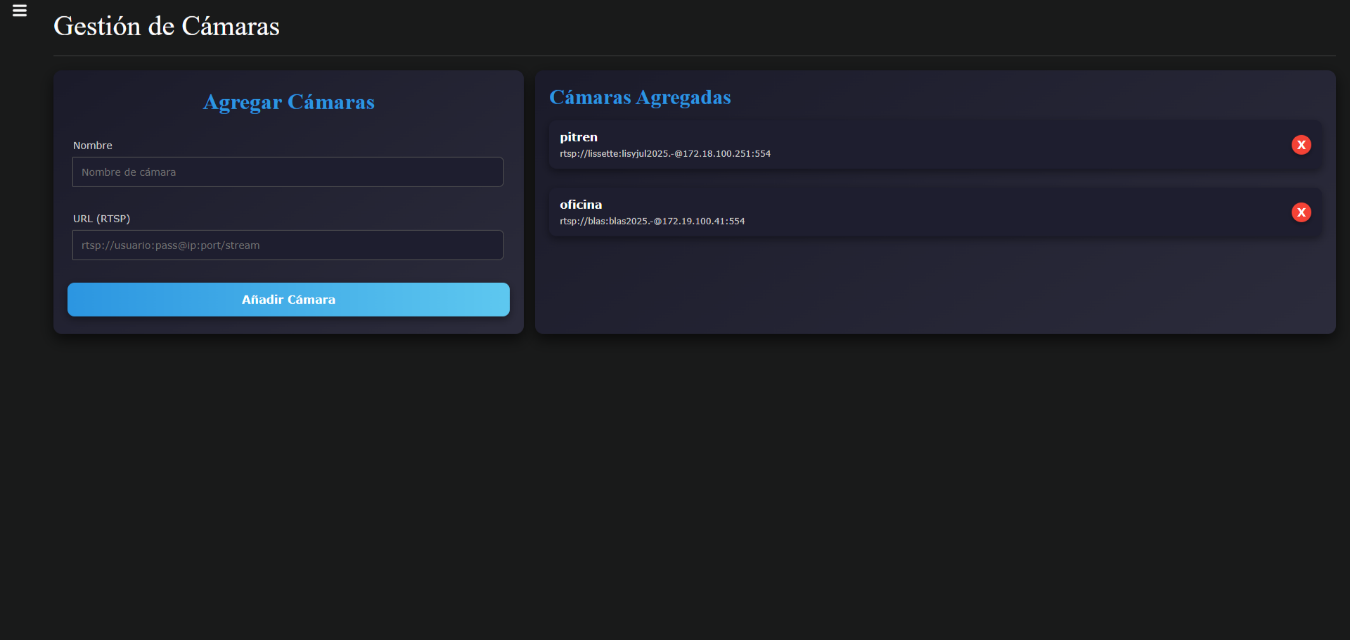
Si el usuario logra iniciar sesión recupera las credenciales mediante configuración.

