Based on your Vue.js shopping cart application, I need to create a MySQL table structure for storing shopping cart data that integrates with your existing product system.

Here's the recommended table structure for your shopping cart:

sql

CREATE TABLE shopping\_cart (

id\_cart INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

id\_product INT NOT NULL,

amount INT NOT NULL DEFAULT 1,

date\_added TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

date\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

status ENUM('active', 'purchased', 'removed') DEFAULT 'active',

FOREIGN KEY (user\_id) REFERENCES user(user\_id) ON DELETE CASCADE,

FOREIGN KEY (id\_product) REFERENCES product(id\_product) ON DELETE CASCADE,

UNIQUE KEY unique\_user\_product (user\_id, id\_product, status)

);

Esta tabla incluye:

* **id\_cart**: Identificador único del item en el carrito
* **user\_id**: Referencia al usuario (debe coincidir con tu tabla de usuarios)
* **id\_product**: Referencia al producto (coincide con tu código Vue)
* **amount**: Cantidad del producto (coincide con tu variable **quantity**)
* **date\_added/date\_updated**: Timestamps para auditoría
* **status**: Estado del item (activo, comprado, removido)

La clave única **unique\_user\_product** evita duplicados del mismo producto por usuario en estado activo.

También podrías necesitar una tabla adicional para órdenes de compra:

sql

CREATE TABLE orders (

id\_order INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

total\_amount DECIMAL(10,2) NOT NULL,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('pending', 'completed', 'cancelled') DEFAULT 'pending',

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

CREATE TABLE order\_items (

id\_order\_item INT AUTO\_INCREMENT PRIMARY KEY,

id\_order INT NOT NULL,

id\_product INT NOT NULL,

quantity INT NOT NULL,

unit\_price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (id\_order) REFERENCES orders(id\_order) ON DELETE CASCADE,

FOREIGN KEY (id\_product) REFERENCES products(id\_product)

);

Esta estructura te permitirá manejar tanto el carrito temporal como las órdenes finalizadas, manteniendo la integridad referencial con tus productos existentes.

Te ayudo a crear la estructura completa para manejar el carrito de compras. Basándome en tu código Vue.js, aquí tienes todos los componentes necesarios:

**1. Model (ShoppingCart.js)**

javascript

const db = require('../config/database');

class ShoppingCart {

// Agregar producto al carrito

static async addToCart(cartData) {

const { user\_id, id\_product, amount } = cartData;

try {

// Verificar si el producto ya existe en el carrito del usuario

const [existing] = await db.execute(

'SELECT \* FROM shopping\_cart WHERE user\_id = ? AND id\_product = ? AND status = "active"',

[user\_id, id\_product]

);

if (existing.length > 0) {

// Si existe, actualizar la cantidad

const [result] = await db.execute(

'UPDATE shopping\_cart SET amount = amount + ?, date\_updated = CURRENT\_TIMESTAMP WHERE id\_cart = ?',

[amount, existing[0].id\_cart]

);

return { success: true, message: 'Cantidad actualizada en el carrito' };

} else {

// Si no existe, crear nuevo registro

const [result] = await db.execute(

'INSERT INTO shopping\_cart (user\_id, id\_product, amount) VALUES (?, ?, ?)',

[user\_id, id\_product, amount]

);

return { success: true, message: 'Producto agregado al carrito', id: result.insertId };

}

} catch (error) {

throw error;

}

}

// Obtener carrito por usuario

static async getCartByUser(user\_id) {

try {

const [rows] = await db.execute(`

SELECT

sc.id\_cart,

sc.user\_id,

sc.id\_product,

sc.amount,

sc.date\_added,

p.name,

p.description,

p.price,

p.image,

(sc.amount \* p.price) as subtotal

FROM shopping\_cart sc

INNER JOIN products p ON sc.id\_product = p.id\_product

WHERE sc.user\_id = ? AND sc.status = "active"

ORDER BY sc.date\_added DESC

`, [user\_id]);

return rows;

} catch (error) {

throw error;

}

}

// Actualizar cantidad de producto en carrito

static async updateCartItem(id\_cart, amount) {

try {

const [result] = await db.execute(

'UPDATE shopping\_cart SET amount = ?, date\_updated = CURRENT\_TIMESTAMP WHERE id\_cart = ?',

[amount, id\_cart]

);

return result.affectedRows > 0;

} catch (error) {

throw error;

}

}

// Eliminar producto del carrito

static async removeFromCart(id\_cart) {

try {

const [result] = await db.execute(

'UPDATE shopping\_cart SET status = "removed" WHERE id\_cart = ?',

[id\_cart]

);

return result.affectedRows > 0;

} catch (error) {

throw error;

}

}

// Limpiar carrito (marcar como comprado)

static async clearCart(user\_id) {

try {

const [result] = await db.execute(

'UPDATE shopping\_cart SET status = "purchased" WHERE user\_id = ? AND status = "active"',

[user\_id]

);

return result.affectedRows > 0;

} catch (error) {

throw error;

}

}

// Obtener total del carrito

static async getCartTotal(user\_id) {

try {

const [rows] = await db.execute(`

SELECT

COUNT(\*) as total\_items,

SUM(sc.amount \* p.price) as total\_amount

FROM shopping\_cart sc

INNER JOIN products p ON sc.id\_product = p.id\_product

WHERE sc.user\_id = ? AND sc.status = "active"

`, [user\_id]);

return rows[0];

} catch (error) {

throw error;

}

}

}

module.exports = ShoppingCart;

**2. Controller (shoppingCartController.js)**

javascript

const ShoppingCart = require('../models/ShoppingCart');

const shoppingCartController = {

// Agregar producto al carrito

async addToCart(req, res) {

try {

const { user\_id, id\_product, amount } = req.body;

