Documentación de la Clase `CrudHandler`

La clase `CrudHandler` proporciona funcionalidades para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en una entidad específica a través de una API. Se utiliza para simplificar la gestión de datos y la interacción con una interfaz de usuario.

Constructor

Parámetros:

`apiClient`: Instancia de la clase `ApiClient` que maneja las solicitudes a la API.

`entity`: Nombre de la entidad que se gestionará a través de la clase.

`config`: Configuración que incluye elementos como tablas, modal, botones y otros elementos necesarios para la funcionalidad.

```
const crudHandler = new CrudHandler(
  new ApiClient(urlRest),
  "type_accounts",
  config
);
```

Métodos Principales

`init()`

Inicializa la clase, configurando la tabla, los manejadores de eventos y cargando los datos de la entidad.

`initTable()`

Configura las tablas utilizando el plugin Bootstrap Table.

`showModal()`

Muestra el modal de edición.

`hideModal()`

Oculta el modal de edición.

`initEventHandlers()`

Configura los manejadores de eventos para los botones de guardar, editar, actualizar, eliminar y restaurar.

`loadDataIndex()`

Carga los datos de la entidad principal y actualiza la tabla correspondiente.

`loadDataDelete()`

Carga los datos de la entidad eliminada y actualiza la tabla correspondiente.

`postClientInsert()`

Realiza una solicitud POST para insertar un nuevo registro en la entidad.

`postClientUpdate()`

Realiza una solicitud POST para actualizar un registro existente en la entidad.

`getDeleteClient(event)`

Realiza una solicitud GET para eliminar un registro de la entidad.

`getRestoreClient(event)`

Realiza una solicitud GET para restaurar un registro eliminado.

`deleteClient(event)`

Muestra un cuadro de diálogo para confirmar la eliminación de un registro.

`restoreClient(event)`

Muestra un cuadro de diálogo para confirmar la restauración de un registro eliminado.

`actionsAfterUpdate(dataClient)`

Realiza acciones específicas después de actualizar un registro, como actualizar la interfaz de usuario y ocultar el modal.

`loadDataEdit(event)`

Carga los datos de un registro específico para su edición en el modal.

`assembleObjectClient()`

Recopila datos de entrada del usuario para crear un nuevo objeto de cliente.

`assembleObjectClientEdit()`

Recopila datos de entrada del usuario para editar un objeto de cliente existente.

`getNextId()`

Obtiene el siguiente ID disponible para un nuevo registro.

`loadTableIndex()`

Carga los datos de la entidad principal en la tabla correspondiente.

`loadTableDelete()`

Carga los datos de la entidad eliminada en la tabla correspondiente.

`cleanInput()`

Limpia los campos de entrada en el modal.

`handleSuccessfulResponse(response, dataClient)`

Maneja una respuesta exitosa después de insertar un nuevo registro.

`handleSuccessfulDelete(response, id)`

Maneja una respuesta exitosa después de eliminar un registro.

`handleSuccessfulRestore(response, id)`

Maneja una respuesta exitosa después de restaurar un registro eliminado.

`handleError(error)`

Maneja errores de solicitud, mostrando mensajes específicos de acuerdo al tipo de error.

`showAlert(icon, title, time)`

Muestra una alerta visual utilizando la librería SweetAlert.

`showAlertDialog(event, callback, title, buttonText)`

Muestra un cuadro de diálogo con opciones de confirmación y ejecuta una acción según la respuesta del usuario.

`validateInsert(labels)`

Valida los campos de entrada antes de realizar una operación de inserción o actualización.

`validation(input)`

Realiza la validación de un campo de entrada específico.

Uso de la Clase en la Aplicación

Ejemplo de Configuración (**`config`**):

```
const config = {
  table: $("#table"),
  tableDeletes: $("#table-deletes"),
  modal: $("#modal-edit"),
  loader: $(".page-loader-wrapper"),
  saveBtn: "#btn-save",
  updateBtn: "#btn-update",
  editBtn: ".edit",
  deleteBtn: ".delete",
  restoreBtn: ".restore",
  selectors: ["name", "description", "id"],
  selectorsNotNull: ["name"],
  formatActions: formatActions,
  formatActionsDelete: formatActionsDelete,
};
```

Ejemplo de Creación de una Instancia de **`CrudHandler`**:

```
const crudHandler = new CrudHandler(
  new ApiClient(urlRest),
  "type_accounts",
  config
);
```

Ejemplo de Configuración de Datos Iniciales y Ejecución de Operación de Inserción:

```
const crudHandler = new CrudHandler(
  new ApiClient(urlRest),
  "type_accounts",
  config
);

crudHandler.dataClient = {
  name: "static",
  description: "static"
}
```

Ejemplo tabla plana

```
/**
 * Objeto de configuración para el manejo del CRUD.
 * @typedef {Object} CrudConfig
 * @property {JQuery} table - Objeto jQuery que representa la tabla principal.
 * @property {JQuery} tableDeletes - Objeto jQuery que representa la tabla de elementos eliminados.
 * @property {JQuery} modal - Objeto jQuery que representa el modal de edición.
 * @property {JQuery} loader - Objeto jQuery que representa el loader en la página.
 * @property {string} saveBtn - Selector del botón para guardar nuevos elementos.
 * @property {string} updateBtn - Selector del botón para actualizar elementos existentes.
 * @property {string} showModalEditBtn - Selector de los botones que muestran el modal de edición.
 * @property {string} showDeleteAlertBtn - Selector de los botones que muestran la alerta de eliminación.
 * @property {string} showRestoreAlertBtn - Selector de los botones que muestran la alerta de restauración.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "independent" p "secondary").
 */
// Configuración para el manejo del crud
const config = {
  table: $("#table"),
  tableDeletes: $("#table-deletes"),
  modal: $("#modal-edit"),
  loader: $(".page-loader-wrapper"),
  saveBtn: "#btn-save",
  updateBtn: "#btn-update",
  showModalEditBtn: ".edit",
  showDeleteAlertBtn: ".delete",
  showRestoreAlertBtn: ".restore",
  selectors: ["name", "description", "id"],
  selectorsNotNull: ["name"],
  nameTypeTable: "independent"
};
// Uso de la clase para el manejo de categorias
const crudHandler = new CrudHandler(
  new ApiClient(urlRest),
  "categories",
  config
);
```

Ejemplo tabla del lado muchos

```
/**
 * Objeto de configuración para el manejo del CRUD.
 * @typedef {Object} CrudConfig
 * @property {JQuery} table - Objeto jQuery que representa la tabla principal.
 * @property {JQuery} tableDeletes - Objeto jQuery que representa la tabla de elementos eliminados.
 * @property {JQuery} modal - Objeto jQuery que representa el modal de edición.
 * @property {JQuery} loader - Objeto jQuery que representa el loader en la página.
 * @property {string} saveBtn - Selector del botón para guardar nuevos elementos.
 * @property {string} updateBtn - Selector del botón para actualizar elementos existentes.
 * @property {string} removeIdsBtn - Selector del botón para eliminar elementos seleccionados.
 * @property {string} restoreIdsBtn - Selector del botón para restaurar elementos seleccionados.
 * @property {string} showModalEditBtn - Selector de los botones que muestran el modal de edición.
 * @property {string} showDeleteAlertBtn - Selector de los botones que muestran la alerta de eliminación.
 * @property {string} showRestoreAlertBtn - Selector de los botones que muestran la alerta de restauración.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {boolean} contentImage - Indica si se maneja contenido de imágenes.
 * @property {string[]} filesInputImage - Lista de campos que representan entradas de archivos de imagen.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "secondary" o "independent").
 * @property {Object[]} tableRelation - Configuración para las relaciones de la tabla principal.
 * @property {string} tableRelation[].nameSecondary - Nombre del recurso secundario relacionado.
 * @property {string[]} tableRelation[].nameIndex - Campos a mostrar como índices del recurso secundario.
 * @property {string} tableRelation[].select - Selector del elemento de interfaz para seleccionar la relación.
 * @property {string} tableRelation[].foreignKey - Clave foránea en la tabla principal que se relaciona con la
secundaria.
 */

/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Instancia de la clase ApiClient para interactuar con la API.
 * @param {string} resourceName - Nombre del recurso que se manejará (por ejemplo, "subcategories").
 * @param {CrudConfig} config - Configuración para el manejo del CRUD.
*/
```

```
// Configuración para el manejo del crud
const config = {
  table: $("#table"),
  tableDeletes: $("#table-deletes"),
  modal: $("#modal-edit"),
  loader: $(".page-loader-wrapper"),
  saveBtn: "#btn-save",
  updateBtn: "#btn-update",
  removeIdsBtn: "#btn-remove",
  restoreIdsBtn: "#btn-restore",
  showModalEditBtn: ".edit",
  showDeleteAlertBtn: ".delete",
  showRestoreAlertBtn: ".restore",
  selectors: ["name", "description", "id", "id_category"],
  selectorsNotNull: ["name"],
  contentImage: true,
  filesInputImage: ["image"],
  nameTypeTable: "secondary", //independent
  tableRelation: [
    {
      nameSecondary: "categories",
      nameIndex: ["name", "description"],
      select: "#id_category",
      foreignKey: "id_category",
    },
  ],
};

// Uso de la clase para el manejo del crud
const crudHandler = new CrudHandler(
  new ApiClient(urlRest),
  "subcategories",
  config
);
```

