En products. Js Crea una nueva función llamada openSuperEditDialog que se encargará de abrir el diálogo de super edición del producto.

javascript

openSuperEditDialog: function(id) {

// Aquí debes agregar la lógica necesaria para mostrar el diálogo de super edición

// Puedes copiar y modificar el código de la función `openDialog` para lograr esto

this.description = "";

this.code\_bar = "";

this.selected = id;

this.code = "";

this.tax\_id = null;

this.brand\_id = 1;

this.presentation\_id = 19;

this.categories = [];

this.packing\_unit = 1;

// Agrega aquí la lógica adicional para la super edición, como campos extras, permisos, etc.

// Abre el diálogo de super edición

this.dialog = "super-edit";

},

Crea un nuevo diálogo para la super edición del producto. Puedes copiarlo del diálogo de edición normal y modificarlo según tus necesidades.

<dialog-basic :active="dialog == 'super-edit'">

<template slot="dialog-title">

<span>Super Edición del Producto</span>

</template>

<template slot="dialog-content">

<!-- Agrega aquí los campos y controles adicionales para la super edición -->

<v-row>

<v-col cols="6">

<v-text-field label="Campo adicional 1" v-model="field1"></v-text-field>

</v-col>

<v-col cols="6">

<v-text-field label="Campo adicional 2" v-model="field2"></v-text-field>

</v-col>

<!-- Agrega más campos según sea necesario -->

</v-row>

</template>

<template slot="dialog-actions">

<v-btn color="cyan dark-2" v-on:click="updateSuperProduct" outlined>

<span>Guardar</span>

<v-icon>mdi-content-save</v-icon>

</v-btn>

<v-btn outlined color="red" v-on:click="dialog = null">

<span>Cancelar</span>

<v-icon>mdi-close</v-icon>

</v-btn>

</template>

</dialog-basic>

Agrega una nueva función updateSuperProduct que se encargará de guardar los cambios realizados en la super edición del producto.

javascript

updateSuperProduct: function() {

// Aquí debes agregar la lógica necesaria para actualizar el producto con los nuevos campos de la super edición

fetchControllers("Products", "updateSuperProduct", {

params: {

id: this.selected,

attributes: {

description: this.description,

brand\_id: this.brand\_id,

presentation\_id: this.presentation\_id,

code\_bar: this.code\_bar,

code: this.code,

packing\_unit: this.packing\_unit,

tax\_id: this.tax\_id,

field1: this.field1,

field2: this.field2

// Agrega aquí los demás campos de la super edición

},

categories: this.categories

},

success: (response) => {

this.getIndex();

this.$gridbox[0].forceReload();

this.dialog = null;

},

failed: (response) => {

console.log(response.message);

},

});

},

Actualiza el botón de "Super Edición" en la plantilla del gridbox para que llame a la nueva función openSuperEditDialog.

<gridbox-app

url="Products.index"

:headers="headers"

:new="openDialog"

:edit="openDialog"

:show="openDialog"

:delete="openDialog"

:buttonsGrid="buttonsGrid"

>

</gridbox-app>

Modifica la definición del botón "Super Edición" en el arreglo buttonsGrid para que llame a la nueva función openSuperEditDialog.

javascript

buttonsGrid: [

{

text: "Categorias",

icon: "plus",

color: "pink",

action: this.openDialog,

id: "add-category"

},

{

text: "Super Edición",

icon: "pencil",

color: "yellow",

action: (id) => this.openSuperEditDialog(id),

id: "super-edicion"

}

],

En manageprices.js trae los  
import dialogEditPrices from "../Utils/dialogEditPrices.js";

import dialogMassiveUpdate from "../Utils/dialogMassiveUpdatePrices.js";  
  
Luego, Primero, vamos a agregar los elementos necesarios para la edición de precios en el componente "products-app":

javascript

data: function () {

return {

// ... datos existentes

showEditPriceDialog: false,

productPrice: null

};

},

methods: {

// ... métodos existentes

openEditPriceDialog() {

this.showEditPriceDialog = true;

},

closeEditPriceDialog() {

this.showEditPriceDialog = false;

},

saveProductPrice() {

// Aquí debes agregar la lógica para guardar el precio del producto

// Puedes utilizar el método `updateProduct` o crear uno nuevo

this.closeEditPriceDialog();

}

},

template: `

<!-- ... plantilla existente -->

<div>

<v-dialog v-model="showEditPriceDialog" max-width="600px">

<v-card>

<v-card-title>Editar precio del producto</v-card-title>

<v-card-text>

<v-text-field label="Precio" v-model="productPrice"></v-text-field>

</v-card-text>

<v-card-actions>

<v-btn color="blue darken-1" text @click="closeEditPriceDialog">

Cancelar

</v-btn>

<v-btn color="blue darken-1" text @click="saveProductPrice">

Guardar

</v-btn>

</v-card-actions>

</v-card>

</v-dialog>

</div>

`

Ahora, en el método openSuperEditDialog, vamos a agregar la lógica para abrir el diálogo de edición de precio:

javascript

openSuperEditDialog: function(id) {

this.description = "";

this.code\_bar = "";

this.selected = id;

this.code = "";

this.tax\_id = null;

this.brand\_id = 1;

this.presentation\_id = 19;

this.categories = [];

this.packing\_unit = 1;

// Agrega aquí la lógica adicional para la super edición, como campos extras, permisos, etc.

// Abre el diálogo de super edición

this.dialog = "super-edit";

this.openEditPriceDialog(); // Abre el diálogo de edición de precio

},

Finalmente, en el método updateSuperProduct, debes agregar la lógica para actualizar el precio del producto junto con los demás campos de la super edición:

javascript

updateSuperProduct: function() {

fetchControllers("Products", "updateSuperProduct", {

params: {

id: this.selected,

attributes: {

description: this.description,

brand\_id: this.brand\_id,

presentation\_id: this.presentation\_id,

code\_bar: this.code\_bar,

code: this.code,

packing\_unit: this.packing\_unit,

tax\_id: this.tax\_id,

price: this.productPrice // Aquí se debe utilizar el nuevo precio

// Agrega aquí los demás campos de la super edición

},

categories: this.categories

},

success: (response) => {

this.getIndex();

this.$gridbox[0].forceReload();

this.dialog = null;

},

failed: (response) => {

console.log(response.message);

},

});

},