DOCUMENTACIÓN DE MEJORAS PENDIENTES

Sistema de Gestión de Inventario

Este documento describe tres funcionalidades identificadas para mejora futura y explica técnicamente cómo se implementarían.

MEJORA 1: EDICIÓN DE CATEGORÍAS

Situación Actual

Actualmente, el módulo de categorías permite crear y eliminar categorías correctamente. Sin embargo, al hacer clic en el botón de editar (ícono del lápiz), el modal se abre pero no carga los datos existentes de la categoría seleccionada.

Causa Técnica

El problema se encuentra en el archivo JavaScript de categorías (categories.js). La función de edición necesita:

1. Recibir el ID de la categoría a editar

2. Hacer una petición GET al backend para obtener los datos actuales

3. Rellenar los campos del formulario con esos datos

4. Cambiar el modo del formulario de 'crear' a 'editar'

5. Al guardar, enviar PUT en lugar de POST

Solución Propuesta

Se debería modificar la función editCategory() en el archivo frontend/js/categories.js de la siguiente manera:

```
// Variable global para saber si estamos editando
let editingCategoryId = null;

// Función para editar categoría
async function editCategory(id) {
    try {
        // 1. Obtener datos actuales de la categoría
        const response = await fetch(
            `${API_CONFIG.BASE_URL}/usuario/${userId}/clasificaciones/${id}`,
            { headers: { 'x-access-token': Auth.getToken() } }
        );
        const data = await response.json();

        // 2. Rellenar el formulario con los datos
        document.getElementById('categoryName').value = data.name;
        document.getElementById('categoryDescription').value =
            data.descripcion || '';

        // 3. Guardar el ID que estamos editando
        editingCategoryId = id;

        // 4. Cambiar título del modal
        document.getElementById('modalTitle').textContent =
            'Editar Categoría';

        // 5. Mostrar el modal
        document.getElementById('categoryModal').style.display = 'flex';

    } catch (error) {
        console.error('Error:', error);
        alert('Error al cargar la categoría');
    }
}
```

Archivos Involucrados

• frontend/js/categories.js - Lógica de edición

- backend/api/routes/categories.py - Endpoint GET por ID

MEJORA 2: PERSISTENCIA DE DESCRIPCIÓN EN CATEGORÍAS

Situación Actual

Al crear o editar una categoría, el campo de descripción no se guarda correctamente en la base de datos. El campo aparece vacío aunque el usuario haya ingresado texto.

Causa Técnica

El problema puede estar en uno de estos puntos:

1. El frontend no está enviando el campo 'descripcion' en el JSON

2. El backend no está leyendo correctamente el campo del request

3. La consulta SQL INSERT/UPDATE no incluye el campo descripcion

Solución Propuesta

Paso 1: Verificar el Frontend (categories.js)

```
async function saveCategory() {
    const name = document.getElementById('categoryName').value.trim();
    const descripcion = document.getElementById('categoryDescription')
        .value.trim();

    if (!name) {
        alert('El nombre es requerido');
        return;
    }

    const body = JSON.stringify({
        name: name,
        descripcion: descripcion  // Importante: incluir este campo
    });

    const response = await fetch(url, {
        method: editingCategoryId ? 'PUT' : 'POST',
        headers: {
            'Content-Type': 'application/json',
            'x-access-token': Auth.getToken()
        },
        body: body
    });
}
```

Paso 2: Verificar el Backend (routes/categories.py)

```
@app.route('/usuario//clasificaciones', methods=['POST'])
def crear_categoria(user_id):
    data = request.get_json()
    name = data.get('name', '').strip()
    descripcion = data.get('descripcion', '').strip()

    cursor.execute(
        '''INSERT INTO categories (name, descripcion, user_id)
           VALUES (%s, %s, %s)''',
        (name, descripcion, user_id)
    )
```

Archivos Involucrados

- frontend/js/categories.js - Envío de datos
- backend/api/routes/categories.py - Recepción y guardado

MEJORA 3: VINCULAR PROVEEDOR CON PRODUCTO

Situación Actual

Actualmente el sistema permite gestionar proveedores y productos de forma independiente, pero no existe una funcionalidad para vincularlos entre sí. Esto significa que no podemos saber qué proveedor suministra cada producto.

