Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Творческая работа**

**«АРМ товароведа продуктового магазина»**

Выполнила:

студентка группы РИС-23-1б

Вилесова Ирина Ивановна

Проверила:

Доцент кафедры ИТАС

О. А. Полякова

2024 г.

**Постановка задачи:**

Проблема работы товароведа в том, что во время ревизии очень сложно сосчитать количество товара в магазине, чтобы соотнести сумму цен остатков продуктов и доход. Поэтому необходимо создать Автоматизированное Рабочее Место товароведа.

**Анализ задачи:**

Для создания используем платформу создания приложений QT, в приложении должна быть предусмотрена авторизация пользователя, добавление продукта, удаление продукта, продажа, и действия с бюджетом.

**Код программы в QT:**

**Autorisation**

#ifndef AUTORISATION\_H

#define AUTORISATION\_H

#include <QDialog>

#include <QMainWindow>

#include <QMessageBox>

#include <QMap>

#include <QFile>

#include <QJsonObject>

#include <QFileDialog>

#include <QStandardItem>

#include <QJsonDocument>

#include <QJsonArray>

#include <QJsonParseError>

#include "login.h"

*namespace* **Ui** {

*class* **Autorisation**;

}

*class* **Autorisation** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **Autorisation**(QWidget \*parent = *nullptr*);

~***Autorisation***();

QJsonDocument doc;

QJsonArray docAr;

QJsonParseError docError;

QJsonObject UsersDataJson;

QJsonObject json;

*private* slots:

void **on\_Registr\_pushButton\_clicked**();

*private*:

Ui::Autorisation \*ui;

Login\* to\_login;

Autorisation\* to\_autor;

void **ToRegistr**();

void **ToLogin**();

QString Log, Pas;

QMap<QString, QString> UsersData;

bool **isLogValid**();

bool **isPasValid**();

};

#endif *//* *AUTORISATION\_H*

#include "autorisation.h"

#include "ui\_autorisation.h"

Autorisation::**Autorisation**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::Autorisation)

{

ui->setupUi(*this*);

QFile file("./Users.json");

setWindowTitle("Регистрация");

file.*open*(QIODevice::*ReadOnly*|QFile::*Text*);

doc = QJsonDocument::fromJson(QByteArray(file.readAll()), *&docError*);

file.*close*();

QJsonValue Users = doc.object().value("Users");

docAr = QJsonValue(Users).toArray();

json = docAr.at(0).toObject();

setFixedSize(width(), height());

}

Autorisation::~***Autorisation***()

{

*delete* ui;

}

bool Autorisation::**isLogValid**()

{

*//Логин* *не* *менее* *4* *символов*

*//Содержит* *хотя* *бы* *одну* *букву*

*//Не* *соддержит* *пробелов*

*//Латиница*

bool Valid = (Log.length() > 3);

bool ContainsW = *false*;

*for* (int i = 0; i < Log.length() && Valid; i++)

{

Valid = (Log[i] != ' ') && (Log[i] < 127);

*if* ((Log[i] > 64 && Log[i] < 91) || (Log[i] > 96 && Log[i] < 123))

{

ContainsW = *true*;

}

}

*return* (Valid && ContainsW);

}

bool Autorisation::**isPasValid**()

{

*//Не* *менее* *8* *символов*

*//Не* *содержит* *пробелов*

bool Valid = (Pas.length() > 7);

*for* (int i = 0; i < Log.length() && Valid; i++) {

Valid = (Log[i] != ' ');

}

*return* Valid;

}

void Autorisation::**ToLogin**()

{

hide();

to\_login=*new* Login(*this*);

to\_login->show();

}

void Autorisation::**ToRegistr**()

{

hide();

to\_autor=*new* Autorisation(*this*);

to\_autor->show();

}

void Autorisation::**on\_Registr\_pushButton\_clicked**()

{

QFile file("./Users.json");

Log = ui->Login\_lineEdit->text();

Pas = ui->Password\_lineEdit->text();

*if* (!isLogValid() && !isPasValid())

{

char a[1000] = "Логин и пароль не корректны!\n\n"

"৹ Логин должен:\n"

"৹ Содержать не менее 4 символов\n"

"৹ Включать в себя хотя бы одну латинскую букву\n"

"৹ Не содержать пробелы\n"

"৹ Базироваться на латинице\n\n"

"Пароль должен:\n"

"৹ Содержать не менее 8 символов\n"

"৹ Не включать в себя пробелы";