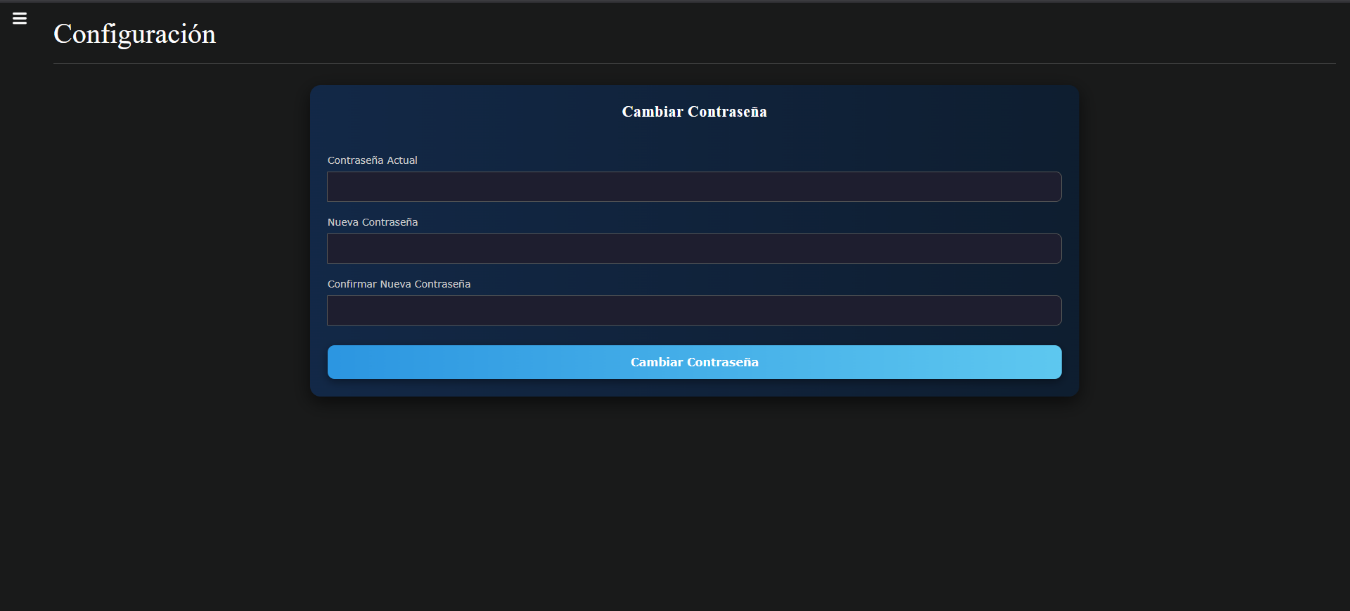
1. Dashboard y vistas de cámaras

2. Vista de Cámara (cuadricula con acciones)



3. Galería de capturas

4. Gestión de cámaras

5. Configuración

**Dashboard.js**

Archivo principal de la lógica del frontend.

**Propósito**

Controla toda la interacción del usuario con la interfaz, incluyendo:

* Navegación entre vistas.
* Visualización de cámaras en grilla y modo single view.
* Captura de imágenes y descarga/eliminación de capturas.
* Control de cámaras PTZ.
* Modales para confirmaciones y cambio de contraseña.
* Notificaciones en tiempo real.

**Variables Principales**

* cameras: Array con información de las cámaras.
* currentSingleViewIndex: Índice de la cámara mostrada en single view.
* selectedCaptures: Conjunto de capturas seleccionadas.
* notificationTimeout: Timeout para mostrar notificaciones.

**Funciones Destacadas**

1. showNotification(message, type): Muestra notificaciones tipo info, success o error.
2. fetchFromAPI (action, options): Hace llamadas al backend (api.php) y devuelve la respuesta JSON.
3. switchView (viewId, clickedLink): Cambia entre las vistas (cameras, captures, camera-settings).
4. setupHlsPlayer (videoElement, streamUrl, videoContainer): Inicializa un reproductor HLS para streaming de video.
5. displayCamerasInGrid (): Renderiza todas las cámaras en una grilla.
6. showSingleView (index): Muestra la cámara seleccionada en modo single view.
7. hideSingleView (): Oculta la cámara en single view y vuelve al dashboard.
8. toggleFullscreen (event): Alterna entre pantalla completa y normal.
9. handlePtzClick (event): Maneja los botones PTZ de una cámara.
10. takeSingleViewCapture (): Toma captura de la cámara mostrada en single view.
11. renderCaptures (): Renderiza todas las capturas guardadas.
12. handleAddCamera (event): Agrega una nueva cámara al sistema.
13. displayCameraSettings (): Muestra la lista de cámaras con opción de eliminar.

Modal de **cambiar contraseña**: Controla la apertura, confirmación y envío del cambio de contraseña.

**Logout.php**

* Elimina la sesión iniciada y finaliza la sesión del usuario.

**Config.php**

* Recupera la IP del servidor.
* Define la carpeta de capturas.
* Configura la lista de cámaras disponibles.

**Función para obtener las cámaras (array):**

1. Verifica si existe el archivo de configuración de cámaras:
   * Si existe → retorna el array.
   * Si no existe o hay errores → retorna un array vacío.
2. Define un array de cámaras.
3. Recorre el array de cámaras:
   * Se reemplaza el id de la cámara usando expresiones regulares para generar el nombre.
   * Se construye la URL del stream RTSP mediante la IP del servidor.
   * Se definen las propiedades de cada cámara con:
     + - ID de cámara.
       - Nombre de la cámara.
       - URL RTSP.
       - Tipo de stream.
       - URL del stream.
       - Soporte PTZ (movimiento).

