Index html:  
  
**Buenas Prácticas Observadas**

1. **Uso de DOCTYPE y Metadatos**:
   * Se incluye el <!DOCTYPE html> para asegurar que el navegador interprete el documento como HTML5.
   * Se utiliza <meta charset="UTF-8"> para definir la codificación de caracteres.
   * Se incluye <meta name="viewport" content="width=device-width, initial-scale=1.0"> para asegurar la correcta visualización en dispositivos móviles.
2. **Uso de Etiquetas Semánticas**:
   * Uso de etiquetas como <label>, <input>, <div>, <form>, <button>, y <h1> que mejoran la accesibilidad y el SEO.
3. **Validación de Formulario**:
   * Se utilizan atributos como required, pattern, y maxlength para validar la entrada del usuario.
   * Se incluye retroalimentación para entradas inválidas con la clase invalid-feedback
4. **Separación de CSS y JavaScript**:
   * Los archivos CSS y JavaScript están separados del HTML, lo cual es una buena práctica para la mantenibilidad del código.

JS de empresa:

**Buenas Prácticas Observadas**

1. **Uso de const y let**:
   * Se utilizan const para las constantes y let para variables que pueden cambiar, lo que es una buena práctica para mantener la inmutabilidad y claridad en el código.
2. **Funciones Asíncronas (async/await)**:
   * El uso de async/await facilita la lectura y el manejo de código asíncrono, en lugar de usar promesas (.then).
3. **Manipulación del DOM**:
   * Se utilizan métodos como getElementById y addEventListener para manipular el DOM de manera efectiva.
4. **Separación de Concerns**:
   * El código está bien organizado en funciones específicas para manejar diferentes tareas, como openCreate, readOne, fillTable, openDelete, y openUpdate.
5. **Comentarios Descriptivos**:
   * Los comentarios detallados ayudan a entender el propósito de cada función y los parámetros que utilizan.

JS de categoría:

 **Uso de Constantes para Referencias de Elementos del DOM**:

* Se utilizan constantes para almacenar referencias a elementos del DOM, lo que asegura que estas referencias no sean reasignadas accidentalmente y mejora la legibilidad del código.

 **Uso de addEventListener para Manejar Eventos**:

* addEventListener es utilizado para agregar manejadores de eventos en lugar de métodos obsoletos, lo que es una práctica moderna y más flexible.

 **Funciones Asíncronas y await**:

* Se emplean funciones asíncronas y la palabra clave await para manejar operaciones asincrónicas, lo que mejora la legibilidad del código y permite un flujo de trabajo más lógico en lugar de usar promesas encadenadas.

 **Prevención del Comportamiento Predeterminado del Formulario**:

* El método event.preventDefault() se usa para evitar que los formularios recarguen la página al ser enviados, permitiendo un control total sobre el comportamiento del formulario.

 **Uso de FormData para Manejar Datos del Formulario**:

* FormData se utiliza para recoger y manejar datos del formulario de manera eficiente y sencilla, facilitando la transmisión de datos en peticiones asíncronas.

 **Separación de Responsabilidades**:

* Las funciones están bien definidas para realizar tareas específicas, lo que mejora la claridad y el mantenimiento del código al tener responsabilidades claramente separadas.

 **Manejo de Respuestas del Servidor**:

* Se comprueba el estado de las respuestas del servidor (data.status) para manejar correctamente tanto las respuestas exitosas como los errores, proporcionando retroalimentación adecuada al usuario.

 **Manipulación del DOM dentro de Bucles**:

* Las manipulaciones del DOM se minimizan mediante la construcción del contenido en un solo paso y luego la inserción en el DOM, lo que mejora el rendimiento.

 **Uso de Template Literals**:

* Se emplean template literals para construir cadenas de HTML, lo que mejora la legibilidad y la gestión del código al evitar concatenaciones complicadas.

 **Uso de try-catch para Manejo de Errores**:

* Los bloques try-catch se utilizan para manejar errores y excepciones, mejorando la robustez del código y facilitando la detección y gestión de problemas.

Clientes php:

 **Inclusión de Clases Necesarias**:

* Uso de require\_once para incluir archivos necesarios, evitando inclusiones múltiples y errores relacionados.

 **Comprobación de la Existencia de la Acción**:

* Verificación inicial para asegurar que una acción ha sido pasada al script, evitando ejecuciones innecesarias y posibles errores.

 **Manejo de Sesiones**:

* Uso de session\_start() para gestionar sesiones de usuario y permitir el uso de variables de sesión.

 **Instanciación de Clases**:

* Creación de instancias de clases necesarias solo cuando se verifica que existe una acción a realizar, mejorando la eficiencia.

 **Inicialización de Variables**:

* Inicialización de un arreglo $result para almacenar resultados y respuestas, asegurando consistencia en el formato de respuesta.