// Validaciones

if (!user\_id || !id\_product || !amount) {

return res.status(400).json({

success: false,

message: 'Todos los campos son requeridos: user\_id, id\_product, amount'

});

}

if (amount <= 0) {

return res.status(400).json({

success: false,

message: 'La cantidad debe ser mayor a 0'

});

}

const result = await ShoppingCart.addToCart({ user\_id, id\_product, amount });

res.status(201).json({

success: true,

message: result.message,

data: result

});

} catch (error) {

console.error('Error al agregar al carrito:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor',

error: error.message

});

}

},

// Obtener carrito del usuario

async getCart(req, res) {

try {

const { user\_id } = req.params;

if (!user\_id) {

return res.status(400).json({

success: false,

message: 'ID de usuario es requerido'

});

}

const cartItems = await ShoppingCart.getCartByUser(user\_id);

const cartTotal = await ShoppingCart.getCartTotal(user\_id);

res.status(200).json({

success: true,

data: {

items: cartItems,

summary: cartTotal

}

});

} catch (error) {

console.error('Error al obtener carrito:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor',

error: error.message

});

}

},

// Actualizar cantidad de producto

async updateCartItem(req, res) {

try {

const { id\_cart } = req.params;

const { amount } = req.body;

if (!amount || amount <= 0) {

return res.status(400).json({

success: false,

message: 'La cantidad debe ser mayor a 0'

});

}

const updated = await ShoppingCart.updateCartItem(id\_cart, amount);

if (!updated) {

return res.status(404).json({

success: false,

message: 'Item del carrito no encontrado'

});

}

res.status(200).json({

success: true,

message: 'Cantidad actualizada correctamente'

});

} catch (error) {

console.error('Error al actualizar item del carrito:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor',

error: error.message

});

}

},

// Eliminar producto del carrito

async removeFromCart(req, res) {

try {

const { id\_cart } = req.params;

const removed = await ShoppingCart.removeFromCart(id\_cart);

if (!removed) {

return res.status(404).json({

success: false,

message: 'Item del carrito no encontrado'

});

}

res.status(200).json({

success: true,

message: 'Producto eliminado del carrito'

});

} catch (error) {

console.error('Error al eliminar del carrito:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor',

error: error.message

});

}

},

// Limpiar carrito completo

async clearCart(req, res) {

try {

const { user\_id } = req.params;

const cleared = await ShoppingCart.clearCart(user\_id);

res.status(200).json({

success: true,

message: cleared ? 'Carrito limpiado correctamente' : 'El carrito ya estaba vacío'

});

} catch (error) {

console.error('Error al limpiar carrito:', error);

res.status(500).json({

success: false,

message: 'Error interno del servidor',

error: error.message

});

}

}

};

module.exports = shoppingCartController;

**3. Routes (shoppingCartRoutes.js)**

javascript

const express = require('express');

const router = express.Router();

const shoppingCartController = require('../controllers/shoppingCartController');

// Rutas del carrito de compras

router.post('/add', shoppingCartController.addToCart);

router.get('/user/:user\_id', shoppingCartController.getCart);

router.put('/item/:id\_cart', shoppingCartController.updateCartItem);

router.delete('/item/:id\_cart', shoppingCartController.removeFromCart);

router.delete('/clear/:user\_id', shoppingCartController.clearCart);

module.exports = router;

**4. API Service (shoppingCartApi.js) - Frontend**

javascript

const API\_BASE\_URL = 'http://localhost:5000/api';

export const addToCartAPI = async (cartData) => {

try {

const response = await fetch(`${API\_BASE\_URL}/cart/add`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(cartData)

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Error al agregar al carrito');

}

return data;

} catch (error) {

console.error('Error en addToCartAPI:', error);

throw error;

}

};

export const getCartAPI = async (user\_id) => {

try {

const response = await fetch(`${API\_BASE\_URL}/cart/user/${user\_id}`, {

method: 'GET',

headers: {

'Content-Type': 'application/json',

}

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Error al obtener carrito');

}

return data;

} catch (error) {

console.error('Error en getCartAPI:', error);

throw error;

}

};

export const updateCartItemAPI = async (id\_cart, amount) => {

try {

const response = await fetch(`${API\_BASE\_URL}/cart/item/${id\_cart}`, {

method: 'PUT',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ amount })

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Error al actualizar item');

}

return data;

} catch (error) {

console.error('Error en updateCartItemAPI:', error);

throw error;

}

};

export const removeFromCartAPI = async (id\_cart) => {

try {

const response = await fetch(`${API\_BASE\_URL}/cart/item/${id\_cart}`, {

method: 'DELETE',

headers: {

'Content-Type': 'application/json',

}

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Error al eliminar del carrito');

}

return data;

} catch (error) {

console.error('Error en removeFromCartAPI:', error);

throw error;

}

};

export const clearCartAPI = async (user\_id) => {

try {

const response = await fetch(`${API\_BASE\_URL}/cart/clear/${user\_id}`, {

method: 'DELETE',

headers: {

'Content-Type': 'application/json',

}

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Error al limpiar carrito');

}

return data;

} catch (error) {

console.error('Error en clearCartAPI:', error);

throw error;

}

};

**5. Configuración en tu archivo principal (app.js o server.js)**

javascript

// Importar las rutas del carrito

const shoppingCartRoutes = require('./routes/shoppingCartRoutes');

// Usar las rutas

app.use('/api/cart', shoppingCartRoutes);

Esta estructura te permitirá manejar completamente el carrito de compras desde tu aplicación Vue.js, con todas las operaciones CRUD necesarias para agregar, obtener, actualizar y eliminar productos del carrito.