Descripción de la Mejora

Se propone implementar una funcionalidad que permita:

1. Vincular uno o más proveedores a un producto específico

2. Consultar qué proveedores suministran un producto determinado

3. Ver qué productos ofrece cada proveedor

Implementación en Base de Datos

Ya existe la tabla intermedia suppliers_products en el schema de la base de datos que permite esta relación muchos-a-muchos:

```
CREATE TABLE suppliers_products (
    supplier_id INT NOT NULL,
    product_id INT NOT NULL,
    user_id INT NOT NULL,
    PRIMARY KEY (supplier_id, product_id),
    FOREIGN KEY (supplier_id) REFERENCES suppliers(id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Implementación en Backend

Se necesitarían crear los siguientes endpoints en el archivo routes/supplier.py:

```
# Vincular producto con proveedor
@supplier_bp.route('/usuario//proveedores//productos/',
                   methods=['POST'])
def vincular_producto_proveedor(user_id, supplier_id, product_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('''
        INSERT INTO suppliers_products (supplier_id, product_id, user_id)
        VALUES (%s, %s, %s)
    ''', (supplier_id, product_id, user_id))

    conn.commit()
    return jsonify({'message': 'Producto vinculado exitosamente'})


# Obtener proveedores de un producto
@supplier_bp.route('/usuario//articulos//proveedores',
                   methods=['GET'])
def obtener_proveedores_producto(user_id, product_id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    cursor.execute('''
        SELECT s.id, s.name_supplier, s.phone, s.mail
        FROM suppliers s
        JOIN suppliers_products sp ON s.id = sp.supplier_id
        WHERE sp.product_id = %s AND sp.user_id = %s
```

```
    ''', (product_id, user_id))

    proveedores = cursor.fetchall()
    return jsonify({'data': proveedores})
```

Implementación en Frontend

En la página de proveedores (suppliers.html), se agregarían dos secciones:

1. Formulario para vincular: Dos selectores (producto y proveedor) + botón 'Vincular'

2. Consulta de proveedores: Selector de producto + botón 'Consultar' que muestra tabla con proveedores

```
// Vincular producto con proveedor
async function vincularProductoProveedor() {
    const productId = document.getElementById('selectProducto').value;
    const supplierId = document.getElementById('selectProveedor').value;

    const response = await fetch(
        `${API_CONFIG.BASE_URL}/usuario/${userId}/proveedores/${supplierId}/prod
uctos/${productId}`,
        {
            method: 'POST',
            headers: { 'x-access-token': Auth.getToken() }
        }
    );

    if (response.ok) {
        UI.showSuccess('Producto vinculado exitosamente');
    }
}

// Consultar proveedores de un producto
async function consultarProveedores() {
    const productId = document.getElementById('selectProductoConsulta').value;

    const response = await fetch(
        `${API_CONFIG.BASE_URL}/usuario/${userId}/articulos/${productId}/proveed
ores`,
        { headers: { 'x-access-token': Auth.getToken() } }
    );

    const data = await response.json();
    mostrarTablaProveedores(data.data);
}
```

Archivos Involucrados

• backend/api/routes/supplier.py - Nuevos endpoints

• backend/api/models/supplier.py - Métodos del modelo

• frontend/js/suppliers.js - Lógica de vinculación

• frontend/pages/suppliers.html - Interfaz de usuario

• Base de datos: tabla suppliers_products (ya existe)

CONCLUSIÓN

Estas tres mejoras están identificadas y documentadas para una futura iteración del proyecto. No afectan el funcionamiento core del sistema, ya que:

• Las categorías se pueden crear y eliminar correctamente

• Los productos funcionan completamente (crear, editar, eliminar)

• El inventario se actualiza correctamente

• Las órdenes de compra funcionan y actualizan el stock

• Los proveedores se pueden gestionar (crear, eliminar, listar)

• Los reportes muestran datos precisos

Nota Final

La tabla suppliers_products ya existe en el schema de la base de datos, lo que demuestra que la arquitectura del sistema fue diseñada contemplando esta funcionalidad desde el inicio. La implementación quedó pendiente por cuestiones de tiempo, pero la infraestructura necesaria ya está preparada.