QMessageBox::critical(*this*, "Ошибка", a);

ToRegistr();

}

*else* *if* (!isLogValid())

{

char a[1000] = "Логин не корректен!\n\n"

"৹ Логин должен:\n"

"৹ Содержать не менее 4 символов\n"

"৹ Включать в себя хотя бы одну латинскую букву\n"

"৹ Не содержать пробелы\n"

"৹ Базироваться на латинице";

QMessageBox::critical(*this*, "Ошибка", a);

ToRegistr();

}

*else* *if* (!isPasValid())

{

char a[1000] = "Пароль не корректен!\n\n"

"Пароль должен:\n"

"৹ Содержать не менее 8 символов\n"

"৹ Не включать в себя пробелы";

QMessageBox::critical(*this*, "Ошибка", a);

ToRegistr();

}

*else* *if* (json.find(Log) == json.end())

{ *//проверка* *на* *оригинальность* *логина*

file.*open*(QIODevice::*WriteOnly*);

json.insert(Log, Pas);

QJsonArray docToWrite = {};

docToWrite.append(json);

doc.setArray(docToWrite);

file.write("{\"Users\":"+doc.toJson()+"}");

file.*close*();

QMessageBox::about(*this*, "Регистрация", "Регистрация прошла успешно!");

ToLogin();

}

*else*

{

QMessageBox::critical(*this*, "Ошибка", "Логин уже используется!");

ToLogin();

}

}

**Login**

#include "login.h"

#include "ui\_login.h"

#include <autorisation.h>

Login::**Login**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::Login)

{

ui->setupUi(*this*);

setWindowTitle("Вход");

QFile file("./Users.json");

file.*open*(QIODevice::*ReadOnly*|QFile::*Text*);

doc = QJsonDocument::fromJson(QByteArray(file.readAll()), *&docError*);

file.*close*();

QJsonValue Users = doc.object().value("Users");

docAr = QJsonValue(Users).toArray();

json = docAr.at(0).toObject();

}

Login::~***Login***()

{

*delete* ui;

}

void Login::**ToMain**()

{

hide();

main\_product=*new* Product(*this*);

main\_product->show();

}

void Login::**on\_Enter\_pushButton\_clicked**()

{

Log = ui->Login\_reg\_lineEdit->text();

Pas = ui->Password\_reg\_lineEdit->text();

*if* (json.find(Log) != json.end())

{*//если* *логин* *существует*

*if* (json.value(Log) == Pas)

{*//проверка* *сходимости* *паролей*

ToMain();

}

*else*

{

QMessageBox::critical(*this*, "Ошибка", "Логин или пароль не верны!");

}

}

*else*

{

QMessageBox::critical(*this*, "Ошибка", "Указанный логин не существует!");

}

}

**Product**

#ifndef PRODUCT\_H

#define PRODUCT\_H

#include <QMainWindow>

*namespace* **Ui** {

*class* **Product**;

}

*class* **Product** : *public* QMainWindow

{

Q\_OBJECT

*public*:

*explicit* **Product**(QWidget \*parent = *nullptr*);

~***Product***();

void **addProduct**();

void **deleteLProduct**();

void **sellProduct**();

double totalSales = 0.0;

double **getTotalSales**() *const*;

void **calculateTotalBudget**();

void **TotalSalesSum**();

void **getRemainsSales**();

void **performRevision**();

void **displayProductInfo**();

*private* slots:

void **on\_addProduct\_clicked**();

void **on\_deleteLProduct\_clicked**();

void **on\_sellProduct\_clicked**();

void **on\_calculateTotalBudget\_clicked**();

void **on\_sale\_clicked**();

void **on\_Remains\_clicked**();

void **on\_revizion\_clicked**();

void **on\_Information\_clicked**();

*private*:

Ui::Product \*ui;

};

#endif *//* *PRODUCT\_H*

#include "product.h"

#include "ui\_product.h"

#include <QApplication>

#include <QMainWindow>

#include <QPushButton>

#include <QTableWidget>

#include <QLineEdit>

#include <QMessageBox>

Product::**Product**(QWidget \*parent) :

QMainWindow(*parent*),

ui(*new* Ui::Product)

{

ui->setupUi(*this*);

}

Product::~***Product***()

{

*delete* ui;

}

void Product::**addProduct**()

{

*//* *Считывание* *данных* *из* *полей* *ввода*

QString sku = ui->skuLineEdit->text();

QString name = ui->nameLineEdit->text();

double price = ui->priceSpinBox->value();

int quantity = ui->quantitySpinBox->value();