 **Verificación de Sesiones Activas**:

* Comprobación de si un usuario ha iniciado sesión antes de permitir la ejecución de acciones específicas, mejorando la seguridad.

 **Estructura de Control Claramente Definida**:

* Uso de switch para manejar diferentes acciones de manera clara y estructurada, facilitando la lectura y el mantenimiento del código.

 **Validación de Datos**:

* Validación de entradas mediante funciones específicas como Validator::validateSearch() y Validator::validateForm(), mejorando la seguridad y la integridad de los datos.

 **Manejo de Errores**:

* Asignación de mensajes de error claros en caso de fallos en la validación o ejecución de operaciones, proporcionando retroalimentación útil para el usuario.

 **Uso de Métodos de Clases para Operaciones CRUD**:

* Uso de métodos de clases como createRow(), readAll(), updateRow(), y deleteRow() para realizar operaciones CRUD, manteniendo una separación clara entre la lógica de negocios y la lógica de presentación.

 **Manejo de Respuestas del Servidor**:

* Verificación del estado de las respuestas ($result['status']) para manejar tanto respuestas exitosas como errores, y proporcionar mensajes adecuados.

 **Encabezados HTTP Correctos**:

* Uso de header('Content-type: application/json; charset=utf-8') para definir el tipo de contenido de la respuesta, asegurando que los datos se interpreten correctamente en el cliente.

 **Salida en Formato JSON**:

* Uso de print(json\_encode($result)) para devolver respuestas en formato JSON, lo que es estándar y fácilmente manejable en aplicaciones web modernas.

 **Comentarios Claros**:

* Uso de comentarios para explicar secciones clave del código, mejorando la comprensión y el mantenimiento del código.

 **Seguridad en el Manejo de Datos del reCAPTCHA**:

* Validación de reCAPTCHA para evitar bots y mejorar la seguridad de la aplicación.

Usuario php:  
  
 **Inclusión de Clases Necesarias**:

* Uso de require\_once para incluir archivos necesarios, evitando inclusiones múltiples y errores relacionados.

 **Comprobación de la Existencia de la Acción**:

* Verificación inicial para asegurar que una acción ha sido pasada al script, evitando ejecuciones innecesarias y posibles errores.

 **Manejo de Sesiones**:

* Uso de session\_start() para gestionar sesiones de usuario y permitir el uso de variables de sesión.

 **Instanciación de Clases**:

* Creación de instancias de clases necesarias solo cuando se verifica que existe una acción a realizar, mejorando la eficiencia.

 **Inicialización de Variables**:

* Inicialización de un arreglo $result para almacenar resultados y respuestas, asegurando consistencia en el formato de respuesta.

 **Verificación de Sesiones Activas**:

* Comprobación de si un usuario ha iniciado sesión antes de permitir la ejecución de acciones específicas, mejorando la seguridad.

 **Estructura de Control Claramente Definida**:

* Uso de switch para manejar diferentes acciones de manera clara y estructurada, facilitando la lectura y el mantenimiento del código.

 **Validación de Datos**:

* Validación de entradas mediante funciones específicas como Validator::validateSearch() y Validator::validateForm(), mejorando la seguridad y la integridad de los datos.

 **Manejo de Errores**:

* Asignación de mensajes de error claros en caso de fallos en la validación o ejecución de operaciones, proporcionando retroalimentación útil para el usuario.

 **Uso de Métodos de Clases para Operaciones CRUD**:

* Uso de métodos de clases como createRow(), readAll(), updateRow(), y deleteRow() para realizar operaciones CRUD, manteniendo una separación clara entre la lógica de negocios y la lógica de presentación.

 **Manejo de Respuestas del Servidor**:

* Verificación del estado de las respuestas ($result['status']) para manejar tanto respuestas exitosas como errores, y proporcionar mensajes adecuados.

 **Encabezados HTTP Correctos**:

* Uso de header('Content-type: application/json; charset=utf-8') para definir el tipo de contenido de la respuesta, asegurando que los datos se interpreten correctamente en el cliente.

 **Salida en Formato JSON**:

* Uso de print(json\_encode($result)) para devolver respuestas en formato JSON, lo que es estándar y fácilmente manejable en aplicaciones web modernas.

 **Comentarios Claros**:

* Uso de comentarios para explicar secciones clave del código, mejorando la comprensión y el mantenimiento del código.

 **Seguridad en el Manejo de Archivos**:

* Validación y manejo seguro de archivos subidos por el usuario ($\_FILES), incluyendo la verificación y almacenamiento seguro de archivos.

 **Manejo de Excepciones de Base de Datos**:

* Asignación de cualquier excepción de la base de datos al arreglo de resultados para asegurar que se captura y maneja adecuadamente cualquier error del servidor de base de datos.

 **Separación de Funciones de Sesión**:

* Diferenciación clara de acciones permitidas dentro y fuera de una sesión iniciada, asegurando que solo usuarios autenticados puedan realizar ciertas acciones.