Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Институт информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: «Программная инженерия»

Дисциплина: «Объектно-Ориентированное Программирование»

**Курсовой проект**

**Тема: «Складской учёт»**

Выполнил: студент группы ПИ-3-21

Алтынбеков Даурен

Проверил: Мусабаев Э.Б

**Бишкек 2024**

Оглавление

[**Введение** 3](#_Toc167482726)

[**Описание классов** 3](#_Toc167482727)

[**Класс Product** 3](#_Toc167482728)

[**Класс Cart** 3](#_Toc167482729)

[**Класс Order** 4](#_Toc167482730)

[**Класс OrderInventory** 4](#_Toc167482731)

[**Класс Inventory** 4](#_Toc167482732)

[**Класс User** 5](#_Toc167482733)

[**Класс Client** 6](#_Toc167482734)

[**Класс Manager** 6](#_Toc167482735)

[**Класс Provider** 6](#_Toc167482736)

[**Класс ProviderInventory** 6](#_Toc167482737)

[**Диаграмма классов** 7](#_Toc167482738)

[**Листинг программы** 8](#_Toc167482739)

[**Тесты** 26](#_Toc167482740)

[**Заключение** 33](#_Toc167482741)

# **Введение**

Программа предназначена для автоматизации складского учёта. Она обеспечивает хранение информации о товарах, поставщиках, заказах и пользователях. Основная цель программы – упрощение управления складом, улучшение контроля за запасами, а также упрощение процесса заказа товаров.

# **Описание классов**

## **Класс Product**

**Описание:** Класс, представляющий продукт в магазине.

**Назначение:** Хранение информации о продукте, такой как идентификатор, название, категория, цена, количество и идентификатор поставщика.

**Поля:**

* **int id**: идентификатор продукта.
* **string name**: название продукта.
* **string category**: категория продукта.
* **double price**: цена продукта.
* **int quantity**: количество продукта.
* **int providerId**: идентификатор поставщика продукта.

**Методы:**

* **Product(int id, string name, string category, double price, int quantity, int providerId):** конструктор для инициализации полей товара.
* **int getId() const:** метод для получения идентификатора товара.
* **void setId(int id):** метод для установки идентификатора товара.
* **string getName() const**: метод для получения названия товара.
* **void setName(string name):** метод для установки названия товара.
* **string getCategory() const**: метод для получения категории товара.
* **void setCategory(string category**): метод для установки категории товара.
* **double getPrice() const:** метод для получения цены товара.
* **void setPrice(double price):** метод для установки цены товара.
* **int getQuantity() const:** метод для получения количества товара.
* **void setQuantity(int quantity):** метод для установки количества товара.
* **int getProviderId() const**: метод для получения идентификатора поставщика товара.
* **void setProviderId(int providerId):** метод для установки идентификатора поставщика товара.

## **Класс Cart**

**Описание:** Класс, представляющий корзину покупок клиента.

**Назначение:** Хранение продуктов, добавленных клиентом в корзину, а также предоставление методов для управления содержимым корзины.

**Поля:**

* **vector<Product> products**: вектор продуктов в корзине.

**Методы:**

* **void addProduct(const Product& product)**: метод для добавления продукта в корзину.
* **void removeProduct(int id)**: метод для удаления продукта из корзины по его идентификатору.
* **void viewCart() const**: метод для отображения содержимого корзины.
* **bool isEmpty() const**: метод для проверки, пуста ли корзина.
* **const vector<Product>& getProducts() const**: метод для получения списка продуктов в корзине.

## **Класс Order**

**Описание:** Класс, представляющий заказ товаров.

**Назначение:** Хранение информации о заказе, такой как идентификатор заказа и список продуктов.

**Поля:**

* **int orderId**: идентификатор заказа.
* **vector<Product> products**: список продуктов в заказе.

**Методы:**

* **Order(int orderId, const vector<Product>& products)**: конструктор для инициализации полей заказа.
* **int getOrderId() const**: метод для получения идентификатора заказа.
* **void setOrderId(int orderId)**: метод для установки идентификатора заказа.
* **const vector<Product>& getProducts() const**: метод для получения списка продуктов в заказе.
* **void displayOrder() const**: метод для отображения информации о заказе.

## **Класс OrderInventory**

**Описание:** Класс для управления заказами.

**Назначение:** Хранение информации о заказах, а также предоставление методов для добавления, удаления, обновления и загрузки заказов.

**Поля:**

* **vector<Order> orders**: вектор заказов.

**Методы:**

* **void saveOrderInventoryToFile(string filename)**: метод для сохранения информации о заказах в файл.
* **void addOrder(Order order)**: метод для добавления заказа.
* **void updateOrder(int orderId, const vector<Product>& products)**: метод для обновления информации о заказе.
* **void removeOrder(int orderId)**: метод для удаления заказа.
* **void printOrders() const**: метод для вывода информации о всех заказах.
* **void loadOrdersFromFile(string filename)**: метод для загрузки информации о заказах из файла.

## **Класс Inventory**

**Описание:** Класс для управления инвентарем товаров.

**Назначение:** Хранение информации о товарах, а также предоставление методов для добавления, удаления, обновления и загрузки товаров.

**Поля:**

* **vector<Product> products**: вектор товаров.

**Методы:**

* **void saveInventoryToFile(string filename)**: метод для сохранения информации о товарах в файл.
* **void addProduct(Product product)**: метод для добавления товара.
* **void updateProduct(int id, string name, string category, double price, int quantity, int providerId)**: метод для обновления информации о товаре.
* **void removeProduct(int id)**: метод для удаления товара.
* **Product\* findProduct(int id)**: метод для поиска товара по его идентификатору.
* **void printProduct() const**: метод для вывода информации о всех товарах.
* **void loadInventoryFromFile(string filename)**: метод для загрузки информации о товарах из файла.

## **Класс User**

**Описание:** Абстрактный базовый класс для представления пользователей системы.

**Назначение:** Предоставление базовой функциональности для всех типов пользователей, такой как аутентификация и управление учетной записью.

**Поля:**

* **int id**: идентификатор пользователя.
* **string name**: имя пользователя.
* **string login**: логин пользователя.
* **string password**: пароль пользователя.
* **string number**: контактный номер пользователя.
* **bool isAuth**: флаг, указывающий, аутентифицирован ли пользователь.
* **string role**: роль пользователя в системе.

**Методы:**

* **User()**: конструктор по умолчанию.
* **~User()**: виртуальный деструктор.
* **string getName() const**: метод для получения имени пользователя.
* **string getRole() const**: метод для получения роли пользователя.
* **void signUp(string name, string login, string password, string number, string role)**: метод для регистрации нового пользователя.
* **bool loginUser(string login, string password)**: метод для входа пользователя в систему.
* **void saveUserToFile(string filename)**: метод для сохранения информации о пользователе в файл.
* **bool isAuthenticated() const**: метод для проверки, аутентифицирован ли пользователь.
* **virtual void displayInterface() const = 0**: виртуальный метод для отображения интерфейса пользователя.
* **void displayInfo() const**: метод для отображения информации о пользователе.

## **Класс Client**

**Описание:** Класс для представления клиентов системы.

**Назначение:** Предоставление интерфейса и функциональности, специфичных для клиентов.

**Методы:**

* **Client()**: конструктор по умолчанию.
* **void displayInterface() const override**: метод для отображения интерфейса клиента.

## **Класс Manager**

**Описание:** Класс для представления менеджеров системы.

**Назначение:** Предоставление интерфейса и функциональности, специфичных для менеджеров.

**Методы:**

* **Manager()**: конструктор по умолчанию.
* **void displayInterface() const override**: метод для отображения интерфейса менеджера.
* **void displayProductInterface() const**: метод для отображения интерфейса управления товарами.
* **void displayProviderInterface() const**: метод для отображения интерфейса управления поставщиками.