*//* *Проверка* *на* *правильность* *введенных* *данных*

*if* (sku.isEmpty() || name.isEmpty() || price <= 0 || quantity <= 0) {

QMessageBox::warning(*this*, "Внимание", "Все поля должны быть корректно заполнены!");

*return*;

}

*//* *Создание* *новой* *строки* *в* *таблице*

int currentRow = ui->tableWidget->rowCount();

ui->tableWidget->insertRow(currentRow);

QTableWidgetItem \*skuItem = *new* QTableWidgetItem(sku);

QTableWidgetItem \*nameItem = *new* QTableWidgetItem(name);

QTableWidgetItem \*priceItem = *new* QTableWidgetItem(QString::number(price, 'f', 2));

QTableWidgetItem \*quantityItem = *new* QTableWidgetItem(QString::number(quantity));

ui->tableWidget->setItem(currentRow, 0, *nameItem*);

ui->tableWidget->setItem(currentRow, 1, *skuItem*);

ui->tableWidget->setItem(currentRow, 2, *priceItem*);

ui->tableWidget->setItem(currentRow, 3, *quantityItem*);

ui->skuLineEdit->clear();

ui->nameLineEdit->clear();

ui->priceSpinBox->setValue(0.0);

ui->quantitySpinBox->setValue(0);

}

void Product::**deleteLProduct**()

{

*//* *Получаем* *список* *выделенных* *элементов* *в* *таблице*

QList<QTableWidgetItem\*> selectedItems = ui->tableWidget->selectedItems();

*if* (selectedItems.isEmpty()) {

QMessageBox::warning(*this*, "Внимание", "Необходимо выбрать продукт для удаления!");

*return*;

}

*//* *Спрашиваем* *пользователя,* *уверен* *ли* *он* *в* *своем* *действии*

*auto* reply = QMessageBox::question(*this*, "Подтверждение удаления",

"Вы уверены, что хотите удалить выбранные продукты?",

QMessageBox:: *Yes* | QMessageBox::*No*);

*if* (reply == QMessageBox::*Yes*)

{

*//* *Проходимся* *по* *всем* *выделенным* *строкам* *и* *удаляем* *их*

*while* (!selectedItems.isEmpty())

{

*//* *Удаляем* *строку* *каждого* *первого* *элемента* *из* *списка* *выделенных*

int row = selectedItems.first()->row();

ui->tableWidget->removeRow(row);

*//* *Необходимо* *обновить* *список* *выделенных* *элементов* *после* *удаления* *строки*

selectedItems = ui->tableWidget->selectedItems();

}

}

}

void Product::**sellProduct**()

{

*//* *Получаем* *выбранную* *строку*

int currentRow = ui->tableWidget->currentRow();

*if* (currentRow == -1)

{

QMessageBox::warning(*this*, "Внимание", "Выберите продукт для продажи!");

*return*;

}

*//* *Получаем* *элементы* *из* *строки* *таблицы*

QTableWidgetItem \*quantityItem = ui->tableWidget->item(currentRow, 3);

QTableWidgetItem \*priceItem = ui->tableWidget->item(currentRow, 2);

*//* *Обновляем* *количество*

int quantity = quantityItem->text().toInt();

*if* (quantity > 0)

{

quantity--;

quantityItem->setText(QString::number(quantity));

*//* *Обновляем* *сумму* *продаж*

double price = priceItem->text().toDouble();

totalSales += price; *//* *Обновление* *суммы* *продаж*

}

*else*

{

QMessageBox::warning(*this*, "Ошибка", "Продукт невозможно продать, так как его нет на складе!");

}

}

double Product::**getTotalSales**() *const*

{

*return* totalSales;

}

void Product::**calculateTotalBudget**()

{

double totalInventoryValue = 0.0;

double totalSalesValue = 0.0;

*//* *Обход* *всех* *строк* *в* *таблице*

*for* (int row = 0; row < ui->tableWidget->rowCount(); ++row)

{

QTableWidgetItem \*priceItem = ui->tableWidget->item(row, 2);

QTableWidgetItem \*quantityItem = ui->tableWidget->item(row, 3);

*if* (!priceItem || !quantityItem) *continue*; *//* *Пропустить* *пустые* *строки*

double price = priceItem->text().toDouble();

int quantity = quantityItem->text().toInt();

totalInventoryValue += price \* quantity;

}

totalSalesValue = getTotalSales();