Ejemplo de uso con tabla del lado muchos incluyendo el extra de detailformater

```
/**
 * Objeto de configuración para el manejo del CRUD.
 * @typedef {Object} CrudConfig
 * @property {JQuery} table - Objeto jQuery que representa la tabla principal.
 * @property {JQuery} tableDeletes - Objeto jQuery que representa la tabla de elementos eliminados.
 * @property {JQuery} modal - Objeto jQuery que representa el modal de edición.
 * @property {JQuery} loader - Objeto jQuery que representa el loader en la página.
 * @property {string} saveBtn - Selector del botón para guardar nuevos elementos.
 * @property {string} updateBtn - Selector del botón para actualizar elementos existentes.
 * @property {string} removeIdsBtn - Selector del botón para eliminar elementos seleccionados.
 * @property {string} restoreIdsBtn - Selector del botón para restaurar elementos seleccionados.
 * @property {string} showDeleteAlertBtn - Selector de los botones que muestran la alerta de eliminación.
 * @property {string} showRestoreAlertBtn - Selector de los botones que muestran la alerta de restauración.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {boolean} contentImage - Indica si se maneja contenido de imágenes.
 * @property {string[]} filesInputImage - Lista de campos que representan entradas de archivos de imagen.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "secondary" o "independent").
 * @property {Object[]} tableRelation - Configuración para las relaciones de la tabla principal.
 * @property {string} tableRelation[].entityRest - Nombre del recurso relacionado.
 * @property {string} tableRelation[].nameSecondary - Nombre del recurso secundario relacionado.
 * @property {string[]} tableRelation[].nameIndex - Campos a mostrar como índices del recurso secundario.
 * @property {string} tableRelation[].select - Selector del elemento de interfaz para seleccionar la relación.
 * @property {string} tableRelation[].foreignKey - Clave foránea en la tabla principal que se relaciona con la
secundaria.
 * @property {Object[]} tableRelation[].with - Configuración adicional para relaciones secundarias.
 * @property {string} tableRelation[].with[].nameSecondary - Nombre del recurso secundario adicional.
 * @property {string[]} tableRelation[].with[].nameIndex - Campos a mostrar como índices del recurso
secundario adicional.
 * @property {Object} contentDetailFormatter - Configuración para el formateo de detalles del contenido.
 * @property {string} contentDetailFormatter.nameAttributeDetail - Nombre del atributo de detalle del
producto.
 * @property {string[]} contentDetailFormatter.attributes - Lista de atributos a incluir en el detalle.
 * @property {string[]} contentDetailFormatter.images - Lista de campos de imágenes a incluir en el detalle.
 * @property {Object} contentDetailFormatter.configTableDetail - Configuración para la tabla de detalles.
 * @property {string} contentDetailFormatter.configTableDetail.headTitle - Título de la tabla de detalles.
 * @property {string[]} contentDetailFormatter.configTableDetail.headers - Encabezados de la tabla de
detalles.
 */
```

```
/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Instancia de la clase ApiClient para interactuar con la API.
 * @param {string} resourceName - Nombre del recurso que se manejará (por ejemplo, "products").
 * @param {CrudConfig} config - Configuración para el manejo del CRUD.
 */

// Configuración para el manejo del crud
const config = {
  table: $("#table"),
  tableDeletes: $("#table-deletes"),
  modal: $("#modal-edit"),
  loader: $(".page-loader-wrapper"),
  saveBtn: "#btn-save",
  updateBtn: "#btn-update",
  removeIdsBtn: "#btn-remove",
  restoreIdsBtn: "#btn-restore",
  // showModalEditBtn: "none",
  showDeleteAlertBtn: ".delete",
  showRestoreAlertBtn: ".restore",
  selectors: [
    "id",
    "name",
    "description",
    "price_buys",
    "price_sale",
    "priority",
    "information_general",
    "information_important",
    "information_warning",
    "id_subcategory",
  ],
  selectorsNotNull: ["name"],
  contentImage: true,
  filesInputImage: [
    "image_1",
    "image_2",
    "image_3",
    "image_detail_1",
    "image_detail_2",
    "image_detail_3",
  ],
  nameTypeTable: "secondary", //independent
  tableRelation: [
    {
      entityRest: "subcategories",
      nameSecondary: "subcategory",
      nameIndex: ["name"],
      select: "#id_subcategory",
      foreignKey: "id_subcategory",
      with: [
        {
          nameSecondary: "category",
          nameIndex: ["name"],
        },
      ],
    },
  ],
  contentDetailFormatter: {
    nameAttributeDetail: "detail_product",
    attributes: [
      "information_general",
      "information_important",
      "information_warning",
    ],
    images: ["image_1", "image_2", "image_3"],
    configTableDetail: {
      headTitle: "Detalle importante",
      headers: [
        "información general",
        "información importante",
        "Precaución",
        "imagenes",
      ],
    },
  },
};

// Uso de la clase para el manejo del crud
const crudHandler = new CrudHandler(new ApiClient(urlRest), "products", config);

/**
 * Función para formatear los detalles de un elemento.
 * @param {number} index - Índice del elemento.
 * @param {Object} row - Datos del elemento.
```

```
 * @returns {string} - Detalles formateados del elemento.
*/

function detailFormatter(index, row) {
  return crudHandler.detailFormatterOne(index, row);
}
```

Ejemplo de index para una tabla de relación de muchos a muchos, ejemplo con Catálogos y productos, tercer tabla catalogo_producto.

