**Inventory Management System**

**Features:**

* **Track Stock Levels, Orders, and Suppliers**
* **CRUD Operations** (Create, Read, Update, Delete)
* **User Authentication & Role-Based Access Control**

**Code Overview:**

**1.1 Product and Supplier Struct**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

struct Product {

int id;

string name;

int quantity;

double price;

};

struct Supplier {

int id;

string name;

string contactInfo;

};

**1.2 Inventory Management Class**

class InventoryManager {

private:

vector<Product> products;

vector<Supplier> suppliers;

int productIdCounter = 0;

int supplierIdCounter = 0;

public:

// CRUD for Products

void createProduct(string name, int quantity, double price) {

Product newProduct = { productIdCounter++, name, quantity, price };

products.push\_back(newProduct);

cout << "Product added successfully!" << endl;

}

void updateProduct(int id, string newName, int newQuantity, double newPrice) {

for (Product& product : products) {

if (product.id == id) {

product.name = newName;

product.quantity = newQuantity;

product.price = newPrice;

cout << "Product updated successfully!" << endl;

return;

}

}

cout << "Product not found!" << endl;

}

void deleteProduct(int id) {

for (auto it = products.begin(); it != products.end(); ++it) {

if (it->id == id) {

products.erase(it);

cout << "Product deleted successfully!" << endl;

return;

}

}

cout << "Product not found!" << endl;

}

void displayProducts() {

for (const Product& product : products) {

cout << "ID: " << product.id << " | Name: " << product.name << " | Quantity: " << product.quantity << " | Price: " << product.price << endl;

}

}

// CRUD for Suppliers

void createSupplier(string name, string contactInfo) {

Supplier newSupplier = { supplierIdCounter++, name, contactInfo };

suppliers.push\_back(newSupplier);

cout << "Supplier added successfully!" << endl;

}

void updateSupplier(int id, string newName, string newContactInfo) {

for (Supplier& supplier : suppliers) {

if (supplier.id == id) {

supplier.name = newName;

supplier.contactInfo = newContactInfo;

cout << "Supplier updated successfully!" << endl;

return;

}

}

cout << "Supplier not found!" << endl;

}

void deleteSupplier(int id) {

for (auto it = suppliers.begin(); it != suppliers.end(); ++it) {

if (it->id == id) {

suppliers.erase(it);

cout << "Supplier deleted successfully!" << endl;

return;

}

}

cout << "Supplier not found!" << endl;

}

void displaySuppliers() {

for (const Supplier& supplier : suppliers) {

cout << "ID: " << supplier.id << " | Name: " << supplier.name << " | Contact Info: " << supplier.contactInfo << endl;

}

}

};

**1.3 User Roles & Authentication**

enum Role { ADMIN, MANAGER };

class RoleBasedUser {

public:

string username;

string password;

Role role;

bool isAuthenticated;

RoleBasedUser(string u, string p, Role r) : username(u), password(p), role(r), isAuthenticated(false) {}

bool login(string u, string p) {

if (username == u && password == p) {

isAuthenticated = true;

return true;

}

return false;

}

void logout() {

isAuthenticated = false;

}

};

**1.4 Main Logic**

int main() {

RoleBasedUser admin("admin", "admin123", ADMIN);

InventoryManager inventoryManager;

string username, password;

cout << "Login - Username: ";

cin >> username;

cout << "Password: ";

cin >> password;

if (admin.login(username, password)) {

cout << "Login successful!" << endl;

inventoryManager.createProduct("Laptop", 50, 75000);

inventoryManager.createSupplier("TechSupplier", "contact@techsupplier.com");

inventoryManager.displayProducts();

inventoryManager.displaySuppliers();

inventoryManager.updateProduct(0, "Laptop", 45, 74000);

inventoryManager.updateSupplier(0, "TechSupplier Co.", "info@techsupplier.com");

inventoryManager.displayProducts();

inventoryManager.displaySuppliers();

inventoryManager.deleteProduct(0);

inventoryManager.deleteSupplier(0);

} else {

cout << "Invalid credentials!" << endl;

}

return 0;

}