## **Класс Provider**

**Описание:** Класс для представления поставщиков товаров.

**Назначение:** Хранение информации о поставщиках.

**Поля:**

* **int id**: идентификатор поставщика.
* **string name**: имя поставщика.
* **string contactInfo**: контактная информация поставщика.

**Методы:**

* **Provider(int id, string name, string contactInfo)**: конструктор для инициализации полей поставщика.
* **int getId() const**: метод для получения идентификатора поставщика.
* **void setId(int id)**: метод для установки идентификатора поставщика.
* **string getName() const**: метод для получения имени поставщика.
* **void setName(string name)**: метод для установки имени поставщика.
* **string getContactInfo() const**: метод для получения контактной информации поставщика.
* **void setContactInfo(string contactInfo)**: метод для установки контактной информации поставщика.

## **Класс ProviderInventory**

**Описание:** Класс для управления информацией о поставщиках.

**Назначение:** Хранение информации о поставщиках, а также предоставление методов для добавления, удаления, обновления и загрузки информации о поставщиках.

**Поля:**

* **vector<Provider> providers**: вектор объектов класса **Provider**, представляющих поставщиков.

**Методы:**

* **void saveProviderInventoryToFile(string filename)**: метод для сохранения информации о поставщиках в файл.
* **void addProvider(Provider provider)**: метод для добавления нового поставщика.
* **void updateProvider(int id, string name, string contactInfo)**: метод для обновления информации о поставщике.
* **void removeProvider(int id)**: метод для удаления поставщика.
* **void printProviders() const**: метод для вывода информации о всех поставщиках.
* **void loadProvidersFromFile(string filename)**: метод для загрузки информации о поставщиках из файла.

# **Диаграмма классов**

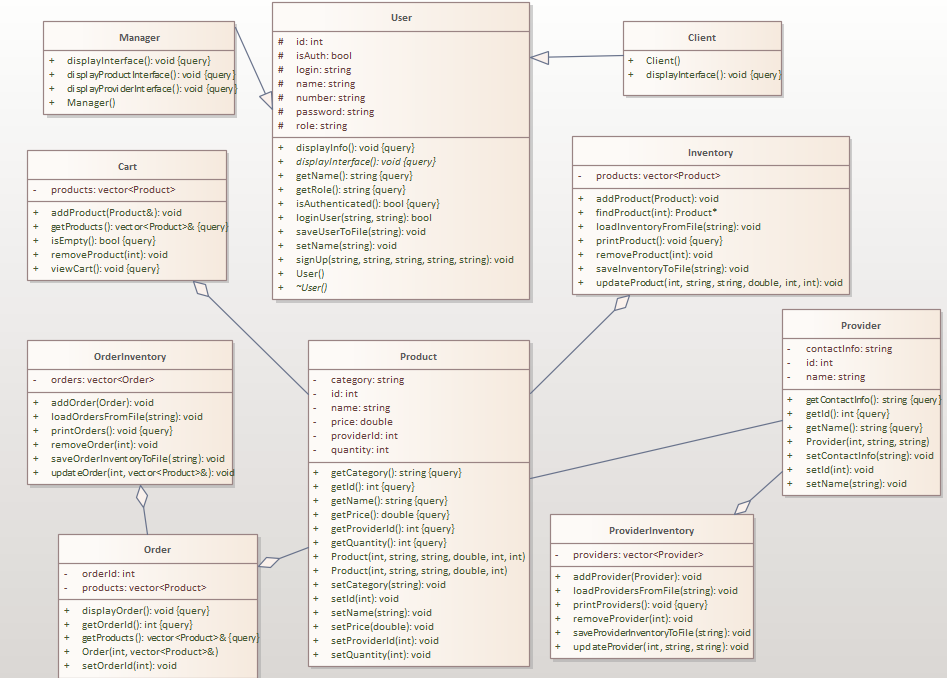


Рисунок 1. Диаграмма классов

# **Листинг программы**

#include <iostream>

#include <vector>

#include <string>

#include <sstream>

#include <fstream>

using namespace std;

const string MANAGER\_KEY = "MANAGER\_KEY";

static int orderIdCounter = 2;

class Product {

private:

int id;

string name;

string category;

double price;

int quantity;

int providerId;

public:

Product(int id, string name, string category, double price, int quantity, int providerId)

: id(id), name(name), category(category), price(price), quantity(quantity), providerId(providerId) {}

Product(int id, string name, string category, double price, int quantity)

: id(id), name(name), category(category), price(price), quantity(quantity), providerId(1) {}

int getId() const { return id; }

void setId(int id) { this->id = id; }

string getName() const { return name; }

void setName(string name) { this->name = name; }

string getCategory() const { return category; }

void setCategory(string category) { this->category = category; }

double getPrice() const { return price; }

void setPrice(double price) { this->price = price; }

int getQuantity() const { return quantity; }

void setQuantity(int quantity) { this->quantity = quantity; }

int getProviderId() const { return providerId; }

void setProviderId(int providerId) { this->providerId = providerId; }

};

class Cart {

private:

vector<Product> products;

public:

void addProduct(const Product& product) {

products.push\_back(product);

}

void removeProduct(int id) {

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

if (it->getId() == id) {

products.erase(it);

return;

}

}

}

void viewCart() const {

cout << "Cart contents:" << endl;

for (const auto& product : products) {

cout << "ID: " << product.getId() << ", Name: " << product.getName()

<< ", Category: " << product.getCategory() << ", Price: " << product.getPrice()

<< ", Quantity: " << product.getQuantity() << ", Provider ID: " << product.getProviderId() << endl;

}

}

bool isEmpty() const {

return products.empty();

}

const vector<Product>& getProducts() const {

return products;

}

};

class Order {

private:

int orderId;

vector<Product> products;

public:

Order(int orderId, const vector<Product>& products)

: orderId(orderId), products(products) {}

int getOrderId() const { return orderId; }

void setOrderId(int orderId) { this->orderId = orderId; }

const vector<Product>& getProducts() const { return products; }

void displayOrder() const {

cout << "Order ID: " << orderId << endl;

for (const auto& product : products) {

cout << "ID: " << product.getId() << ", Name: " << product.getName()

<< ", Category: " << product.getCategory() << ", Price: " << product.getPrice()

<< ", Quantity: " << product.getQuantity() << ", Provider ID: " << product.getProviderId() << endl;

}

}

};

class OrderInventory {

private:

vector<Order> orders;

public:

void saveOrderInventoryToFile(string filename) {

ofstream file;

file.open(filename, ios::out | ios::trunc);

for (const auto& order : orders) {

file << order.getOrderId();

for (const auto& product : order.getProducts()) {

file << "," << product.getId() << "," << product.getName() << "," << product.getCategory()

<< "," << product.getPrice() << "," << product.getQuantity() << "," << product.getProviderId();

}

file << endl;

}

file.close();

}

void addOrder(Order order) {

bool found = false;

for (const auto& o : orders) {

if (o.getOrderId() == order.getOrderId()) {

cout << "Order ID already exists." << endl;

found = true;

break;

}

}

if (!found) {

orders.push\_back(order);

cout << "Order added successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

saveOrderInventoryToFile("orders.csv");

}

}

void updateOrder(int orderId, const vector<Product>& products) {

bool found = false;

for (auto& order : orders) {

if (order.getOrderId() == orderId) {

order = Order(orderId, products);

found = true;

saveOrderInventoryToFile("orders.csv");

break;

}

}

if (!found) {

cout << "Order ID does not exist." << endl;