Nota. – Para ver el detailFormater en el index son las funciones extras

```
/**
 * Objeto de configuración para el manejo del CRUD.
 * @typedef {Object} CrudConfig
 * @property {JQuery} table - Objeto jQuery que representa la tabla principal.
 * @property {JQuery} tableDeletes - Objeto jQuery que representa la tabla de elementos eliminados.
 * @property {JQuery} loader - Objeto jQuery que representa el loader en la página.
 * @property {string} saveBtn - Selector del botón para guardar nuevos elementos.
 * @property {string} removeIdsBtn - Selector del botón para eliminar elementos seleccionados.
 * @property {string} restoreIdsBtn - Selector del botón para restaurar elementos seleccionados.
 * @property {string} showModalEditBtn - Selector de los botones que muestran el modal de edición.
 * @property {string} showDeleteAlertBtn - Selector de los botones que muestran la alerta de eliminación.
 * @property {string} showRestoreAlertBtn - Selector de los botones que muestran la alerta de restauración.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {boolean} contentImage - Indica si se maneja contenido de imágenes.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "independent" o "secondary").
 */



/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Instancia de la clase ApiClient para interactuar con la API.
 * @param {string} resourceName - Nombre del recurso que se manejará (por ejemplo, "products").
 * @param {CrudConfig} config - Configuración para el manejo del CRUD.
 */
// Configuración para el manejo del crud
const config = {
    table: $("#table"),
    tableDeletes: $("#table-deletes"),
    loader: $(".page-loader-wrapper"),
    saveBtn: "#btn-save",
    removeIdsBtn: "#btn-remove",
    restoreIdsBtn: "#btn-restore",
    removeIdsBtn: "#btn-remove",
    restoreIdsBtn: "#btn-restore",
    showModalEditBtn: ".edit",
    showDeleteAlertBtn: ".delete",
    showRestoreAlertBtn: ".restore",
    selectors: ["date", "show_catalog", "id"],
    selectorsNotNull: ["name"],
    contentImage: false,
    nameTypeTable: "independent",
  };

  // Uso de la clase para el manejo del crud
  const crudHandler = new CrudHandler(
    new ApiClient(urlRest),
    endpoint,
    config
  );


/**
 * Función para formatear los detalles de un elemento.
 * @param {number} index - Índice del elemento.
 * @param {Object} row - Datos del elemento.
 * @returns {string} - Detalles formateados del elemento.
 */
function detailFormatter(index, row) {
    contentDetail = `
    <div class="content-detail col-12">
        <h6 class="text-center bg-info text-white">Productos del catálogo</h6>
```

```
        <table class="col-12">
            <thead>
                  <tr class="text-dark">
                      <th>ID</th>
                      <th>Nombre</th>
                      <th>Descripción</th>
                      <th>Precio venta</th>
                      <th>Subcategoria</th>
                      <th>Cantidad</th>
                  </tr>
            </thead>
            <tbody>
                ${detailRow(row)}
            </tbody>
        </table>
    </div>
    `;

    return contentDetail;
}

 /**
 * Función para formatear las filas de detalles de productos.
 * @param {Object} row - Datos del elemento.
 * @returns {string} - Filas de detalles formateados.
 */
function detailRow (row) {
  let content = "";
  row.products.forEach((element, index) => {
      const rowClass = index % 2 == 0 ? "even-row" : "odd-row"
      content += `
          <tr class="${rowClass}">
              <td>${element.id_product}</td>
              <td>${element.product.name}</td>
              <td>${element.product.description}</td>
              <td>${element.product.price_sale}</td>
              <td>${element.product.subcategory.name}</td>
              <td>${element.amount}</td>
          </tr>
      `;
  });
  return content;
}
```

Ejemplo de crear para una relación de uno a muchos, esto en una nueva página, por ende se instancia otro objecto crudHandler

```
let cart = [];
const selectProducts = $("#id_product");
const tableProducts = $("#table-products");
const tableCartTemp = $("#table-products-temp");
const modal = $("#modal-products");
const inputCheck = $("#show_catalog");
const inputDate = $("#date");

document.addEventListener('DOMContentLoaded', () => {
    inputDate.val(getDate());
});

const formatActionsRelation = (element) => {
    return `
             <button class="btn btn-sm btn-info mx-1"><i data-id="${element.id}" class="fa fa-plus add-
cart"></i></button>
          `;
};

const formatActionsTemp = (element) => {
    const actions = {
      acciones: `<button data-id="${element.id}" class="btn btn-sm btn-danger mx-1 delete-cart"><i class="fa
fa-trash delete-cart"  data-id="${element.id}"></i></button>`,
      amount_input: `<input data-id="${element.id}" type="number" value="${element.amount ?? 1}" class="form-
control input-amount">`,
    };
    return actions;
};
```

```
/**
 * Objeto de configuración para el manejo del CRUD de descuentos.
 * @typedef {Object} CrudConfig
 * @property {JQuery} loader - Objeto jQuery que representa el loader principal.
 * @property {JQuery} loaderSecondary - Objeto jQuery que representa el loader secundario.
 * @property {JQuery} tableCartRelation - Objeto jQuery que representa la tabla de productos relacionada.
 * @property {JQuery} tableCartTemp - Objeto jQuery que representa la tabla temporal de productos.
 * @property {JQuery} modalTemp - Objeto jQuery que representa el modal de productos temporales.
 * @property {string} showModalTempBtn - Selector del botón para mostrar el modal de productos temporales.
 * @property {string} addProductBtn - Selector del botón para agregar productos al carrito.
 * @property {string} saveTempBtn - Selector del botón para guardar productos temporales.
 * @property {string} addCartBtn - Selector de los botones para agregar productos al carrito.
 * @property {string} deleteCartBtn - Selector de los botones para eliminar productos del carrito.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {string} cartRelation - Nombre de la relación de carrito en la entidad principal.
 * @property {boolean} hasMany - Booleano que indica si es una tabla con relación de muchos a muchos.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "secondary").
 * @property {Object[]} tableRelation - Configuración para las relaciones de la tabla principal.
 * @property {string} tableRelation[].entityRest - Nombre del recurso relacionado.
 * @property {string} tableRelation[].nameSecondary - Nombre del recurso secundario relacionado.
 * @property {string} tableRelation[].select - Selector del elemento de interfaz para seleccionar la relación.
 * @property {Function} formatActionsRelation - Función para formatear las acciones de la relación.
 * @property {Function} formatActionsTemp - Función para formatear las acciones de productos temporales.
 */

// Configuración para el manejo del crud
const config = {
  loader: $(".page-loader-wrapper"),
  loaderSecondary: $("#loader-container"),
  inputChecks: [inputCheck],
  tableCartRelation: tableProducts,
  tableCartTemp: tableCartTemp,
  modalTemp: modal,
  showModalTempBtn: "#btn-modal-products",
  addProductBtn: "#btn-add-products",
  saveTempBtn: "#btn-save",
  addCartBtn: ".add-cart",
  deleteCartBtn: ".delete-cart",
  inputAmountBtn: ".input-amount",
  selectors: ["id", "date", "show_catalog"],
  selectorsNotNull: ["date"],
  cartRelation: "products",
  hasMany: true,
  nameTypeTable: "secondary",
  tableRelation: [
    {
      entityRest: "products",
      nameSecondary: "products",
      select: "#id_product",
    },
  ],
  formatActionsRelation: formatActionsRelation,
  formatActionsTemp: formatActionsTemp
};

/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Instancia de la clase ApiClient para interactuar con la API.
 * @param {string} resourceName - Nombre del recurso que se manejará (por ejemplo, "discounts").
 * @param {CrudConfig} config - Configuración para el manejo del CRUD de descuentos.
 */

const apiClient = new ApiClient(urlRest);
const entity = "catalogs"
const crudHandler = new CrudHandler(apiClient, entity, config);
```

