Explicación del Sistema

El sistema opera bajo el principio del **ciclo de vida de una solicitud HTTP** en la web (Paso 2 en el código):

1. **Solicitud (GET Inicial):** Cuando el usuario accede por primera vez a la página, el código PHP detecta que la solicitud **NO** es un POST ($_SERVER["REQUEST_METHOD"] == "POST" es falso), por lo que omite la lógica de validación y muestra el **Formulario** (Paso 4, else).
2. **Envío (POST):** Cuando el usuario llena el campo y hace clic en "Enviar," el navegador realiza una solicitud **POST** a la misma página.
3. **Procesamiento (Validación y Sanitización):** El código PHP detecta la solicitud POST y ejecuta la lógica de validación (Pasos 2 y 3).
4. **Resultado:**
    - **Éxito:** Si la entrada es válida, se establece la bandera $exito = true y se muestra el **Mensaje de Éxito** (Paso 4, if ($exito)).
    - **Fallo:** Si la entrada no pasa alguna prueba, se guarda el error en $error_mensaje, $exito sigue siendo false, y se vuelve a mostrar el **Formulario** con el mensaje de error y manteniendo el valor ingresado por el usuario (value="<?php echo htmlspecialchars($entrada_usuario); ?>">).

Por Qué Usarlo en Entornos Empresariales (La Seguridad es Prioridad)

Implementar la **Validación y Sanitización del Lado del Servidor** (como se ve en el código) es **obligatorio** en cualquier entorno empresarial por las siguientes razones fundamentales:

1. Seguridad (Prevención de Ataques) 🛡️

La validación del lado del cliente (hecha con JavaScript en el navegador) es **fácilmente evitable** por un atacante. Un actor malicioso puede desactivar JavaScript, o usar herramientas para enviar directamente una solicitud POST con datos inválidos o código malicioso.

- **Prevención de Inyección SQL:** Si un atacante ingresa una instrucción SQL maliciosa en un formulario no validado, esta podría ejecutarse en la base de datos (DB), comprometiendo datos sensibles. La sanitización limpia o rechaza esas entradas.
- **Prevención de Cross-Site Scripting (XSS):** Si el atacante ingresa código JavaScript (scripts), y este se almacena y luego se muestra a otros usuarios, podría robar sus sesiones. La función htmlspecialchars() utilizada en la salida (<?php echo htmlspecialchars($entrada_usuario); ?>) neutraliza cualquier código malicioso al transformarlo en texto inofensivo.

2. Integridad y Fiabilidad de los Datos 📊

Las empresas dependen de la **calidad** de sus datos. Este sistema garantiza que solo la información que cumple con los **requisitos de la empresa** llegue a la base de datos.

- **Consistencia:** Asegura que un campo de "Edad" siempre sea un número (Sanitización de Tipo) y que esté dentro de un rango realista (Validación de Lógica), evitando errores en informes, cálculos de nómina, o análisis de datos.
- **Fiabilidad de la Lógica de Negocio:** Si el negocio requiere que una cantidad sea positiva, la validación de lógica refuerza esa regla antes de que el dato cause una excepción o un error en otro módulo del sistema (ej. un cálculo financiero).

3. Rendimiento del Servidor y Experiencia del Usuario (UX) 🚀

- **Menos Carga en el Servidor:** Al rechazar rápidamente los datos con el formato incorrecto, se evita que los sistemas de *backend* (como la base de datos) tengan que procesar o almacenar datos basura, lo que ahorra recursos.
- **Retroalimentación Clara:** El manejo explícito de $error_mensaje permite mostrar al usuario un mensaje específico y útil ("Solo se aceptan números enteros positivos"), en lugar de un mensaje de error genérico del servidor, lo que mejora la **experiencia del usuario**.