}

}

void removeOrder(int orderId) {

bool found = false;

for (auto i = orders.begin(); i != orders.end(); i++) {

if (i->getOrderId() == orderId) {

orders.erase(i);

found = true;

cout << "Order removed successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

saveOrderInventoryToFile("orders.csv");

break;

}

}

if (!found) {

cout << "Order ID does not exist." << endl;

}

}

void printOrders() const {

for (const auto& order : orders) {

cout << "-----------------------------------------------------------" << endl;

order.displayOrder();

cout << "-----------------------------------------------------------" << endl;

}

}

void loadOrdersFromFile(string filename) {

ifstream file;

file.open(filename);

if (file.is\_open()) {

string line;

while (getline(file, line)) {

stringstream ss(line);

string orderIdStr;

getline(ss, orderIdStr, ',');

int orderId = stoi(orderIdStr);

vector<Product> products;

string idStr, name, category, priceStr, quantityStr, providerIdStr;

while (getline(ss, idStr, ',')) {

getline(ss, name, ',');

getline(ss, category, ',');

getline(ss, priceStr, ',');

getline(ss, quantityStr, ',');

getline(ss, providerIdStr, ',');

int id = stoi(idStr);

double price = stod(priceStr);

int quantity = stoi(quantityStr);

int providerId = stoi(providerIdStr);

Product product(id, name, category, price, quantity, providerId);

products.push\_back(product);

}

Order order(orderId, products);

orders.push\_back(order);

orderIdCounter=orders.back().getOrderId() + 1;

}

file.close();

}

else {

cout << "Error: Could not open file " << filename << endl;

}

}

};

class Inventory {

private:

vector<Product> products;

public:

void saveInventoryToFile(string filename) {

ofstream file;

file.open(filename, ios::out | ios::trunc);

for (const auto& product : products) {

file << product.getId() << "," << product.getName() << "," << product.getCategory() << "," << product.getPrice() << "," << product.getQuantity() << "," << product.getProviderId() << endl;

}

file.close();

}

void addProduct(Product product) {

bool found = false;

for (const auto& p : products) {

if (p.getId() == product.getId()) {

cout << "Id already exists." << endl;

found = true;

break;

}

}

if (!found) {

products.push\_back(product);

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

cout << "-----------------------------------------------------------" << endl;

saveInventoryToFile("inventory.csv");

}

}

void updateProduct(int id, string name, string category, double price, int quantity, int providerId) {

bool found = false;

for (auto& product : products) {

if (product.getId() == id) {

product.setName(name);

product.setCategory(category);

product.setPrice(price);

product.setQuantity(quantity);

product.setProviderId(providerId);

found = true;

saveInventoryToFile("inventory.csv");

break;

}

}

if (!found) {

cout << "ID does not exist." << endl;

}

}

void removeProduct(int id) {

bool found = false;

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

if (i->getId() == id) {

products.erase(i);

found = true;

cout << "Product removed successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

saveInventoryToFile("inventory.csv");

break;

}

}

if (!found) {

cout << "Id does not exist" << endl;

}

}

Product\* findProduct(int id) {

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

if (i->getId() == id) {

return &(\*i);

}

}

return nullptr;

}

void printProduct() const {

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

cout << "-----------------------------------------------------------" << endl;

cout << "ID : " << i->getId() << endl;

cout << "Name : " << i->getName() << endl;

cout << "Category : " << i->getCategory() << endl;

cout << "Price : $" << i->getPrice() << endl;

cout << "Quantity : " << i->getQuantity() << endl;

cout << "Provider ID : " << i->getProviderId() << endl;

cout << "-----------------------------------------------------------" << endl;

}

}

void loadInventoryFromFile(string filename) {

try {

ifstream file;

file.open(filename);

if (file.is\_open()) {

string line;

while (getline(file, line)) {

stringstream ss(line);

string idStr, name, category, priceStr, quantityStr, providerIdStr;

getline(ss, idStr, ',');

getline(ss, name, ',');

getline(ss, category, ',');

getline(ss, priceStr, ',');

getline(ss, quantityStr, ',');

getline(ss, providerIdStr, ',');

int id = stoi(idStr);

double price = stod(priceStr);

int quantity = stoi(quantityStr);

int providerId = stoi(providerIdStr);

Product p(id, name, category, price, quantity, providerId);

products.push\_back(p);

}

file.close();

}

else {

cout << "Error: Could not open file " << filename << endl;

}

}

catch (const exception& e) {

cerr << "Exception: " << e.what() << endl;

}

}

};

class User {

protected:

int id;

string name;

string login;

string password;

string number;

bool isAuth;

string role;

public:

User() : id(0), isAuth(false) {}

virtual ~User() {}

string getName() const {

return name;

}

void setName(string name) {

this->name = name;

}

string getRole() const {

return role;

}

void signUp(string name, string login, string password, string number, string role) {

this->name = name;

this->login = login;

this->password = password;

this->number = number;

this->role = role;

saveUserToFile("users.csv");

}

bool loginUser(string login, string password) {

ifstream file("users.csv");

if (!file.is\_open()) {

cout << "Error: Could not open users file." << endl;

return false;

}

string line;

while (getline(file, line)) {

stringstream ss(line);

string idStr, name, loginFromFile, passwordFromFile, number, role;

getline(ss, idStr, ',');

getline(ss, name, ',');

getline(ss, loginFromFile, ',');

getline(ss, passwordFromFile, ',');

getline(ss, number, ',');

getline(ss, role, ',');

if (login == loginFromFile && password == passwordFromFile) {

this->id = stoi(idStr);

this->name = name;

this->login = loginFromFile;

this->password = passwordFromFile;

this->number = number;

this->role = role;

this->isAuth = true;

file.close();

return true;

}

}

file.close();

return false;

}

void saveUserToFile(string filename) {

ofstream file;

file.open(filename, ios::out | ios::app);

file << id << "," << name << "," << login << "," << password << "," << number << "," << role << endl;

file.close();

}

bool isAuthenticated() const {

return isAuth;

}

virtual void displayInterface() const = 0;

void displayInfo() const {

cout << "User ID: " << id << endl;

cout << "Name: " << name << endl;

cout << "Login: " << login << endl;

cout << "Phone Number: " << number << endl;

cout << "Role: " << role << endl;

cout << "Authenticated: " << (isAuth ? "Yes" : "No") << endl;

}

};

class Client : public User {

public:

Client() : User() {

this->role = "client";

}

void displayInterface() const override {

cout << "--------------------Client Interface-----------------------" << endl;

cout << "1. View Inventory" << endl;

cout << "2. Add Product to Cart" << endl;

cout << "3. Remove Product from Cart" << endl;

cout << "4. View Cart" << endl;

cout << "5. Place Order" << endl;

cout << "6. Find Product" << endl;

cout << "0. Logout" << endl;

cout << "Enter choice: ";

cout << "-----------------------------------------------------------" << endl;

}

};

class Manager : public User {

public:

Manager() : User() {

this->role = "manager";

}

void displayInterface() const override {

cout << "--------------------Manager Interface----------------------" << endl;

cout << "1. Manage Products" << endl;

cout << "2. View Orders" << endl;

cout << "3. Manage Providers" << endl;

cout << "4. Logout" << endl;

cout << "-----------------------------------------------------------" << endl;

}

void displayProductInterface() const {

cout << "------------------Product Management-----------------------" << endl;

cout << "1. Add Product" << endl;

cout << "2. Remove Product" << endl;

cout << "3. Update Product" << endl;

cout << "4. View Inventory" << endl;

cout << "5. Back to Main Menu" << endl;

cout << "-----------------------------------------------------------" << endl;