Ejemplo de edit para relación de muchos a muchos, como es en una nueva página se instancia otro objeto.

```
/**
 * Elementos HTML necesarios para el manejo del carrito y la interfaz.
 */

const selectProducts = $("#id_product");
const tableProducts = $("#table-products");
const tableCartTemp = $("#table-products-temp");
const modal = $("#modal-products");

/**
 * Función para formatear las acciones relacionadas con los productos.
 * @param {Object} element - Elemento relacionado con el descuento.
 * @returns {string} - HTML de las acciones relacionadas con los productos.
 */
const formatActionsRelation = (element) => {
  return `
            <button class="btn btn-sm btn-info mx-1"><i data-id="${element.id}" class="fa fa-plus add-
cart"></i></button>
          `;
};

/**
 * Función para formatear las acciones de productos temporales en el carrito.
 * @param {Object} element - Elemento temporal en el carrito.
 * @returns {Object} - Acciones formateadas para el producto temporal.
 * @property {string} acciones - HTML de las acciones para eliminar el producto del carrito.
 * @property {string} amount_input - HTML del campo de entrada de cantidad para el producto.
 */
const formatActionsTemp = (element) => {
  const actions = {
    acciones: `<button data-id="${element.id}" class="btn btn-sm btn-danger mx-1 delete-cart"><i class="fa fa-
trash delete-cart"  data-id="${element.id}"></i></button>`,
    amount_input: `<input data-id="${element.id}" type="number" value="${
      element.amount ?? 1
    }" class="form-control input-amount">`,
  };
  return actions;
};

/**
 * Configuración para el manejo del CRUD de descuentos y productos en el carrito.
 * @typedef {Object} CrudConfig
 * @property {JQuery} loader - Objeto jQuery que representa el loader principal.
 * @property {JQuery} loaderSecondary - Objeto jQuery que representa el loader secundario.
 * @property {JQuery} tableCartRelation - Objeto jQuery que representa la tabla de productos relacionada.
 * @property {JQuery} tableCartTemp - Objeto jQuery que representa la tabla temporal de productos.
 * @property {JQuery} modalTemp - Objeto jQuery que representa el modal de productos temporales.
 * @property {boolean} newPageEdit - Indica si la edición de elementos se realiza en una nueva página.
 * @property {string} endPointHandler - Ruta del recurso que se manejará (por ejemplo, "catalogs").
 * @property {string} showModalTempBtn - Selector del botón para mostrar el modal de productos temporales.
 * @property {string} addProductBtn - Selector del botón para agregar productos al carrito.
 * @property {string} updateTempBtn - Selector del botón para actualizar productos temporales.
 * @property {string} addCartBtn - Selector de los botones para agregar productos al carrito.
 * @property {string} deleteCartBtn - Selector de los botones para eliminar productos del carrito.
 * @property {string[]} selectors - Lista de campos para la manipulación del CRUD.
 * @property {string[]} selectorsNotNull - Lista de campos que no pueden ser nulos.
 * @property {boolean} hasMany - Indica si hay una relación de uno a muchos.
 * @property {string} cartRelation - Nombre de la relación de carrito en la entidad principal.
 * @property {string} nameTypeTable - Tipo de tabla (por ejemplo, "secondary").
 * @property {Object[]} tableRelation - Configuración para las relaciones de la tabla principal.
 * @property {string} tableRelation[].entityRest - Nombre del recurso relacionado.
 * @property {string} tableRelation[].nameSecondary - Nombre del recurso secundario relacionado.
 * @property {string} tableRelation[].select - Selector del elemento de interfaz para seleccionar la relación.
 * @property {string} tableRelation[].manyName - Nombre de la relación de uno a muchos.
 * @property {Function} formatActionsRelation - Función para formatear las acciones de la relación.
 * @property {Function} formatActionsTemp - Función para formatear las acciones de productos temporales en el
carrito.
 */
```

```
// Configuración para el manejo del crud
const config = {
  loader: $(".page-loader-wrapper"),
  loaderSecondary: $("#loader-container"),
  tableCartRelation: tableProducts,
  tableCartTemp: tableCartTemp,
  modalTemp: modal,
  newPageEdit: true,
  endPointHandler: "discounts",
  showModalTempBtn: "#btn-modal-products",
  addProductBtn: "#btn-add-products",
  updateTempBtn: "#btn-update",
  addCartBtn: ".add-cart",
  deleteCartBtn: ".delete-cart",
  selectors: [
    "id",
    "percentage",
    "by_quantity",
    "promotion_code",
    "start_date",
    "end_date",
  ],
  selectorsNotNull: ["percentage"],
  hasMany: true,
  cartRelation: "products",
  nameTypeTable: "secondary",
  tableRelation: [
    {
      entityRest: "products",
      nameSecondary: "products",
      select: "#id_product",
      manyName: "product",
    },
  ],
  formatActionsRelation: formatActionsRelation,
  formatActionsTemp: formatActionsTemp,
};

/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Instancia de la clase ApiClient para interactuar con la API.
 * @param {string} resourceName - Nombre del recurso que se manejará (por ejemplo, "discounts").
 * @param {CrudConfig} config - Configuración para el manejo del CRUD de descuentos.
 */


const apiClient = new ApiClient(urlRest);
const entity = "catalogs";
const crudHandler = new CrudHandler(
  apiClient,
  "catalogs/show/" + discountId,
  config
);
```