**Cameras.json:**

Obtiene los siguientes parámetros de configuración:

* ID de cámara
* Nombre de cámara
* Rtsp\_url
* Ptz

**Api.php:**

**Archivo backend en PHP.**

Provee servicios RESTful para:

* Gestión de cámaras.
* Capturas de imágenes.
* Control PTZ.
* Cambio de contraseña.

**Comunicación con el frontend vía JSON.**

**Funcionalidades Clave**

* ***getCameras:*** Devuelve lista de cámaras.
* ***addCamera:*** Agrega cámara y guarda datos en cameras.json.
* ***deleteCamera:*** Elimina cámara de cameras.json.
* ***saveCapture:*** Guarda captura en servidor.
* ***getCaptures:*** Retorna lista de capturas.
* ***deleteCapture:*** Elimina captura.
* ***ptzControl:*** Envía comandos PTZ a la cámara.
* ***changePassword:*** Valida contraseña actual, actualiza la contraseña accediendo a ***credentials.json*** y la sobrescribe.

Responde siempre con JSON para ser consumido por dashboard.js.

Incluye el archivo o reglas ‘config.php’

Para agregar cámara se utiliza el método addCamera, que otorga una **entrada especial** en PHP que permite **leer datos sin procesar** enviados en el cuerpo de una solicitud HTTP, como **POST** o **PUT**. (HTML) Los cuales obtienen las siguientes verificaciones:

* Si hay un valor en $data['name'], lo guarda en $name **sin espacios alrededor**.  
  Si no hay nada, deja $name como una cadena vacía
* Si hay un valor en $data['rtsp\_url'], lo guarda en $rtsp **sin espacios alrededor**.  
  Si no hay nada, deja $rtsp\_url como una cadena vacía
* Si el nombre y el rtsp está vacío muestra un mensaje de alerta.
* Si el archivo de configuración de cámaras existe, lo lee, decodifica como JSON y guarda en $cameras. Si no existe, deja $cameras como un array vacío.
* Recorre una lista de cámaras y valida que no haya duplicados por nombre.
* Agrega una nueva cámara al array con sus datos.
* Guarda una nueva cámara en el archivo de configuración, si se guarda correctamente, responde con mensaje de éxito.

Para eliminar una cámara se utiliza el método ***deleteCamera*** que obtiene los siguientes parametros:

* Recibe una id de cámara
* Verifica que la id no este vacía
* Filtra la lista de cámaras, quitando la que tiene ese id.
* Si no existe, muestra error.
* Si la elimina con éxito del archivo, muestra mensaje de éxito, en caso contrario error.

Para habilitar el Control PTZ se utiliza el método ***ptzControl*** que obtiene los siguientes parametros:

* Id de cámara (si no es null)
* Acciones del PTZ (si no es null)
* Preset (si no es null)

Para realizar una captura de una cámara y guardarla se utiliza el método ***saveCapture***:

* Recibe la id de cámara y verifica que no es NULL, si es así manda un error
* Obtiene todas las cámaras mediante la función
* Busca la cámara mediante el array de camera y que coincida con la ID y no sea repetida.
* Obtiene: RTSP\_URL, IP, USER Y PASSWORD
* Busca la cámara y obtiene la imagen desde /snapshot.cgi (cámara IP).
* Configura la conexión al servidor
* Guarda la imagen en el servidor (CAPTURES\_PATH) y verifica que se pueda guardar, en caso contrario error.
* Recibe una imagen en formato Base64, La decodifica y guarda como archivo JPG en el servidor. verifica que se haya guardado en la carpeta, sino error (usa el nombre de la cámara como parte del nombre del archivo.)

Para obtener todas las capturas guardadas y listarlas se utiliza el método ***getCaptures***:

* Obtiene los archivos y los filtra por JPG
* Los ordena por fecha (más recientes primero).
* Devuelve la lista como un arreglo JSON.

Para eliminar las capturas se utiliza el método ***deleteCapture***:

* Recibe el nombre del archivo y verifica que no esté vacío
* Intenta eliminarlos del sistema.
* Informa cuántos se eliminaron correctamente y cuántos fallaron.

Para cambiar contraseña se utiliza el método ***changePassword:***

* Obtiene las credenciales del archivo JSON.
* Obtiene la contraseña y que no sea null.
* Obtiene la nueva contraseña y que no sea null.
* Verifica la nueva contraseña.
* Si es correcta, guarda la nueva en el archivo.
* Informa si el cambio fue exitoso o no.