}

void displayProviderInterface() const {

cout << "-----------------Provider Management-----------------------" << endl;

cout << "Provider Management:" << endl;

cout << "1. Add Provider" << endl;

cout << "2. Remove Provider" << endl;

cout << "3. Update Provider" << endl;

cout << "4. View Providers" << endl;

cout << "5. Back to Main Menu" << endl;

cout << "-----------------------------------------------------------" << endl;

}

};

class Provider {

private:

int id;

string name;

string contactInfo;

public:

Provider(int id, string name, string contactInfo)

: id(id), name(name), contactInfo(contactInfo) {}

int getId() const { return id; }

void setId(int id) { this->id = id; }

string getName() const { return name; }

void setName(string name) { this->name = name; }

string getContactInfo() const { return contactInfo; }

void setContactInfo(string contactInfo) { this->contactInfo = contactInfo; }

};

class ProviderInventory {

private:

vector<Provider> providers;

public:

void saveProviderInventoryToFile(string filename) {

ofstream file;

file.open(filename, ios::out | ios::trunc);

for (const auto& provider : providers) {

file << provider.getId() << "," << provider.getName() << "," << provider.getContactInfo() << endl;

}

file.close();

}

void addProvider(Provider provider) {

bool found = false;

for (const auto& p : providers) {

if (p.getId() == provider.getId()) {

cout << "Id already exists." << endl;

found = true;

break;

}

}

if (!found) {

providers.push\_back(provider);

cout << "Provider added successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

saveProviderInventoryToFile("providers.csv");

}

}

void updateProvider(int id, string name, string contactInfo) {

bool found = false;

for (auto& provider : providers) {

if (provider.getId() == id) {

provider.setName(name);

provider.setContactInfo(contactInfo);

found = true;

saveProviderInventoryToFile("providers.csv");

break;

}

}

if (!found) {

cout << "ID does not exist." << endl;

}

}

void removeProvider(int id) {

bool found = false;

for (auto i = providers.begin(); i != providers.end(); i++) {

if (i->getId() == id) {

providers.erase(i);

found = true;

cout << "Provider removed successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

saveProviderInventoryToFile("providers.csv");

break;

}

}

if (!found) {

cout << "Id does not exist" << endl;

}

}

void printProviders() const {

for (const auto& provider : providers) {

cout << "-----------------------------------------------------------" << endl;

cout << "ID : " << provider.getId() << endl;

cout << "Name : " << provider.getName() << endl;

cout << "Contact Info : " << provider.getContactInfo() << endl;

cout << "-----------------------------------------------------------" << endl;

}

}

void loadProvidersFromFile(string filename) {

ifstream file;

file.open(filename);

if (file.is\_open()) {

string line;

while (getline(file, line)) {

stringstream ss(line);

string idStr, name, contactInfo;

getline(ss, idStr, ',');

getline(ss, name, ',');

getline(ss, contactInfo, ',');

int id = stoi(idStr);

Provider p(id, name, contactInfo);

providers.push\_back(p);

}

file.close();

}

else {

cout << "Error: Could not open file " << filename << endl;

}

}

};

int main() {

Inventory inventory;

inventory.loadInventoryFromFile("inventory.csv");

ProviderInventory providerInventory;

providerInventory.loadProvidersFromFile("providers.csv");

OrderInventory orderInventory;

orderInventory.loadOrdersFromFile("orders.csv");

cout << "-----------------------------------------------------------" << endl;

cout << "---------------------Auth to continue----------------------" << endl;

cout << "-----------------------------------------------------------" << endl;

User\* currentUser = nullptr;

Cart cart;

int userChoice;

do {

cout << "1. Sign Up" << endl;

cout << "2. Login" << endl;

cout << "0. Exit" << endl;

cout << "Enter choice: ";

cin >> userChoice;

if (userChoice == 1) {

string name, login, password, number, role;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Login: ";

cin >> login;

cout << "Enter Password: ";

cin >> password;

cout << "Enter Phone Number: ";

cin >> number;

while (true) {

cout << "Enter Role (client/manager): ";

cin >> role;

if (role == "client") {

currentUser = new Client();

break;

}

else if (role == "manager") {

string managerKey;

cout << "Enter Manager Key: ";

cin >> managerKey;

if (managerKey == MANAGER\_KEY) {

currentUser = new Manager();

break;

}

else {

cout << "Invalid Manager Key. Please try again." << endl;

}

}

else {

cout << "Invalid role. Please enter 'client' or 'manager'." << endl;

}

}

currentUser->signUp(name, login, password, number, role);

cout << "User signed up successfully." << endl;

}

else if (userChoice == 2) {

string login, password;

cout << "Enter Login: ";

cin >> login;

cout << "Enter Password: ";

cin >> password;

currentUser = new Client();

if (currentUser->loginUser(login, password)) {

if (currentUser->getRole() == "manager") {

delete currentUser;

currentUser = new Manager();

currentUser->loginUser(login, password);

cout << "-----------------------------------------------------------" << endl;

cout << "----------------Login successful as Manager----------------" << endl;

cout << "------------------------- Welcome! ------------------------" << endl;

}

else {

cout << "-----------------------------------------------------------" << endl;

cout << "----------------Login successful as Client-----------------" << endl;

cout << "------------------------- Welcome! ------------------------" << endl;

}

userChoice = 0;

continue;

}

else {

cout << "Login failed. Incorrect credentials." << endl;

delete currentUser;

currentUser = nullptr;

}

}

else if (userChoice == 0) {

cout << "You exited" << endl;

return 0;

}

else {

cout << "Invalid choice. Try again." << endl;

}

} while (userChoice != 0 || (currentUser && !currentUser->isAuthenticated()));

if (!currentUser || !currentUser->isAuthenticated()) {

cout << "Exiting application due to failed authentication." << endl;

return 0;

}

while (true) {

currentUser->displayInterface();

int choice;

cin >> choice;

if (Client\* client = dynamic\_cast<Client\*>(currentUser)) {

if (choice == 1) {

inventory.printProduct();

}

else if (choice == 2) {

int productId;

cout << "Enter product ID to add to cart: ";

cin >> productId;

Product\* product = inventory.findProduct(productId);

if (product != nullptr) {

cart.addProduct(\*product);

cout << "Product added to cart." << endl;

}

else {

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

}

}

else if (choice == 3) {

int productId;

cout << "Enter product ID to remove from cart: ";

cin >> productId;

cart.removeProduct(productId);

cout << "Product removed from cart." << endl;

}

else if (choice == 4) {

cart.viewCart();

}

else if (choice == 5) {

if (cart.isEmpty()) {

cout << "Cart is empty. Add products to the cart before placing an order." << endl;

}

else {

Order order(orderIdCounter++, cart.getProducts());

orderInventory.addOrder(order);

order.displayOrder();

cart = Cart();

cout << "Order placed successfully." << endl;

}

}

else if (choice == 6) {

int id;

cout << "Enter product id to search: ";

cin >> id;

Product\* product = inventory.findProduct(id);

if (product) {

cout << "Name: " << product->getName() << endl;

cout << "Category: " << product->getCategory() << endl;

cout << "Price: $ " << product->getPrice() << endl;

cout << "Quantity: " << product->getQuantity() << endl;

cout << "-----------------------------------------------------------" << endl;

}

else {

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

cout << "-----------------------------------------------------------" << endl;

}

}

else if (choice == 0) {

cout << "Logging out..." << endl;

delete currentUser;

currentUser = nullptr;

return 0;

}

else {

cout << "Invalid choice. Try again." << endl;