Código de la clase crud handler

```
/**
 * Constructor de la clase CrudHandler.
 * @param {ApiClient} apiClient - Cliente para realizar solicitudes HTTP.
 * @param {string} entity - Nombre de la entidad con la que se trabajará (por ejemplo, 'usuarios').
 * @param {object} config - Configuración de la clase, incluyendo elementos de la interfaz de usuario.
 */

class CrudHandler {
  constructor(apiClient, entity, config) {
    this.apiClient = apiClient;
    this.entity = entity;
    this.config = config;
    this.dataClient = {};
    this.dataIndex = [];
    this.dataDelete = [];
    this.dataSecondary = {};
    this.dataCart = [];

    this.init();
  }

  /*
   * Inicializa la configuración y carga inicial de datos.
   */
  async init() {
    try {
      this.initTable();
      this.initEventHandlers();
      await this.loadDataIndex();
      await this.loadEntitySecondaries();
      if (this.config.tableDeletes != undefined) {
        this.loadDataDelete();
      }
      if (this.config.newPageEdit) {
        this.loadTableSecondaryEdit();
        this.loadInputsEdit();
      }
      if (this.config.labelEdit == undefined) {
        this.config.labelEdit = "-edit";
      }
      if (this.config.newPageEdit) {
        this.previewImage(this.dataIndex);
      }
    } catch (error) {
      console.log("Error in init", error);
    }
  }

  /*
   * Inicializa la tabla principal y la tabla de elementos eliminados.
   */
  initTable() {
    try {
      this.config.table.bootstrapTable();
      this.config.tableDeletes.bootstrapTable();
    } catch (error) {}
  }

  /*
   * Muestra el modal en la interfaz de usuario.
   */
  showModal() {
    this.config.modal.modal("show");
  }

  /*
   * Oculta el modal en la interfaz de usuario.
   */
  hideModal() {
    this.config.modal.modal("hide");
  }

  /*
   * Inicializa los manejadores de eventos para los botones y elementos de la interfaz de usuario.
   */
  initEventHandlers() {
    $(this.config.saveBtn).on("click", () => {
      this.postInsertEnd();
    });

    $(this.config.updateBtn).on("click", () => {
```

```
    if (this.validateInsert(this.config.labelEdit)) {
      this.postInsertEndEdit();
    }
  });

  //v2
  $(this.config.showModalTempBtn).on("click", () => {
    this.config.modalTemp.modal("show");
  });

  $(this.config.addProductBtn).on("click", () => {
    this.addProductTemp();
  });

  $(this.config.saveTempBtn).on("click", () => {
    this.saveCart();
  });

  $(this.config.updateTempBtn).on("click", () => {
    this.updateCart();
  });

  this.eventDocumentReady(this.config.showModalEditBtn, "loadDataEdit");
  this.eventDocumentReady(this.config.showDeleteAlertBtn, "deleteClient");
  this.eventDocumentReady(this.config.showRestoreAlertBtn, "restoreClient");
  this.eventDocumentReady(this.config.removeIdsBtn, "deleteClientIds");
  this.eventDocumentReady(this.config.restoreIdsBtn, "restoreClientIds");

  //v2
  this.eventDocumentReady(this.config.addCartBtn, "addProductTemp");
  this.eventDocumentReady(this.config.deleteCartBtn, "deleteCart");
  this.eventDocumentReady(this.config.inputAmountBtn, "updateAmount");
}

eventDocumentReady(button, method) {
  if (button != undefined) {
    $(document).on("click", button, (event) => this[method](event));
  }
}

/*
 * Carga los datos iniciales de la tabla principal.
 */
async loadDataIndex() {
  if (this.entity == null) {
    return;
  }
  try {
    const response = await this.apiClient.get(this.entity);
    this.dataIndex = response.data;
    if (this.config.table != undefined) {
      this.loadTableIndexInit();
    }
  } catch (error) {
    this.handleError(error);
  }
}

/*
 * Carga los datos de la tabla de elementos eliminados.
 */
async loadDataDelete() {
  try {
    const response = await this.apiClient.get(`${this.entity}/deletes`);
    this.dataDelete = response.data;
    this.loadTableDeleteInit();
  } catch (error) {
    this.handleError(error);
  }
}

postInsertEnd() {
  if (this.validateInsert()) {
    if (this.config.contentImage) {
      this.postClientInsert("assembleObjectClientForm", false);
    } else {
      this.postClientInsert();
    }
  }
}

postInsertEndEdit() {
  if (this.validateInsert(this.config.labelEdit)) {
    if (this.config.contentImage) {
      this.postClientUpdate(
```

```
        "assembleObjectClientForm",
        this.config.labelEdit,
        false
      );
    } else {
      this.postClientUpdate();
    }
  }
}

async postClientInsert(
  methodAssemble = "assembleObjectClient",
  isJson = true
) {
  try {
    this.config.loader.fadeIn();

    const endpoint =
      this.config.endPointHandler != undefined
        ? this.config.endPointHandler
        : this.entity;
    const dataClientInsert =
      this.config.nameTypeTable == "independent" ||
      this.config.nameTypeTable == "secondary"
        ? this[methodAssemble]()
        : this.dataClient;
    const response = await this.apiClient.post(
      `${endpoint}/store`,
      dataClientInsert,
      isJson
    );

    this.config.loader.fadeOut();

    const dataResponse = await response.json();
    this.handleSuccessfulResponseInsertEnd(
      dataResponse.status,
      dataResponse,
      dataClientInsert
    );
  } catch (error) {
    this.handleError(error);
  }
}

handleSuccessfulResponseInsertEnd(
  httpResponse,
  dataResponse,
  dataClientInsert,
  response = null
) {
  const dataInsert =
        this.config.contentImage == true
          ? dataResponse.data.inserted_data  //mejorar esta parte de inserted_data
          : dataClientInsert;
  switch (httpResponse) {
    case 201:
      this.handleSuccessfulResponseInsert(response, dataInsert);
      break;
    case 500:
      this.showAlert("error", "Error al insertar el registro.", 1500);
      break;
    case 400:
        this.showAlert("error", "El usuario ya existe.", 1500);
        break;
    default:
      this.handleSuccessfulResponseInsert(response, dataInsert);
      break;
  }
}

async postClientUpdate(
  methodAssemble = "assembleObjectClientEdit",
  label = "",
  isJson = true
) {
  try {
    this.config.loader.fadeIn();
    const endpoint =
      this.config.endPointHandler != undefined
        ? this.config.endPointHandler
        : this.entity;
    const dataClientInsert =
      this.config.nameTypeTable == "independent" ||
      this.config.nameTypeTable == "secondary"
```

```
        ? this[methodAssemble](label)
        : this.dataClient;
    const response = await this.apiClient.post(
      `${endpoint}/update`,
      dataClientInsert,
      isJson
    );
    this.config.loader.fadeOut();

    const dataResponse = await response.json();
    const dataUpdateResponse = dataResponse.data;
    console.log(dataResponse);
    if (dataResponse.status != 200) {
      this.showAlert("error", "Error al actualizar el registro.", 1500);
      return;
    }

    this.actionsAfterUpdate(dataUpdateResponse);
  } catch (error) {
    this.handleError(error);
  }
}

async postClientDestroyIds(event = null) {
  try {
    this.config.loader.fadeIn();
    const dataClientDestroy = {
      data: this.getSelectedRows(this.config.table),
    };
    const response = await this.apiClient.post(
      `${this.entity}/destroy/ids`,
      dataClientDestroy
    );
    this.config.loader.fadeOut();

    if (response.status !== 200) {
      this.showAlert("error", "Error al eliminar los registros.", 1500);
      return;
    }

    this.actionsAfterDeleteIds(dataClientDestroy.data);
  } catch (error) {
    this.handleError(error);
  }
}

async postClientRestoreIds(event = null) {
  try {
    this.config.loader.fadeIn();
    const dataClientRestore = {
      data: this.getSelectedRows(this.config.tableDeletes),
    };
    const response = await this.apiClient.post(
      `${this.entity}/restore/ids`,
      dataClientRestore
    );
    this.config.loader.fadeOut();

    if (response.status !== 200) {
      this.showAlert("error", "Error al restaurar el registro.", 1500);
      return;
    }

    this.actionsAfterRestoreIds(dataClientRestore.data);
  } catch (error) {
    this.handleError(error);
  }
}

async getDeleteClient(event) {
  try {
    this.config.loader.fadeIn();
    const id = $(event.target).data("id");
    const response = await this.apiClient.get(`${this.entity}/destroy/${id}`);
    this.config.loader.fadeOut();
    if (response.status == 200) {
      this.handleSuccessfulDelete(response, id);
    } else {
      this.showAlert("error", "Error al eliminar el registro.", 1000);
    }
  } catch (error) {
    this.handleError(error);
  }
}
```

```
async getRestoreClient(event) {
  try {
    this.config.loader.fadeIn();
    const id = $(event.target).data("id");
    const response = await this.apiClient.get(`${this.entity}/restore/${id}`);
    this.config.loader.fadeOut();
    if (response.status == 200) {
      this.handleSuccessfulRestore(response, id);
    } else {
      this.showAlert("error", "Error al restaurar el registro.", 1000);
    }
  } catch (error) {
    this.handleError(error);
  }
}

deleteClient(event) {
  this.showAlertDialog(
    event,
    "getDeleteClient",
    "¿Seguro de eliminar el registro?",
    "Eliminar"
  );
}

restoreClient(event) {
  this.showAlertDialog(
    event,
    "getRestoreClient",