**Acción no válida (default)**

* Si la acción no coincide con ningún caso anterior, devuelve un error genérico.

**ptz\_controller.py**

Módulo encargado del control PTZ (Pan-Tilt-Zoom) mediante el protocolo ONVIF.

**Librerías principales:**

* + **Flask** → framework web para crear la API.
  + **jsonify** → convierte datos de Python a JSON para responder al cliente.
  + **ONVIF** → librería para controlar cámaras compatibles con ONVIF.
  + **ONVIFCamera** → clase para conectarse y enviar comandos PTZ.
  + **exceptions** → maneja errores de conexión o comandos fallidos.
  + **time** → controla pausas durante movimientos.

**Conexión a la cámara:**

* + Define la **IP, usuario, contraseña, acción PTZ**.
  + Crea un servicio de **PTZ** y obtiene perfiles de video.

**Búsqueda de un perfil PTZ:**

1. Llama a GetProfiles para obtener los perfiles de la cámara.
2. Busca el que tenga configuración PTZ.
3. Si no encuentra ninguno → muestra error.
4. Obtiene velocidad de movimiento en **X, Y e Zoom**.

**Solicitud de movimiento:**

1. Detiene cualquier movimiento previo.
2. Crea una solicitud de movimiento con:
   * Dirección del vector.
   * Zoom.
   * Movimiento en X e Y.
3. Según la acción recibida:
   * Arriba → velocidad positiva en Y.
   * Abajo → velocidad negativa en Y.
   * Izquierda → velocidad negativa en X.
   * Derecha → velocidad positiva en X.
   * Zoom in → zoom positivo.
   * Zoom out → zoom negativo.
   * En otro caso → error de acción.

**Ejecución del movimiento:**

1. Obtiene velocidad y ejecuta el comando.
2. Envía la orden de movimiento.
3. Espera el tiempo definido.
4. Detiene la cámara y confirma con “Acción ejecutada”.

**API PTZ:**

1. Recibe datos en JSON (IP, usuario, contraseña, acción).
2. Valida parámetros.
   * Si faltan → devuelve error.
   * Si son correctos → mueve la cámara y responde en JSON.

**Ejecución del servidor:**

* Host: 127.0.0.1.
* Puerto: 5000.

**Connector.py**

Servidor encargado de servir los videos HLS generados por FFmpeg.

1. Importa librerías web.
2. Crea la aplicación Flask y habilita **CORS**.
3. Almacena fragmentos .ts y playlists .m3u8 generados por HLS.
4. Cuando un cliente solicita video:
   * Busca los archivos en /var/hls\_output/.
   * Envía el fragmento más reciente (sin caché).
5. Inicia el servidor en:
   * Host: 0.0.0.0.
   * Puerto: 8080.

**Video\_processor.py**

Módulo encargado del análisis de video en tiempo real con IA (**YOLO**) y generación de streaming HLS.

**Descripción:**

* Ejecuta el monitoreo de cámaras IP.
* Detecta **fuego y humo** usando YOLO en GPU.
* Procesa los frames y los envía a HLS para visualización web.

**Flujo principal:**

1. Carga el modelo YOLO en GPU.
2. Lee cámaras desde cameras.json.
3. Para cada cámara:
   * Se conecta al RTSP.
   * Si falla → reintenta.
   * Captura frames.
   * Pasa cada frame por YOLO.
   * Dibuja detecciones en el frame.
   * Envía frames procesados a HLS vía FFmpeg.
4. Si se agregan cámaras → las inicia.
5. Si se eliminan cámaras → las detiene.
6. Supervisa cambios cada CHECK\_INTERVAL.
7. Al cerrar el sistema → detiene hilos y procesos.

**Funciones principales:**

* **load\_cameras\_from\_json ()**  
  Lee cámaras desde el archivo JSON.
* **start\_ffmpeg\_process (camera\_id, frame\_size)**  
  Lanza FFmpeg para convertir frames en HLS con:
  + h264\_nvenc (GPU NVIDIA).
  + Segmentos de 1s.
  + Lista de 6 segmentos.
  + Eliminación de segmentos viejos.
* **process\_camera (camera\_config, stop\_event)**
  + Captura video desde RTSP.
  + Procesa con YOLO.
  + Envía a FFmpeg.
* **manage\_camera\_processes ()**  
  Supervisa lista de cámaras y gestiona procesos.
* **cleanup\_all\_at\_exit ()**  
  Cierra procesos y limpia al salir.