}

}

else if (Manager\* manager = dynamic\_cast<Manager\*>(currentUser)) {

switch (choice) {

case 1: {

while (true) {

manager->displayProductInterface();

int productChoice;

cin >> productChoice;

switch (productChoice) {

case 1: {

int id;

string name, category;

double price;

int quantity;

int providerId;

cout << "Enter ID: ";

cin >> id;

cout << "Enter product name: ";

cin >> name;

cout << "Enter product category: ";

cin >> category;

cout << "Enter product price: $ ";

cin >> price;

cout << "Enter product quantity: ";

cin >> quantity;

cout << "Enter Product Provider id: " << endl;

cin >> providerId;

inventory.updateProduct(id, name, category, price, quantity, providerId);

break;

}

case 2: {

int id;

cout << "Enter product id: ";

cin >> id;

inventory.removeProduct(id);

break;

}

case 3: {

int id;

string name, category;

double price;

int quantity;

int providerId;

cout << "Enter the product id: ";

cin >> id;

cout << "Enter new product name: ";

cin >> name;

cout << "Enter new product category: ";

cin >> category;

cout << "Enter new product price: $ ";

cin >> price;

cout << "Enter new product quantity: ";

cin >> quantity;

cout << "Enter Product Provider id: " << endl;

cin >> providerId;

inventory.updateProduct(id, name, category, price, quantity, providerId);

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

cout << "-----------------------------------------------------------" << endl;

break;

}

case 4: {

inventory.printProduct();

break;

}

case 5: {

goto exitProductMenu;

}

default: {

cout << "Invalid choice. Try again." << endl;

break;

}

}

}

exitProductMenu:

break;

}

case 2: {

orderInventory.printOrders();

break;

}

case 3: {

while (true) {

manager->displayProviderInterface();

int providerChoice;

cin >> providerChoice;

switch (providerChoice) {

case 1: {

int id;

string name, contactInfo;

cout << "Enter ID: ";

cin >> id;

cout << "Enter provider name: ";

cin >> name;

cout << "Enter contact info: ";

cin >> contactInfo;

Provider provider(id, name, contactInfo);

providerInventory.addProvider(provider);

break;

}

case 2: {

int id;

cout << "Enter provider id: ";

cin >> id;

providerInventory.removeProvider(id);

break;

}

case 3: {

int id;

string name, contactInfo;

cout << "Enter the provider id: ";

cin >> id;

cout << "Enter new provider name: ";

cin >> name;

cout << "Enter new contact info: ";

cin >> contactInfo;

providerInventory.updateProvider(id, name, contactInfo);

cout << "Provider updated successfully." << endl;

cout << "-----------------------------------------------------------" << endl;

break;

}

case 4: {

providerInventory.printProviders();

break;

}

case 5: {

goto exitProviderMenu;

}

default: {

cout << "Invalid choice. Try again." << endl;

break;

}

}

}

exitProviderMenu:

break;

}

case 4: {

cout << "Logging out..." << endl;

delete currentUser;

currentUser = nullptr;

return 0;

}

default: {

cout << "Invalid choice. Try again." << endl;

break;

}

}

}

else {

cout << "Invalid user type. Exiting." << endl;

delete currentUser;

return 0;

}

}

return 0;

}

# **Тесты**

При запуске нас приветствует меню, где нам предлагают зарегистрироваться, авторизоваться или завершить программу. Регистрируемся как клиент, вводим необходимые данные. После успешной регистрации, мы авторизуемся с новым аккаунтом.

Далее если мы авторизовались, перед нами выходит меню для клиента.

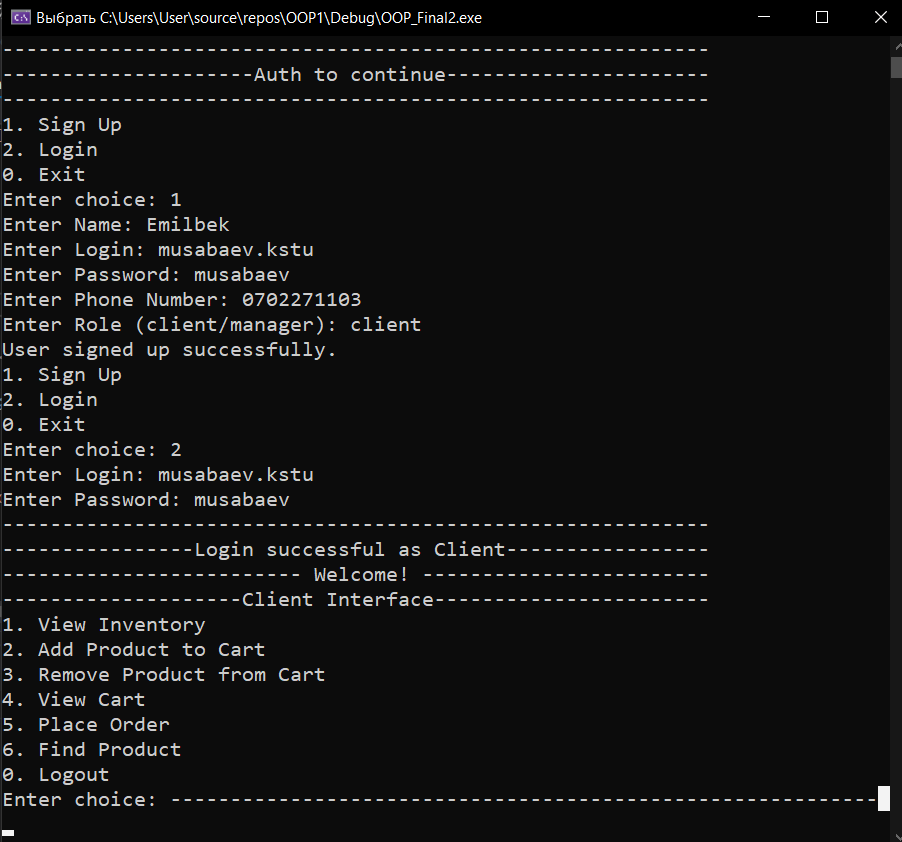


Рисунок 2. Регистрация и авторизация

Все данные о зарегистрированных пользователях, заказах, продуктов и поставщиков у нас хранятся в четырех csv файлах.

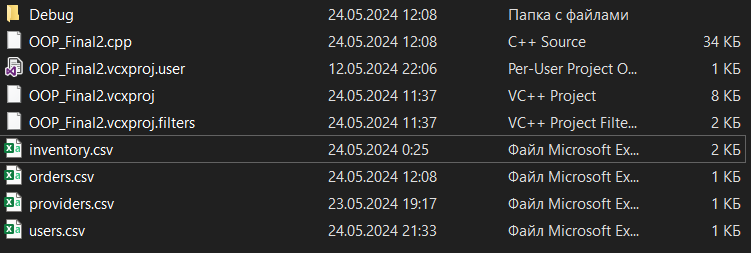


Рисунок 3. Хранимые данные

Как мы можем заметить новый пользователь у нас успешно сохранился в файле users.csv.

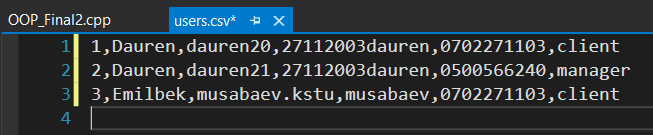


Рисунок 4. Файл с пользователями

Клиент может просматривать все продукты, добавлять продукты в корзину, удалять продукты с корзины, заказывать продукты, хранящиеся в корзине, просматривать свою корзину найти продукт по Id.