    "¿Seguro de restaurar el registro?",
    "Restaurar"
  );
}

deleteClientIds(event = null) {
  this.showAlertDialog(
    event,
    "postClientDestroyIds",
    "¿Seguro de eliminar los registros seleccionados?",
    "Eliminar"
  );
}

restoreClientIds(event = null) {
  this.showAlertDialog(
    event,
    "postClientRestoreIds",
    "¿Seguro de restaurar los registros seleccionados?",
    "Restaurar"
  );
}

actionsAfterUpdate(dataClient) {
  if (this.config.newPageEdit) {
    const labelHref = document.createElement("a");
    let endpoint = this.config.endPointHandler;
    if (endpoint == undefined) {
      endpoint = this.entity;
    }
    this.showAlert("success", "Registro actualizado correctamente.", 1500);
    setTimeout(() => {
      labelHref.href = urlLocal + "/" + endpoint;
      labelHref.click();
    }, 1000);
  } else {
    this.updateObjectEdit(dataClient);
    this.hideModal();
    this.cleanInput();
    this.showAlert("success", "Registro actualizado correctamente.", 1500);
    this.loadTableIndexInit();
  }
}

actionsAfterDeleteIds(dataClient) {
  const arrayIds = dataClient.map((objeto) => objeto.id);
  const newDataDelete = this.dataIndex.filter((element) =>
    arrayIds.includes(element.id)
  );
  this.dataDelete = [...this.dataDelete, ...newDataDelete];
  this.dataIndex = this.dataIndex.filter(
    (element) => !arrayIds.includes(element.id)
  );
  this.loadTableIndexInit();
  this.loadTableDeleteInit();
  this.showAlert("success", "Registros eliminados correctamente.", 1500);
```

```
}

actionsAfterRestoreIds(dataClient) {
  const arrayIds = dataClient.map((objeto) => objeto.id);
  const newDataIndex = this.dataDelete.filter((element) =>
    arrayIds.includes(element.id)
  );
  this.dataIndex = [...this.dataIndex, ...newDataIndex];
  this.dataDelete = this.dataDelete.filter(
    (element) => !arrayIds.includes(element.id)
  );
  this.loadTableIndexInit();
  this.loadTableDeleteInit();
  this.showAlert("success", "Registros restaurados correctamente.", 1500);
}

updateObjectEdit(dataClient) {
  let dataEdit = this.dataIndex.find(
    (element) => element.id == dataClient.id
  );
  for (let property in dataEdit) {
    dataEdit[property] = dataClient[property];
  }
  console.log(dataEdit);
  console.log(this.dataIndex);
}

loadDataEdit(event) {
  const id = $(event.target).data("id");
  const dataEdit = this.dataIndex.find((element) => element.id == id);
  const selectors = this.config.selectors;
  if (dataEdit != undefined) {
    selectors.forEach((element) => {
      $(`#${element + this.config.labelEdit}`).val(dataEdit[element]);
    });
    this.loadSelectEdit(dataEdit);
    this.previewImage(dataEdit);
    this.showModal();
  }
}

assembleObjectClient() {
  const selectors = this.config.selectors;
  const objectClient = {};
  selectors.forEach((element) => {
    const input = $(`#${element}`);
    if (input.is(":checkbox")) {
      objectClient[element] = input.prop('checked') ? 1 : 0;
    } else {
      objectClient[element] = input.val();
    }

  });
  if (!this.config.newPageEdit) {
    objectClient.id = this.getNextId();
  }
  return objectClient;
}

assembleObjectClientForm(label = "") {
  const selectors = this.config.selectors;
  const formData = new FormData();
  const inputImages = this.config.filesInputImage ?? [];
  inputImages.forEach((element) => {
    const fileInput = document.getElementById(`${element}${label}`);
    formData.append(element, fileInput.files[0]);
  });

  selectors.forEach((element) => {
    formData.append(element, $(`#${element}${label}`).val());
  });

  return formData;
}

assembleObjectClientEdit() {
  const selectors = this.config.selectors;
  const objectClient = {};
  selectors.forEach((element) => {
    // objectClient[element] = encodeURIComponent($(`#${element + this.config.labelEdit}`).val());
    objectClient[element] = $(`#${element + this.config.labelEdit}`).val();
  });
  return objectClient;
}
```

```
  getNextId() {
    return this.dataIndex.length > 0
      ? Math.max(...this.dataIndex.map((item) => item.id)) + 1
      : 1;
  }

  loadTableIndex() {
    this.dataIndex.forEach((element) => {
      element.acciones = this.formatActions(element);
    });

    this.dataIndex.sort((elementA, elementB) => elementB.id - elementA.id);
    this.config.table.bootstrapTable("load", this.dataIndex);
  }

  loadTableDelete() {
    this.dataDelete.forEach((element) => {
      element.acciones = this.formatActionsDelete(element);
    });

    this.dataDelete.sort((elementA, elementB) => elementB.id - elementA.id);
    this.config.tableDeletes.bootstrapTable("load", this.dataDelete);
  }

  /**
   * Carga datos secundarios en una tabla, realiza operaciones de formato y manejo de relaciones.
   *
   * @param {Array} data - Los datos a cargar en la tabla secundaria.
   * @param {jQuery} [table] - La tabla Bootstrap a la que se cargarán los datos (opcional).
   * @param {string} [functionActions="formatActions"] - El nombre de la función que realiza acciones
específicas en los elementos (opcional).
   * @param {Array} [tableRelation=this.config.tableRelation] - Configuración de las relaciones entre las
tablas (opcional).
   * @throws {Error} Lanza un error si ocurre algún problema durante el proceso.
   */
  loadTableSecondary(
    data,
    table = undefined,
    functionActions = "formatActions",
    tableRelation = this.config.tableRelation
  ) {
    try {
      /**
       * Procesa las relaciones para un elemento dado.
       *
       * @param {Object} element - El elemento actual que contiene datos y relaciones.
       * @param {Array} relations - Las relaciones que se deben procesar para el elemento.
       */
      const processRelations = (element, relations) => {
        relations.forEach((relation) => {
          const nameSecondary = relation.nameSecondary;
          const nameIndex = relation.nameIndex ?? [];
          const elementAux = element[nameSecondary];

          nameIndex.forEach((elementNameIndex) => {
            let prefix = relation.prefix ?? nameSecondary;
            prefix = prefix != "" ? prefix + "_" : prefix;

            let substring = "image";
            if (elementNameIndex.includes(substring)) {
              prefix += relation.prefixImage + "_";
            }

            if (elementAux != undefined) {
              element[`${prefix}${elementNameIndex}`] =
                elementAux[elementNameIndex];
            }
          });

          if (relation.with && relation.with.length > 0) {
            processRelations(elementAux, relation.with);
          }

          element.acciones = this[functionActions](element);
          this.loadImageTableIndex(element);

          // Copia propiedades del with al elemento principal
          if (relation.with && relation.with.length > 0) {
            relation.with.forEach((withAttribute) => {
              const attributeName = withAttribute.nameSecondary;
              const nameIndexWith = withAttribute.nameIndex;
              element[attributeName] = elementAux[attributeName];

              nameIndexWith.forEach((nameIndexElement) => {
                let prefixAfter = relation.prefix ?? attributeName;
```

```
              prefixAfter =
                prefixAfter != "" ? prefixAfter + "_" : prefixAfter;
              element[prefixAfter + nameIndexElement] =
                elementAux[prefixAfter + nameIndexElement];
            });
          });
        }
      });
    };

    const relationsAttributes = tableRelation;
    data.forEach((element) => {
      processRelations(element, relationsAttributes);
    });

    // Ordena los datos en orden descendente por el campo 'id'
    data.sort((elementA, elementB) => elementB.id - elementA.id);

    // Si se proporciona una tabla, carga los datos en ella utilizando BootstrapTable
    if (table != undefined) {
      table.bootstrapTable("load", data);
    }
  } catch (error) {
    console.error("Error en subtable", error);
    throw new Error(
      "Error durante la carga de datos secundarios en la tabla."
    );
  }
}

loadTableSecondaryEdit() {
  this.loadTableSecondary([this.dataIndex]);
}

loadTableIndexInit() {
  switch (this.config.nameTypeTable) {
    case "independent":
      this.loadTableIndex();
      break;
    case "secondary":
      this.loadTableSecondary(this.dataIndex, this.config.table);
      break;
    default:
      this.loadTableIndex();
      break;
  }
}

loadTableDeleteInit() {
  switch (this.config.nameTypeTable) {
    case "independent":
      this.loadTableDelete();
      break;
    case "secondary":
      this.loadTableSecondary(
        this.dataDelete,
        this.config.tableDeletes,
        "formatActionsDelete"
      );
      break;
    default:
      this.loadTableDelete();
      break;
  }
}

cleanInput() {
  const selectors = this.config.selectors;
  selectors.forEach((element) => {
    $(`#${element}`).val("");
  });
}

/**
 * Maneja la respuesta exitosa después de una operación de inserción.
 * @param {object} response - Respuesta de la solicitud HTTP.
 * @param {object} dataClient - Datos del cliente que se insertaron.
 */
handleSuccessfulResponse(response, dataClient) {
  this.dataIndex.unshift(dataClient);
  this.cleanInput();
  this.loadTableIndexInit();