**Gestión de usuarios**

Consta de los siguientes archivos principales:

* **api.php**: Gestiona la lógica de la base de datos y las operaciones CRUD para los usuarios.
* **dashboard.php**: Proporciona la interfaz de usuario (UI) para interactuar con el sistema, mostrando y gestionando los usuarios.
* **index.php**: Página de inicio de sesión y redirección para usuarios no autenticados.
* **credentials**.json: Archivo JSON donde se almacenan los usuarios, roles y password encriptada.

**1. index.php**

Maneja el **inicio de sesión**:

* Verifica si el usuario y contraseña existen en credentials.json.
* Compara el **password** (hasheado con password\_hash y password\_verify).
* Inicia la sesión con variables $\_SESSION['username'] y $\_SESSION['role'].
* Redirige al dashboard.php si la autenticación es correcta.

**Funciones principales:**

* Validación de credenciales.
* Inicio de sesión con manejo de roles.
* Manejo de errores en login.

**2. dashboard.php**

Interfaz principal para usuarios logueados:

* Si el usuario es **operador**:  
  Puede ver la lista de usuarios, pero **no** puede modificarlos ni eliminarlos.
* Si el usuario es **admin**:  
  Tiene control completo del CRUD:
  + **Agregar usuario.**
  + **Listar usuarios.**
  + **Editar usuario.**
  + **Eliminar usuario.**

**Funciones clave:**

* Interfaz dinámica según rol (admin/operador).
* Botones estilizados para edición y eliminación visibles solo para el admin.
* Integración con api.php mediante **fetch API (JavaScript)** para hacer peticiones AJAX sin recargar la página.

**3. api.php**

Contiene toda la lógica del CRUD conectado al credentials.json.

**Acciones implementadas:**

* getUsers:  
  Retorna la lista completa de usuarios (nombre y rol).
* addUser:  
  Agrega un nuevo usuario.
  + Valida si el nombre ya existe.
  + Hashea la contraseña antes de guardarla.
  + Inserta el objeto en credentials.json.
* updateUser:  
  Modifica un usuario existente.
  + Permite cambiar contraseña y rol.
  + Actualiza el JSON.
* deleteUser:  
  Elimina un usuario del JSON de acuerdo a su username.
* saveUsers:  
  Función interna que guarda el array actualizado de usuarios en credentials.json.

**4. Scripts dentro de dashboard.php**

Este código en JavaScript controla la **interfaz dinámica de la gestión de usuarios**.  
Se conecta con api.php mediante **fetch API** y organiza cómo se muestran los usuarios y botones de acción según el **rol del usuario logueado**.

**Carga de usuarios en la lista**

* Se hace una petición fetch('api.php?action=getUsers').
* Se recorren los usuarios obtenidos desde credentials.json.
* Cada usuario se muestra en una tabla o lista dentro del dashboard.

**Control de roles (Admin vs Operador)**

* Se recibe el **rol actual del usuario** desde PHP ($\_SESSION['role']) y se envía al JS como variable.
* El JS decide si debe mostrar **botones de edición y eliminación**.
  + **Si es Admin** → botones visibles (Editar, Eliminar).
  + **Si es Operador** → no se muestran esos botones.

**Agregar usuario (solo Admin)**

* Formulario controlado desde JS.
* Envía los datos a api.php?action=addUser mediante fetch con POST.
* Valida campos vacíos antes de enviar.
* Actualiza la lista de usuarios en pantalla automáticamente.

**Editar usuario (solo Admin)**

* Al hacer clic en Editar, se abre un formulario modal con los datos del usuario.
* Se permite cambiar **contraseña** y **rol**.
* Se hace petición fetch('api.php?action=updateUser') con método POST.
* Refresca lista en pantalla al confirmar.

**Eliminar usuario (solo Admin)**

* Al hacer clic en Eliminar, se solicita confirmación (confirm("¿Seguro que deseas eliminar?")).
* Si acepta, se ejecuta fetch('api.php?action=deleteUser') con el username.
* Se actualiza la lista dinámicamente.