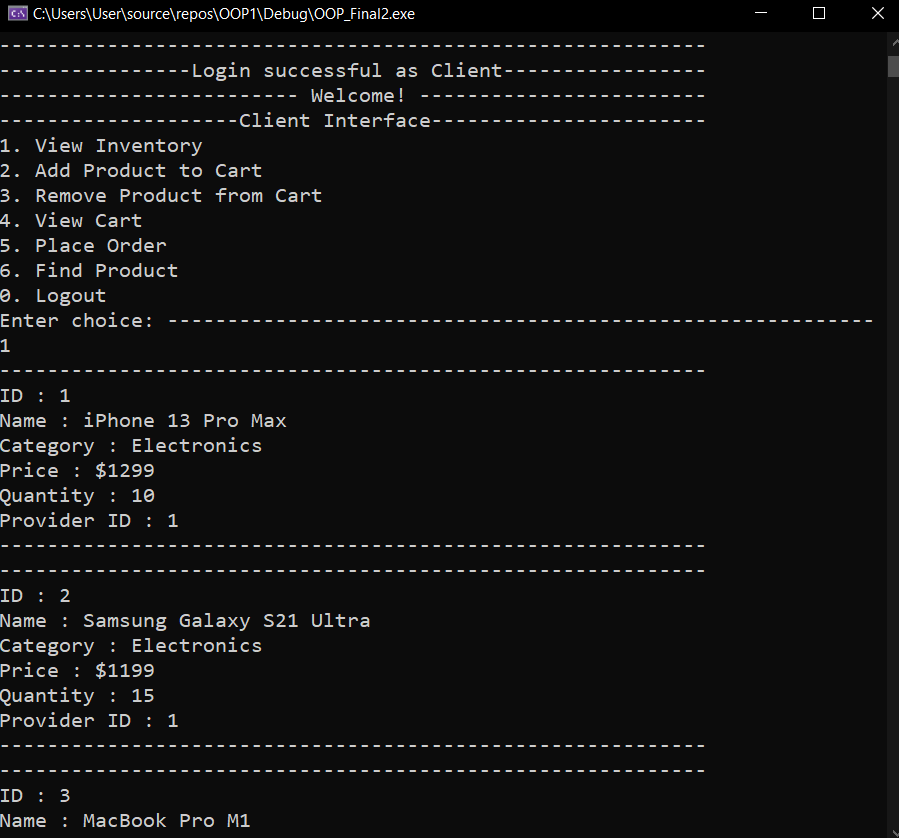


Рисунок 5. Главное меню клиента

Добавим несколько продуктов в корзину.

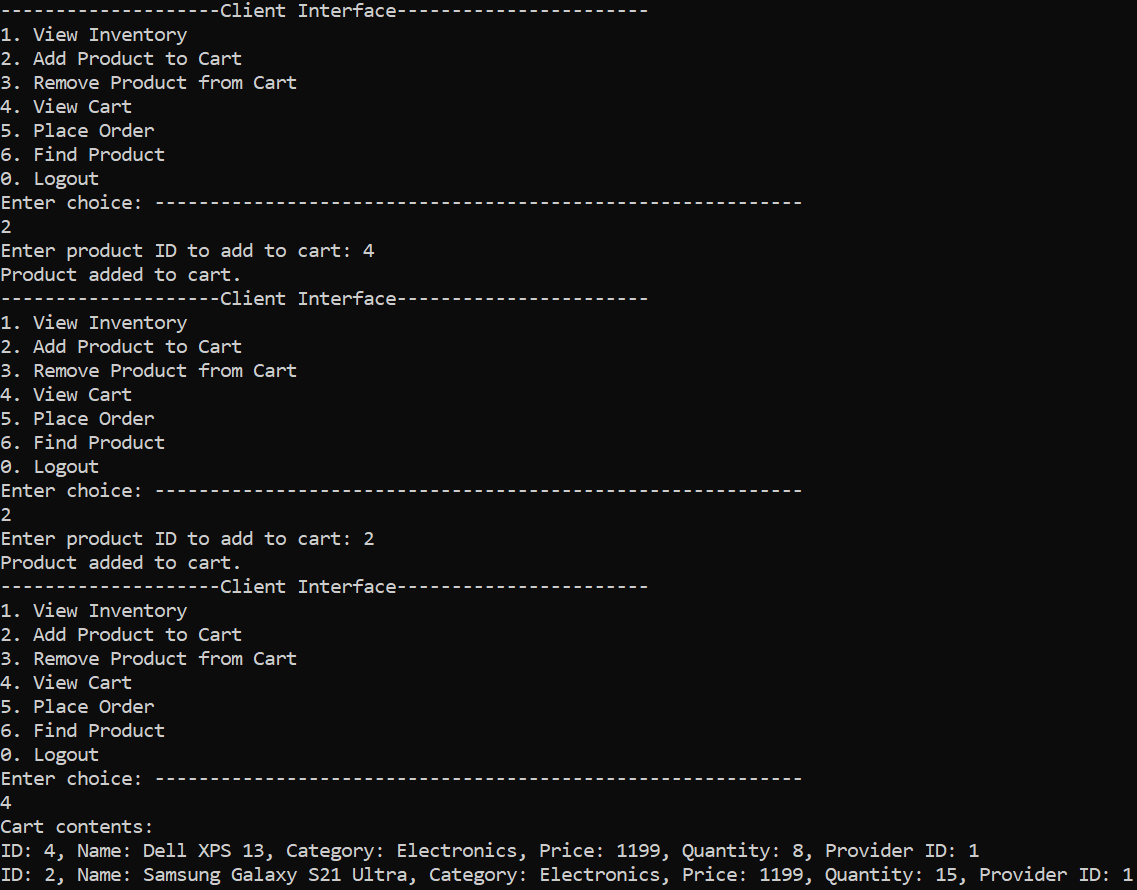


Рисунок 6. Добавление продуктов в корзину

Закажем продукты, которые хранятся у нас в корзине.

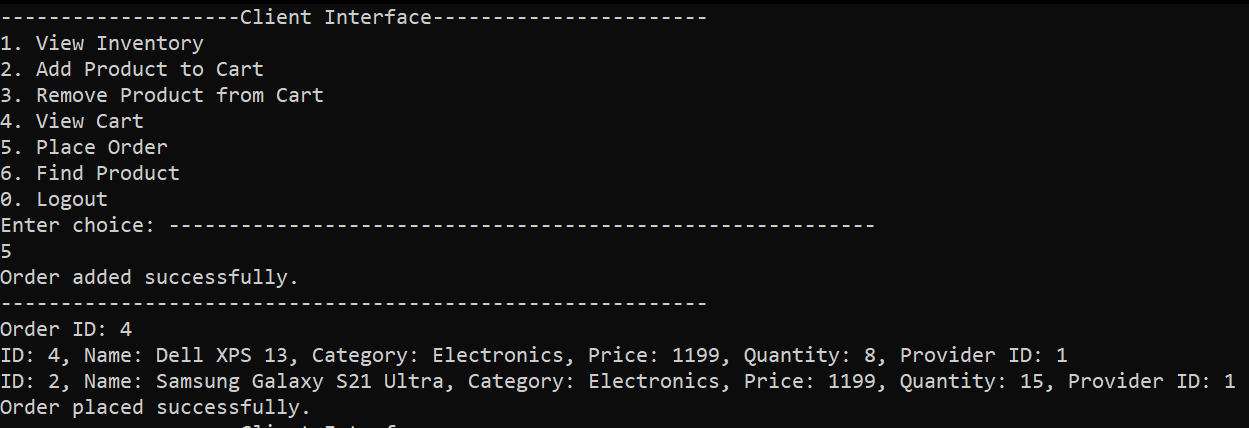


Рисунок 7. Заказ продуктов с корзины

Как мы можем заметить новый заказ добавился в файл для хранения заказов orders.csv, который Менеджер может просмотреть.