QMessageBox::information(*this*, "Общий бюджет",

"Общая стоимость инвентаря: " + QString::number(totalInventoryValue, 'f', 2) +

"\nОбщая стоимость продаж: " + QString::number(totalSalesValue, 'f', 2));

}

void Product::**getRemainsSales**()

{

double totalInventoryValue = 0.0;

*//* *Обход* *всех* *строк* *в* *таблице*

*for* (int row = 0; row < ui->tableWidget->rowCount(); ++row)

{

QTableWidgetItem \*priceItem = ui->tableWidget->item(row, 2);

QTableWidgetItem \*quantityItem = ui->tableWidget->item(row, 3);

*if* (!priceItem || !quantityItem) *continue*; *//* *Пропустить* *пустые* *строки*

double price = priceItem->text().toDouble();

int quantity = quantityItem->text().toInt();

totalInventoryValue += price \* quantity;

}

*//* *Показать* *результаты* *в* *диалоговом* *окне*

QMessageBox::information(*this*, "Общий бюджет",

"Общая стоимость инвентаря: " + QString::number(totalInventoryValue, 'f', 2));

}

void Product::**performRevision**()

{

double totalInventoryValue = 0.0;

double totalSalesValue = getTotalSales();

*//* *Расчет* *общей* *стоимости* *всех* *продуктов* *в* *инвентаре*

*for* (int row = 0; row < ui->tableWidget->rowCount(); ++row)

{

QTableWidgetItem \*priceItem = ui->tableWidget->item(row, 2);

QTableWidgetItem \*quantityItem = ui->tableWidget->item(row, 3);

*if* (!priceItem || !quantityItem) *continue*; *//* *Пропустить* *пустые* *строки*

double price = priceItem->text().toDouble();

int quantity = quantityItem->text().toInt();

totalInventoryValue += price \* quantity;

}

*//* *Вычисление* *разницы*

double revisionResult = totalInventoryValue - totalSalesValue;

*//* *Вывод* *результата*

QMessageBox::information(*this*, "Ревизия",

"Результат ревизии:\nОбщий бюджет: " + QString::number(totalInventoryValue, 'f', 2) +

"\nПродано на: " + QString::number(totalSalesValue, 'f', 2) +

"\nРазница: " + QString::number(revisionResult, 'f', 2));

}

void Product::**TotalSalesSum**()

{

double totalSalesValue = 0.0;

totalSalesValue = getTotalSales();

*//* *Показать* *результаты* *в* *диалоговом* *окне*

QMessageBox::information(*this*, "Общий бюджет",

"Общая стоимость продаж: " + QString::number(totalSalesValue, 'f', 2));

}

void Product::**displayProductInfo**()

{

QString skuToFind = ui->skuLineEdit->text();

*if* (skuToFind.isEmpty())

{

QMessageBox::warning(*this*, "Ошибка", "Введите артикул продукта.");

*return*;

}

bool found = *false*;

QString info;

*//* *Поиск* *в* *каждой* *строке* *таблицы*

*for* (int row = 0; row < ui->tableWidget->rowCount(); ++row)

{

QTableWidgetItem \*skuItem = ui->tableWidget->item(row, 1);

*if* (skuItem && skuItem->text() == skuToFind)

{

found = *true*;

*//* *Сбор* *информации* *о* *продукте*

QString name = ui->tableWidget->item(row, 0)->text();

QString price = ui->tableWidget->item(row, 2)->text();

QString quantity = ui->tableWidget->item(row, 3)->text();

double totalPrice = price.toDouble() \* quantity.toInt(); *//* *Расчет* *общей* *стоимости*

info = "Артикул: " + skuToFind + "\nНазвание: " + name +

"\nЦена за единицу: " + price + "\nКоличество: " + quantity +

"\nОбщая стоимость: " + QString::number(totalPrice, 'f', 2);

*break*;

}

}

*if* (!found)

{

QMessageBox::information(*this*, "Результат поиска", "Продукт с артикулом '" + skuToFind + "' не найден.");

}

*else*

{

QMessageBox::information(*this*, "Информация о продукте", info);

}

}

void Product::**on\_addProduct\_clicked**()

{

addProduct();

}

void Product::**on\_deleteLProduct\_clicked**()

{

deleteLProduct();

}

void Product::**on\_sellProduct\_clicked**()

{

sellProduct();

}

void Product::**on\_calculateTotalBudget\_clicked**()

{

calculateTotalBudget();

}

void Product::**on\_sale\_clicked**()