  this.showAlert("success", "Registro insertado correctamente.", 1500);
}
```

```
/**
 * Maneja la respuesta exitosa después de una operación de inserción en la tabla secondaria.
 * @param {object} response - Respuesta de la solicitud HTTP.
 * @param {object} dataClient - Datos del cliente que se insertaron.
 */
handleSuccessfulResponseSecondary(response, dataClient = {}) {
  this.dataIndex.unshift(dataClient);
  this.cleanInput();
  this.loadTableIndexInit();
  this.showAlert("success", "Registro insertado correctamente.", 1500);

  this.reloadSelect();
}

handleSuccessfulResponseInsert(response, dataClient) {
  switch (this.config.nameTypeTable) {
    case "independent":
      this.handleSuccessfulResponse(response, dataClient);
      break;
    case "secondary":
      this.handleSuccessfulResponseSecondary(response, dataClient);
      break;

    default:
      this.handleSuccessfulResponse(response, dataClient);
      break;
  }
  this.reloadImageAfterInsert();
}
/**
 * Maneja la respuesta exitosa después de una operación de eliminación.
 * @param {object} response - Respuesta de la solicitud HTTP.
 * @param {number} id - Identificador del registro eliminado.
 */
handleSuccessfulDelete(response, id) {
  this.showAlert("success", "Registro eliminado correctamente.", 1000);
  const data = response.data;
  const dataInsertDelete = this.dataIndex.find(
    (element) => element.id == data.id
  );
  this.dataIndex = this.dataIndex.filter((element) => element.id != data.id);
  this.dataDelete.unshift(dataInsertDelete);
  this.loadTableIndexInit();
  this.loadTableDeleteInit();
}

handleSuccessfulRestore(response, id) {
  this.showAlert("success", "Registro restaurado correctamente.", 1000);
  const data = response.data;
  const dataInsertRestore = this.dataDelete.find(
    (element) => element.id == data.id
  );
  this.dataDelete = this.dataDelete.filter(
    (element) => element.id != data.id
  );
  this.dataIndex.unshift(dataInsertRestore);

  this.loadTableIndexInit();
  this.loadTableDeleteInit();
}

handleError(error) {
  console.error(error);

  let errorMessage = "Error en la solicitud, por favor verificar los datos.";
  if (error.name === "NetworkError") {
    errorMessage =
      "Error de red. Por favor, comprueba tu conexión a internet.";
  } else if (error.name === "RequestAbortedError") {
    errorMessage = "La solicitud fue cancelada.";
  }

  this.showAlert("error", errorMessage, 2000);
  this.config.loader.fadeOut();
}

/**
 * Muestra una alerta utilizando la biblioteca Swal.
 * @param {string} icon - Icono de la alerta (por ejemplo, 'success', 'error').
 * @param {string} title - Título de la alerta.
 * @param {number} time - Tiempo de visualización de la alerta en milisegundos.
 */
showAlert(icon, title, time, position = "center") {
  Swal.fire({
    position: position,
```

```
    icon: icon,
    title: title,
    showConfirmButton: false,
    timer: time,
  });
}

/**
 * Muestra un diálogo de confirmación utilizando la biblioteca Swal.
 * @param {object} event - Evento que desencadenó la acción.
 * @param {string} callback - Nombre del método a llamar si se confirma la acción.
 * @param {string} title - Título del diálogo.
 * @param {string} buttonText - Texto del botón de confirmación.
 */
showAlertDialog(event, callback, title, buttonText) {
  Swal.fire({
    title: title,
    showDenyButton: true,
    showCancelButton: true,
    showConfirmButton: false,
    denyButtonText: buttonText,
  }).then((result) => {
    if (result.isDenied) {
      this[callback](event);
    }
  });
}

/**
 * Valida la inserción de datos antes de realizar la operación.
 * @param {string} labels - Etiquetas adicionales para los selectores (por ejemplo, '-edit').
 * @returns {boolean} - Indica si la validación fue exitosa.
 */
validateInsert(labels = "") {
  const selectors = this.config.selectorsNotNull;
  for (const element of selectors) {
    const input = $(`#${element}${labels}`);
    if (!this.validation(input)) {
      return false;
    }
  }
  return true;
}

/**
 * Realiza la validación de un campo de entrada.
 * @param {object} input - Elemento de entrada jQuery.
 * @returns {boolean} - Indica si la validación fue exitosa.
 */
validation(input) {
  let resultado = true;
  if (input.val() == "") {
    input.addClass("invalid");
    resultado = false;
  } else {
    input.removeClass("invalid");
  }
  return resultado;
}

/**
 * Escapa los valores de un objeto para evitar problemas de codificación en las solicitudes HTTP.
 * @param {object} obj - Objeto con valores a escapar.
 * @returns {object} - Objeto con valores escapados.
 */
escapeObjectValues(obj) {
  const escapeObject = {};
  for (const propery in obj) {
    if (obj.hasOwnProperty(propery)) {
      escapeObject[propery] = encodeURIComponent(obj[propery]);
    }
  }
  return escapeObject;
}

/**
 * Obtiene las filas seleccionadas de una tabla dada.
 * @param {object} table - Elemento de tabla BootstrapTable.
 * @returns {array} - Arreglo de filas seleccionadas.
 */
getSelectedRows(table) {
  const seletedRows = table.bootstrapTable("getSelections");
  return seletedRows;
}
```

```
/*
 * Carga las tablas secundarias para la relación uno a muchos
 */
async loadEntitySecondaries() {
  if (this.config.nameTypeTable == "independent") {
    return;
  }
  try {
    const relations = this.config.tableRelation;
    // this.config.loader.fadeIn();
    relations.forEach((element) => {
      const entity =
        element.entityRest != undefined
          ? element.entityRest
          : element.nameSecondary;
      this.apiClient
        .get(`${entity}`)
        .then((response) => {
          const foreignKey = element.foreignKey;
          const idForeign = this.dataIndex[foreignKey];
          this.dataSecondary[element.nameSecondary] = response.data;
          this.loadSelect(element.select, response.data, idForeign);
          this.loadTableCart(response.data);
          this.config.loader.fadeOut();
        })
        .catch((error) => {
          this.config.loader.fadeOut();
          console.error(error);
        });
    });
  } catch (error) {
    this.config.loader.fadeOut();
    this.handleError(error);
  }
}

/**
 * Carga el select2 con los valores de la tabla secundaria.
 * @param {jQuery} select - Elemento de select2.
 * @param {array} array - Arreglo de datos para cargar en el select.
 */
loadSelect(selectId, array, id = -1) {
  const select = $(selectId);
  if (select.is("select") && this.config.nameTypeTable == "secondary") {
    select.empty();
    select.select2({ width: "100%" });
    const option = `<option value = "${0}">Seleccione una opción</option>`;
    select.append(option);

    array.forEach((element) => {
      let selected = "";
      if (element.id == id) {
        selected = "selected";
      }
      const option = `<option value = "${element.id}" ${selected}>${element.id} - ${element.name}</option>`;
      select.append(option);
    });
    select.trigger("change");
  }
}

loadSelectEdit(dataEdit = null) {
  if (this.config.nameTypeTable == "secondary") {
    const tableRelation = this.config.tableRelation;
    tableRelation.forEach((element) => {
      this.loadSelect(
        `${element.select + this.config.labelEdit}`,
        this.dataSecondary[element.nameSecondary],
        dataEdit[element.foreignKey]
      );
    });
  }
}

loadImageTableIndex(element) {
  if (this.config.contentImage) {
    const imagesInput = this.config.filesInputImage ?? [];
    let imageCount = 0;
    let content = `
    <div class="d-flex justify-content-center align-items-center" id="container-img">`;
    for (const nameInput of imagesInput) {
      content += `
              <figure class="mx-1" style="width: 100px; height: 80px;">
                  <img class="w-100 h-100" src="${urlLocal}/assets/images/${element[nameInput]}"
                       onerror="this.src='${urlLocal}/assets/images/does_not_exist.avif'" />
```

```
                </figure>
                `;

        imageCount++;
        if (imageCount == 3) {
          break;
        }
      }
      content += `</div>`;
      element.image_index = content;
    }
  }

  reloadImageAfterInsert(label = "") {
    if (this.config.contentImage) {
      const filesInput = this.config.filesInputImage ?? [];
      filesInput.forEach((element) => {
        const dropifyInstance = $(`#${element}${label}`).data("dropify");
        dropifyInstance.clearElement();
      });
    }
  }

  previewImage(data) {
    if (this.config.contentImage) {
      this.reloadImageAfterInsert(this.config.labelEdit);
      const filesInput = this.config.filesInputImage ?? [];
      filesInput.forEach((element) => {
        const dropifyInstance = $(`#${element + this.config.labelEdit}`).data(
          "dropify"
        );
        let imageUrl = data[element];
        imageUrl = `${urlLocal}/assets/images/${imageUrl}`;
        dropifyInstance.setPreview(true, imageUrl);
      });
    }
  }

  /**
   * Recarga los valores del select2 después de una operación.
   */
  reloadSelect() {
    const relations = this.config.tableRelation;
    relations.forEach((element) => {
      const relationArrayData = this.dataSecondary[element.nameSecondary];
      this.loadSelect(element.select, relationArrayData);