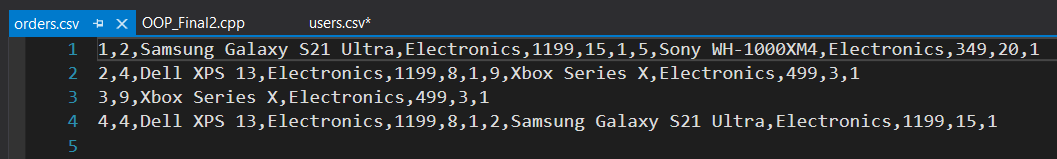


Рисунок 8. Файл с заказами

Теперь зарегистрируемся как Менеджер, для регистрации менеджера необходимо ввести специальный ключ “MANAGER\_KEY”.

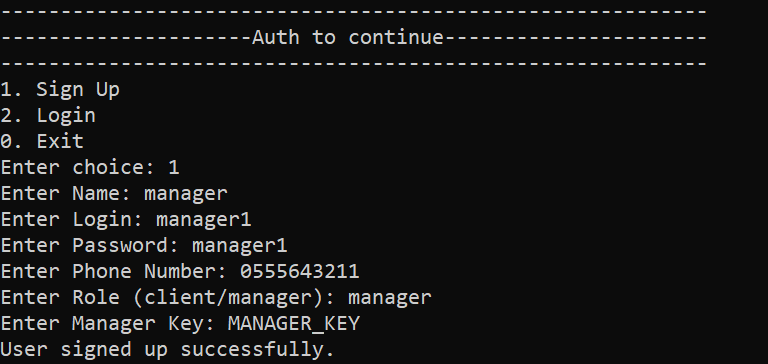


Рисунок 9. Регистрация менеджера

Как можем заметить менеджер успешно зарегистрировался, данные сохранились в файле users.csv.

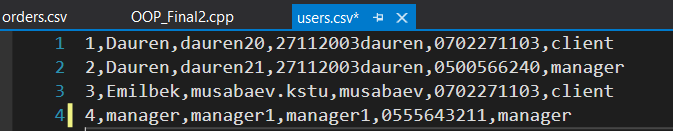


Рисунок 10. Регистрация менеджера

После успешной авторизации, мы попадаем в главное меню менеджера, где нам доступно Управление продуктами, просмотр заказов и управление поставщицами.

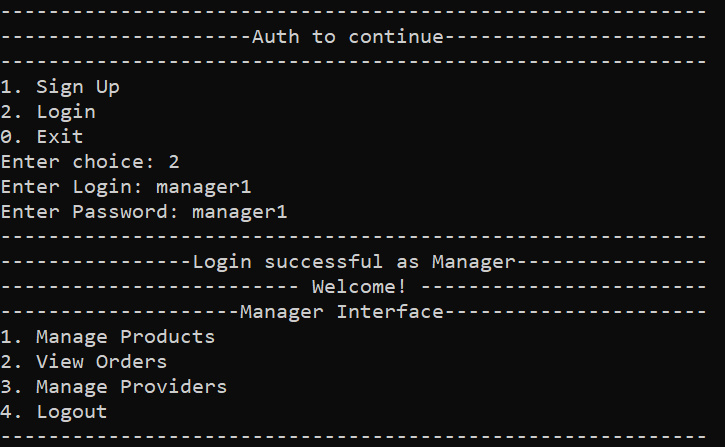


Рисунок 11. Главное меню менеджера

Перейдем в раздел управления продуктами. Тут нам доступны операции по просмотру всех продуктов, удалении продукта, изменения продукта и добавления нового продукта. Выведем список всех продуктов. Данные о всех продуктах берутся считываются с файла inventory.csv.

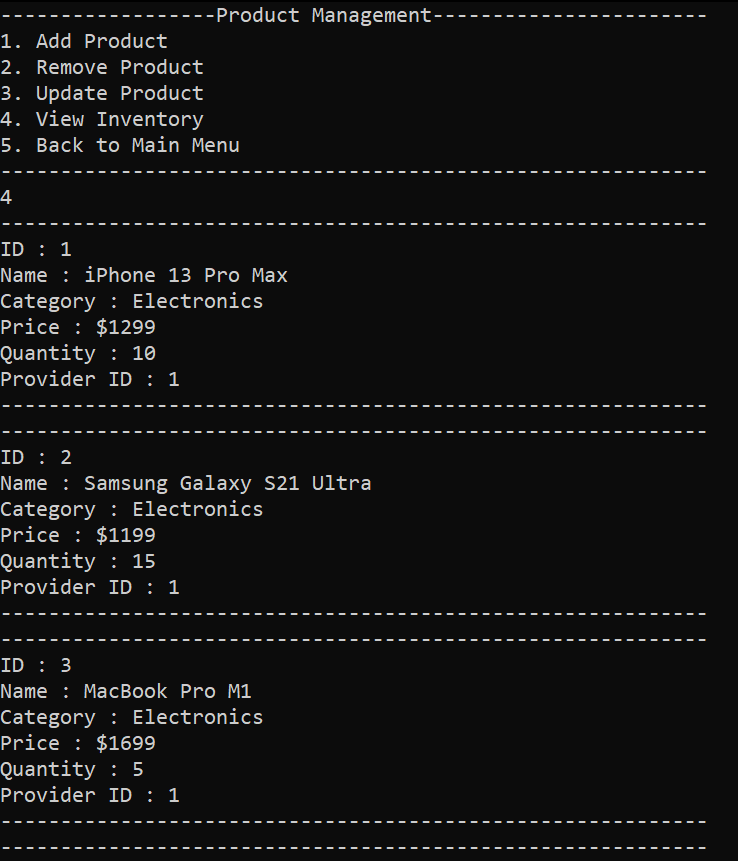


Рисунок 12. Просмотр всех продуктов.

Удалим продукт с id 2. После выведем список всех продуктов, и заметим что продукт с id 2 был успешно удален.

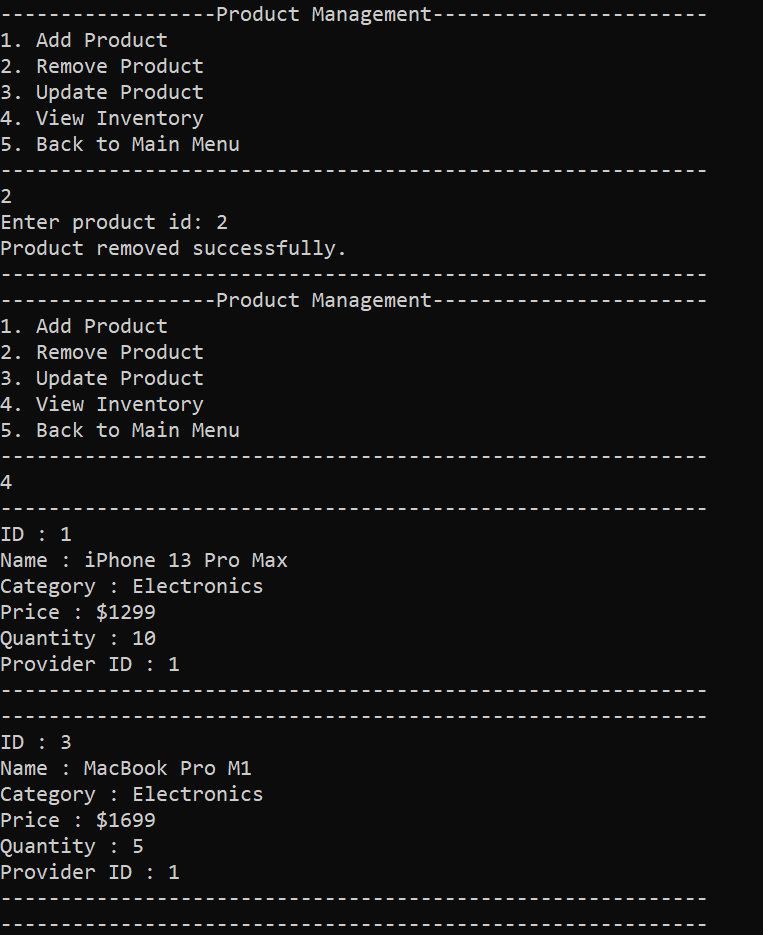


Рисунок 13.Удаление продукта.