{

TotalSalesSum();

}

void Product::**on\_Remains\_clicked**()

{

getRemainsSales();

}

void Product::**on\_revizion\_clicked**()

{

performRevision();

}

void Product::**on\_Information\_clicked**()

{

displayProductInfo();

}

**Product\_class**

#ifndef PRODUCT\_CLASS\_H

#define PRODUCT\_CLASS\_H

#include <QString>

*class* **Product\_class**

{

*public*:

**Product\_class**();

**Product\_class**(*const* QString &sku, *const* QString &name, double price, int quantity);

QString **getSku**() *const*;

QString **getName**() *const*;

double **getPrice**() *const*;

int **getQuantity**() *const*;

void **setQuantity**(int newQuantity);

*private*:

QString sku;

QString name;

double price;

int quantity;

};

#endif *//* *PRODUCT\_CLASS\_H*

#include "product\_class.h"

Product\_class::**Product\_class**()

{

}

Product\_class::**Product\_class**(*const* QString &sku, *const* QString &name, double price, int quantity)

: sku(sku), name(name), price(price), quantity(quantity) {}

QString Product\_class::**getSku**() *const* {

*return* sku;

}

QString Product\_class::**getName**() *const* {

*return* name;

}

double Product\_class::**getPrice**() *const* {

*return* price;

}

int Product\_class::**getQuantity**() *const* {

*return* quantity;

}

void Product\_class::**setQuantity**(int newQuantity) {

quantity = newQuantity;

}

**productmenedger.cpp**

#ifndef PRODUCTMENEDGER\_H

#define PRODUCTMENEDGER\_H

#include <QList>

#include "product\_class.h"

*class* **ProductMenedger**

{

*public*:

**ProductMenedger**();

void **addProduct**(*const* Product\_class &product);

bool **removeProduct**(*const* QString &sku);

void **sellProduct**(*const* QString &sku);

double **calculateTotal**() *const*;

double **calculateSoldTotal**() *const*;

double **calculateInventoryValue**() *const*;

*private*:

QList<Product\_class> products;

QList<Product\_class> soldProducts;

};

#endif *//* *PRODUCTMENEDGER\_H*

#include "productmenedger.h"

ProductMenedger::**ProductMenedger**()

{

}

void ProductMenedger::**addProduct**(*const* Product\_class &product) {

products.append(product);

}

bool ProductMenedger::**removeProduct**(*const* QString &sku) {

*for* (int i = 0; i < products.size(); ++i) {

*if* (products[i].getSku() == sku) {

products.removeAt(i);

*return* *true*;

}

}

*return* *false*;

}

void ProductMenedger::**sellProduct**(*const* QString &sku) {

*for* (int i = 0; i < products.size(); ++i)

{

*if* (products[i].getSku() == sku && products[i].getQuantity() > 0)

{

*//* *Уменьшаем* *количество* *на* *1*

int newQuantity = products[i].getQuantity() - 1;

products[i].setQuantity(newQuantity);

*//* *Добавляем* *продукт* *в* *список* *проданных*

Product\_class soldProduct = products[i];

soldProduct.setQuantity(1); *//* *Устанавливаем* *количество* *проданного* *товара* *равным* *1* *для* *корректной* *записи* *в* *истории* *продаж*

soldProducts.append(soldProduct);

*//* *Если* *количество* *продукта* *становится* *равным* *нулю,* *удаляем* *его* *из* *списка* *доступных*

*if* (newQuantity == 0)

{

products.removeAt(i);

}

*return*; *//* *Завершаем* *функцию,* *так* *как* *продукт* *уже* *обработан*

}

}

}

double ProductMenedger::**calculateTotal**() *const* {

double total = 0;

*for* (*const* Product\_class &product : products) {

total += product.getPrice() \* product.getQuantity();

}

*return* total;

}

double ProductMenedger::**calculateSoldTotal**() *const* {

double total = 0;

*for* (*const* Product\_class &product : soldProducts) {

total += product.getPrice();

}

*return* total;