    });
  }

  loadInputsEdit() {
    const selectors = this.config.selectors;
    const data = this.dataIndex ?? [];
    selectors.forEach((element) => {
      $(`#${element}`).val(data[element]);
    });
    if (this.config.hasMany == undefined) {
      return;
    }
    if (!this.config.hasMany) {
      return;
    }
    const relation = this.config.tableRelation[0];
    const dataMany = this.dataIndex[relation.nameSecondary];
    this.dataCart = dataMany.map((element) => element[relation.manyName]);
  }

  formatActions = (element) => {
    return `
            <button type="button" data-id="${element.id}" class="btn btn-sm edit" title="Editar">
                    <i data-id="${element.id}" class="fa fa-edit"></i>
            </button>
            <button type="button" data-id="${element.id}" class="btn btn-sm delete" title="Borrar">
                <i data-id="${element.id}" class="fa fa-trash text-danger"></i>
            </button>
          `;
  };

  formatActionsDelete = (element) => {
    return `
          <button type="button" data-id="${element.id}" class="btn btn-sm text-info restore"
title="Restaurar">
                  <i data-id="${element.id}" class="fa fa-undo"></i>
          </button>
        `;
  };
```

```
detailFormatterOne(index, row) {
  const contentDetail = this.config.contentDetailFormatter;
  const nameAttributeDetail = contentDetail.nameAttributeDetail;
  const headers = contentDetail.configTableDetail.headers;
  const images = contentDetail.images;

  // Crear elementos HTML de manera programática
  const detailContainer = document.createElement("div");
  detailContainer.classList.add("table-detail");

  const heading = document.createElement("h5");
  heading.classList.add("detail-heading");
  heading.textContent =
    this.config.contentDetailFormatter.configTableDetail.headTitle;

  const table = document.createElement("table");
  table.classList.add("table", "table-bordered", "detail-table");

  const thead = document.createElement("thead");
  thead.classList.add("thead-light");

  const headerRow = document.createElement("tr");
  headers.forEach((element) => {
    const th = document.createElement("th");
    th.textContent = element;
    headerRow.appendChild(th);
  });

  thead.appendChild(headerRow);
  table.appendChild(thead);

  const tbody = document.createElement("tbody");
  const dataRow = document.createElement("tr");

  const valueRowDetail = row[nameAttributeDetail];
  contentDetail.attributes.forEach((element) => {
    const td = document.createElement("td");
    td.textContent = valueRowDetail[element];
    dataRow.appendChild(td);
  });

  const imgContainer = document.createElement("td");
  imgContainer.innerHTML = `
    <div class="d-flex justify-content-center align-items-center" id="container-img">
      ${images
        .map(
          (element) => `
        <figure class="mx-1" style="width: 100px; height: 80px;">
          <img class="w-100 h-100" src="${urlLocal}/assets/images/${valueRowDetail[element]}" />
        </figure>
      `
        )
        .join("")}
    </div>
  `;
  dataRow.appendChild(imgContainer);

  tbody.appendChild(dataRow);
  table.appendChild(tbody);

  // Agregar elementos al contenedor principal
  detailContainer.appendChild(heading);
  detailContainer.appendChild(table);

  return detailContainer;
}

//V2

loadTableCart(data) {
  if (this.config.tableCartRelation == undefined) {
    return;
  }
  this.loadFormatActionsCart(data);
  const table = this.config.tableCartRelation;
  table.bootstrapTable();
  table.bootstrapTable("load", data);
  this.loadTableTemp();
}

loadTableTemp(data = null) {
  if (this.config.tableCartTemp == undefined) {
    return;
  }
```

```
  this.loadFormatActionsCart(this.dataCart, "temporal");
  const table = this.config.tableCartTemp;
  table.bootstrapTable();
  table.bootstrapTable("load", this.dataCart);
}

loadFormatActionsCart(data, functionAction = "relation") {
  switch (functionAction) {
    case "relation":
      data.forEach((element) => {
        element.acciones = this.config.formatActionsRelation(element);
      });
      break;
    case "temporal":
      data.forEach((element) => {
        element.acciones = this.config.formatActionsTemp(element).acciones;
      });
      break;
    default:
      data.forEach((element) => {
        element.acciones = this.config.formatActionsTemp(element).acciones;
      });
      break;
  }
}

findCartItemById = (id) => {
  return this.dataCart.find((element) => element.id == id);
};

incrementCartItemAmount = (cartItem) => {
  cartItem.amount++;
  this.addFormatActionsTemp(this.dataCart);
};

addNewCartItem = (id) => {
  const dataNameRelation = this.config.tableRelation[0].nameSecondary;
  const data = this.dataSecondary[dataNameRelation];
  const dataAdd = data.find((element) => element.id == id);

  if (dataAdd) {
    dataAdd.amount = 1;
    this.dataCart.push(dataAdd);
    this.addFormatActionsTemp(this.dataCart);
  }
};

addProductTemp = (event = null) => {
  let id = 0;
  if (event == null) {
    const selectLabel = this.config.tableRelation[0].select;
    id = $(selectLabel).val();
  } else {
    const button = $(event.target);
    id = button.data("id");
  }
  if (id == 0) {
    this.showAlert("info", "Debe seleccionar un producto", 1000);
    return;
  }

  const existingCartItem = this.findCartItemById(id);

  if (existingCartItem) {
    this.incrementCartItemAmount(existingCartItem);
  } else {
    this.addNewCartItem(id);
  }

  this.config.modalTemp.modal("hide");
  this.showAlert("success", "Agregado correctamente", 400, "top-end");
};

validateAndShowAlert = (value) => {
  let result = true;
  if (value <= 0) {
    this.showAlert(
      "error",
      "No se permiten valores menores o iguales a 0",
      1500,
      "top-end"
    );
    result = false;
  }
  return result;
```

```
};

updateAmount(event) {
  const input = $(event.target);
  const id = input.data("id");
  const minValue = 1;
  let newAmount = parseInt(input.val());

  if (!this.validateAndShowAlert(newAmount)) {
    input.val(minValue);
    newAmount = minValue;
  }

  const dataAmount = this.dataCart.find((element) => element.id == id);
  dataAmount.amount = newAmount;
}

addFormatActionsTemp(data) {
  data.forEach((element) => {
    element.acciones = this.config.formatActionsTemp(element).acciones;
    element.amount_input =
      this.config.formatActionsTemp(element).amount_input;
  });
  this.config.tableCartTemp.bootstrapTable("load", data);
}

deleteCart(event) {
  const id = $(event.target).data("id");
  this.dataCart = this.dataCart.filter((element) => element.id != id);
  this.updateTempTable();
}

updateTempTable() {
  if (this.config.tableCartTemp) {
    this.config.tableCartTemp.bootstrapTable("load", this.dataCart);
  }
}

assembleObjectClientTemp() {
  const dataClient = this.assembleObjectClient();
  const cartRelation = this.config.cartRelation;
  dataClient[cartRelation] = this.dataCart;
  if (this.config.inputChecks != undefined) {
    const inputChecks = this.config.inputChecks;
    inputChecks.forEach((element) => {
      const name = element.attr("id");
      dataClient[name] = element.prop("checked") ? 1 : 0;
    });
  }
  return dataClient;
}

async saveCart() {
  if (!this.validateInsert()) {
    return;
  }
  try {
    const endpoint =
      this.config.endPointHandler != undefined
        ? this.config.endPointHandler
        : this.entity;

    const url = `${endpoint}/store`;
    const data = this.assembleObjectClientTemp();
    this.showLoader();
    const response = await this.apiClient.post(url, data);
    const dataResponse = await response.json();
    this.hideLoader();
    if (dataResponse.status == 201) {
      this.showAlert("success", "Registro insertado correctamente", 1000);
      setTimeout(() => {
        this.hrefPage(entity);
      }, 1100);
    } else {
      this.showAlert("error", "Error al insertar el registro", 1500);
    }
  } catch (error) {
    console.log(error);
  }
}

async updateCart() {
  if (!this.validateInsert()) {
    return;
  }
```

```
    try {
      const endpoint =
        this.config.endPointHandler != undefined
          ? this.config.endPointHandler
          : this.entity;

      const url = `${endpoint}/update`;
      const data = this.assembleObjectClientTemp();
      this.showLoader();
      const response = await this.apiClient.post(url, data);
      const dataResponse = await response.json();
      this.hideLoader();
      if (dataResponse.status == 200) {
        this.showAlert("success", "Registro actualizado correctamente", 1000);
        setTimeout(() => {
          this.hrefPage(entity);
        }, 1100);
      } else {
        this.showAlert("error", "Error al actualizar el registro", 1500);
      }
    } catch (error) {
      console.log(error);
    }
  }

  showLoader() {
    const loader = this.config.loaderSecondary;
    if (loader != undefined) {
      loader.show();
    }
  }

  hideLoader() {
    const loader = this.config.loaderSecondary;
    if (loader != undefined) {
      loader.hide();
    }
  }

  hrefPage(endpoint) {
    const newLink = document.createElement("a");
    newLink.href = `${urlLocal}/${endpoint}`;
    newLink.click();
  }
}
```

By Francisco Vaca