Вернемся в главное меню менеджера и выведем список всех заказов.

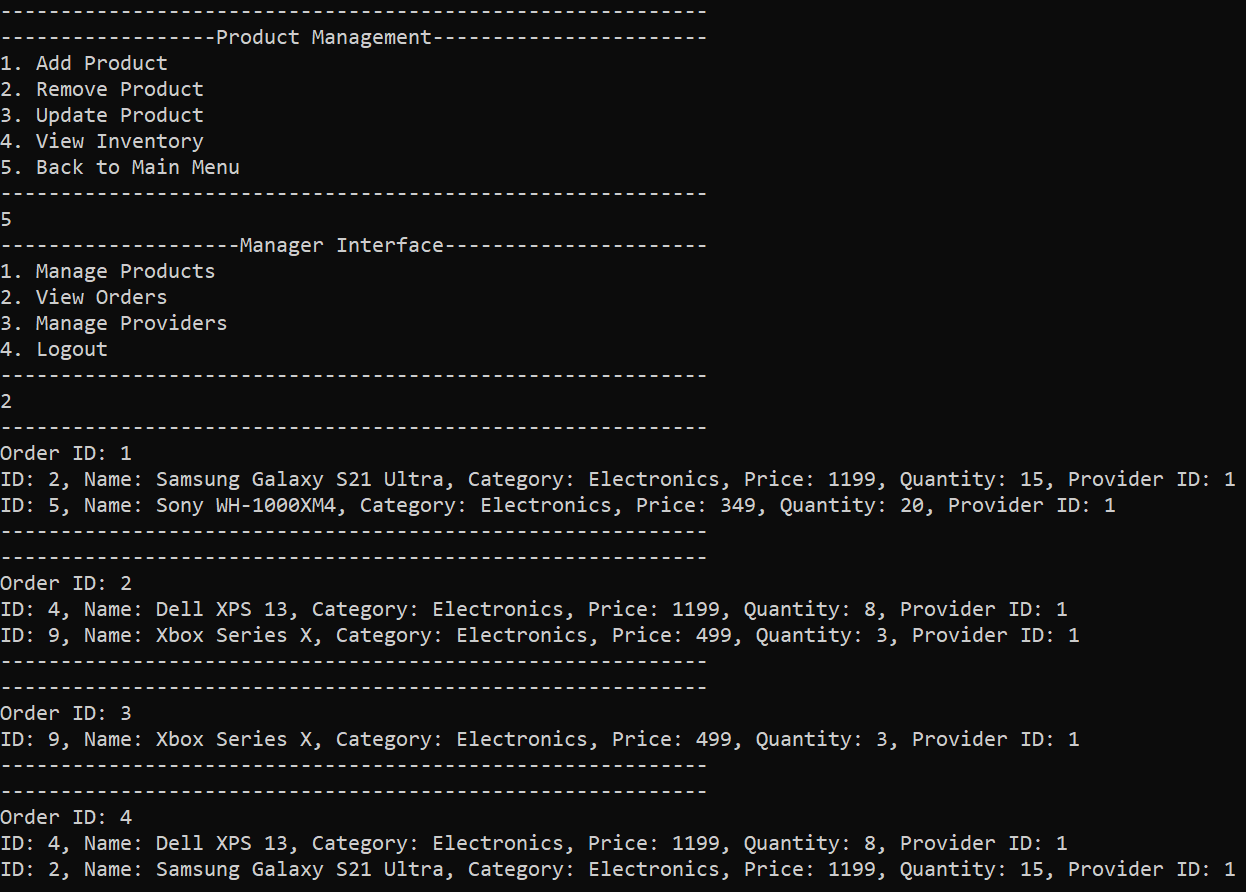


Рисунок 14. Вывод всех заказов.

Далее перейдем в раздел управления поставщиками, где нам доступны такие операции как вывод всех поставщиков, добавление нового поставщика, редактирование поставщика и удаление поставщика. Выведем список всех поставщиков

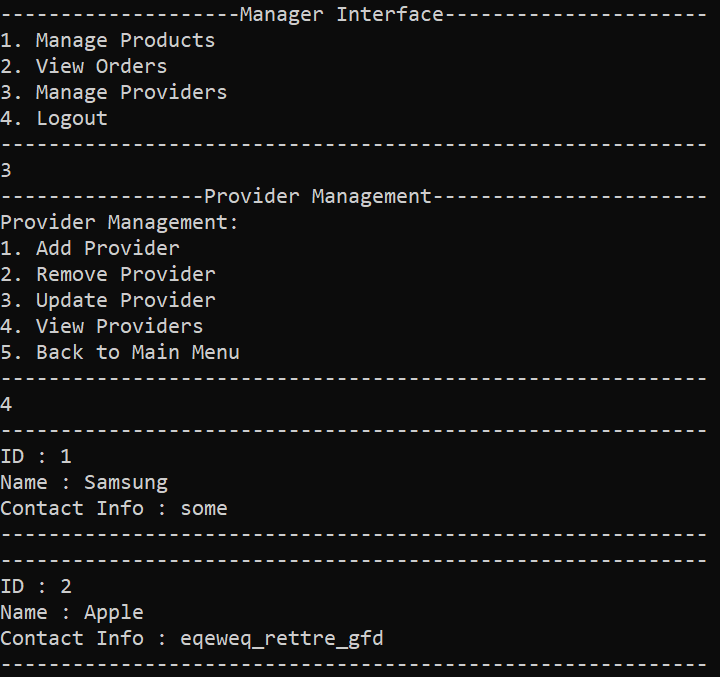


Рисунок 15.Меню управления поставщиками.

# **Заключение**

Программа для складского учёта, представленная в этом проекте, демонстрирует комплексный подход к управлению складом. Она включает в себя функционал для работы с товарами, заказами, поставщиками и пользователями, что позволяет автоматизировать многие рутинные процессы и повысить эффективность управления складом.

Программа обеспечивает следующие ключевые преимущества:

1. **Упрощение управления товарами:** Возможность добавления, обновления и удаления информации о товарах позволяет менеджерам легко поддерживать актуальность данных о складских запасах.
2. **Эффективное оформление заказов:** Функционал для создания и управления заказами помогает клиентам легко оформлять заказы, а менеджерам контролировать их выполнение.
3. **Управление поставщиками:** Возможность добавления, обновления и удаления информации о поставщиках обеспечивает прозрачность и контроль над поставками.
4. **Интерфейсы пользователей:** Разделение ролей пользователей на клиентов и менеджеров с соответствующими интерфейсами упрощает работу с программой и делает её использование интуитивно понятным.
5. **Сохранение данных:** Регулярное сохранение данных о товарах, заказах и поставщиках в файлы гарантирует их сохранность и возможность восстановления при перезапуске программы.

Внедрение такой системы автоматизации складского учёта позволит значительно сократить затраты времени и усилий на управление складом, минимизировать ошибки, связанные с человеческим фактором, и улучшить общий контроль над бизнес-процессами. Это в свою очередь способствует увеличению эффективности и продуктивности работы склада, улучшению обслуживания клиентов и повышению конкурентоспособности предприятия.

В будущем возможны дальнейшие улучшения программы, включая добавление новых функциональных возможностей, таких как отчёты и аналитика, интеграция с другими системами управления и расширение интерфейсов для пользователей. Это позволит сделать систему ещё более универсальной и полезной для различных типов предприятий.

GITHUB https://github.com/Dauren27/ inventory-control