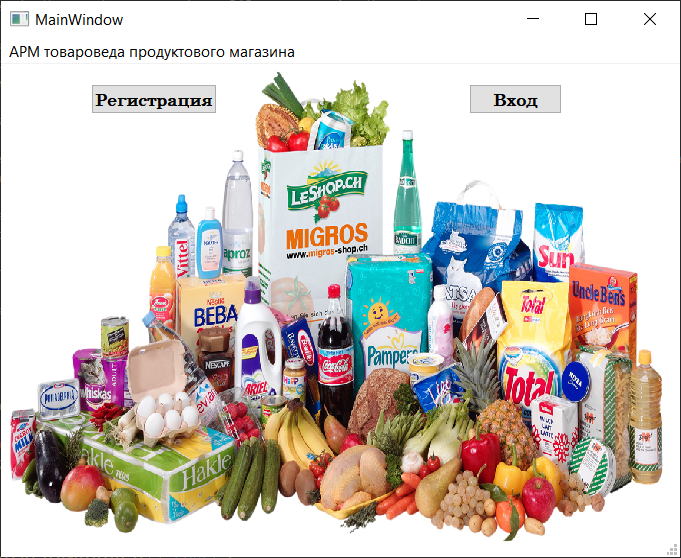
}

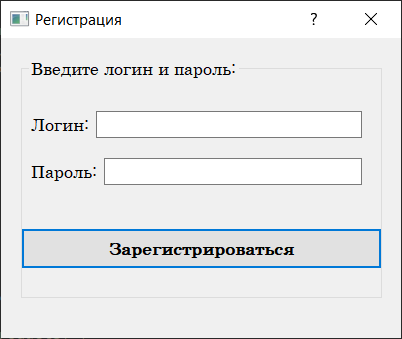
double ProductMenedger::**calculateInventoryValue**() *const* {

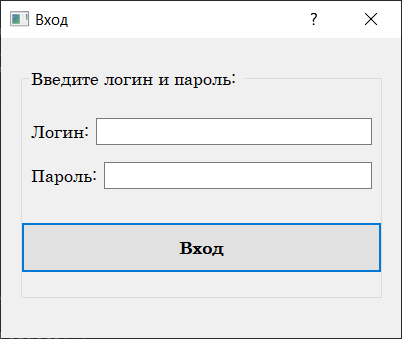
*return* calculateTotal() - calculateSoldTotal();

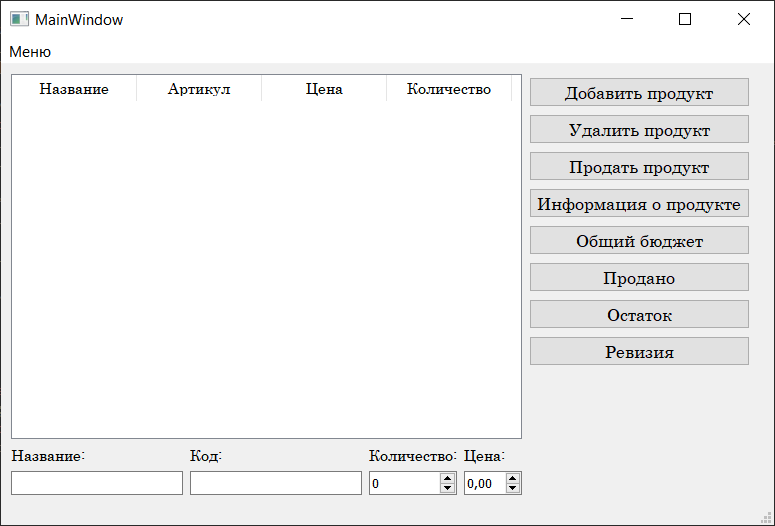
}

**Скриншоты пропущенного алгоритма через движок:**

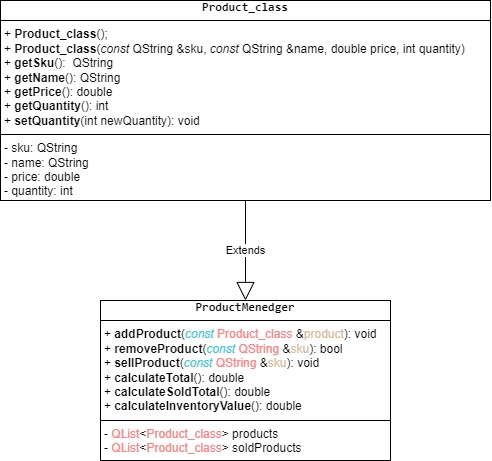
****

****

****

****

**UML-диаграмма:**



**Ссылка на GitHub:**

[**https://github.com/M-Oduvanchick/inf\_2\_sem/blob/main/ARMM.pro.user**](https://github.com/M-Oduvanchick/inf_2_sem/blob/main/ARMM.pro.user)

**Вывод:**

Программа работает корректно, все условия поставленной задачи соблюдены.