

Колледж космического машиностроения и технологий

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Разработка книжного магазина. Клиентская часть»**

Выполнил студент

Митюшин П. А.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королев 2021 г.**

Оглавление

[Введение. 3](#_Toc69924108)

[1 Теоретическая часть 4](#_Toc69924109)

[1.1 Описание предметной области 4](#_Toc69924110)

[1.2 Классификации книжных магазинов. 4](#_Toc69924111)

[1.3 Описание существующих разработок. 5](#_Toc69924112)

[2 Проектная часть 8](#_Toc69924113)

[2.1 Диаграмма прецедентов 8](#_Toc69924114)

[2.2 Выбор инструментов 9](#_Toc69924115)

[2.2.1 Язык программирования 9](#_Toc69924116)

[2.2.2 Среда разработки программного обеспечения 9](#_Toc69924117)

[2.2.3 База данных 10](#_Toc69924118)

[2.3 Проектирование сценария 11](#_Toc69924119)

[2.4 Диаграммы классов 12](#_Toc69924120)

[2.5 Описание главного модуля. 12](#_Toc69924121)

[2.6 Описание спецификаций к модулям 12](#_Toc69924122)

[2.7 Описание модулей 13](#_Toc69924123)

[2.8 Описание тестовых наборов модулей. 18](#_Toc69924124)

[2.9 Описание применения средств отладки 19](#_Toc69924125)

[2.10 Анализ оптимальности использования памяти и быстродействия 19](#_Toc69924126)

[3 Эксплуатационная часть 20](#_Toc69924127)

[3.1 Руководство оператора 20](#_Toc69924128)

[3.1.1 Назначение программы 20](#_Toc69924129)

[3.1.2 Условия выполнения программы 20](#_Toc69924130)

[3.1.3 Выполнение программы 20](#_Toc69924131)

[3.1.4 Сообщение оператору 22](#_Toc69924132)

[3.2 To-Do лист 22](#_Toc69924133)

[Заключение 23](#_Toc69924134)

[Литература 23](#_Toc69924135)

[Приложение 24](#_Toc69924136)

# Введение.

Данный курсовой проект заключается в разработке серверной части (Клиентская часть описана у Слепова Андрея) операционную систему Ubuntu. Приложение представляет собой приложение для составления заказов, которым будет пользоваться кассир. Разработка программы будет происходить на языке программирования C++.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# Теоретическая часть

## Описание предметной области

**Книжный магазин** - это предприятие розничной торговли, реализующее издательскую книжную продукцию. Как правило, книжный магазин имеет торговый зал, где осуществляется реализация продукции (и внутренняя реклама магазина), а также складские помещения для приема и хранения книготоргового ассортимента.

## Классификации книжных магазинов.

**Крупнейшие книжные магазины** - эти магазины действуют на рынке в течение нескольких десятков лет и устойчиво воспринимаются потребителями как основные источники приобретения книг в Москве. Они обеспечивают большой объем товарооборота и имеют крупные торговые площади в удобных и привычных для покупателя местах.

**Самостоятельно действующие (Частные) книжные магазины** - это самая многочисленная группа розничных книготорговых предприятий. Они реализуют основную социальную функцию книжной торговли доступ покупателей к достаточно широкому ассортименту книжной продукции, в удобном для покупателей месте и в удобное для них время.

**Магазины издательств** — это магазины которые формируют ассортимент книг из «своего» издательства и «чужих» изданий сходной тематики, приобретенных у других издательств (путем обмена или обычной купли-продажи).

**Магазины религиозной литературы** - это немногочисленная группа магазинов, расположенных обычно при действующих храмах или вблизи от них. В таких магазинах предлагается, как правило, не очень широкий ассортимент изданий религиозной (преимущественно православной) тематики.

**Магазины школьно-письменных принадлежностей**, в которых представлен достаточно широкий ассортимент учебников и учебно-методической литературы. Эти предприятия не являются в полном смысле слова книжными магазинами, но обеспечивают населению доступ к книгам и в целом отвечают критериям, перечисленным выше.

## Описание существующих разработок.

На данный момент существует уже много разработанных приложений на данную тематику. Почти все поддерживают основной функционал:

* фиксируют каждую денежную сделку;
* печатают чек;
* хранят информации о количество проданных товаров и услуг по ранее введенному коду;
* выдают информацию о денежной сумме, которая должна находиться в кассе;
* хранят всю информацию о продажах на бумажной ленте и у себя на носителе памяти.

Примеры приложений:

* 1С:Касса

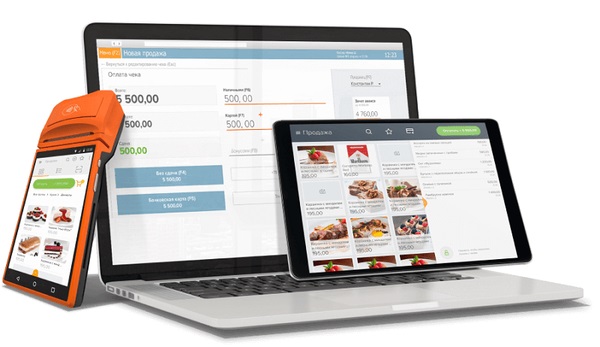
1С:Касса – это облачный сервис, который устанавливается на автономный кассовый аппарат.



Рис1. облачный сервис 1С:Касса.

* Бизнес.ру

Бизнес.Ру – это кассовая программа, которая подходит для любой ККТ и устанавливается на любое устройство – от смартфона до ПК – с ОС Windows или Andriod. Работает на базе облачного сервиса



* Рис2. кассовая программа Бизнес.ру
* Мой Склад

МойСклад – это программа для ведения розничной торговли. Для работы она устанавливается на любой ПК, к которому подключается фискальный регистратор.

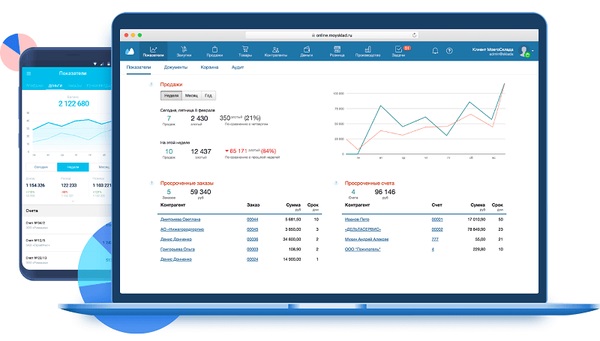


Рис3. кассовая программа Мой Склад

# Проектная часть

## Диаграмма прецедентов

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения.

Кассир работает за кассой, составляет отчёт о прибыли за день/месяц.

Бухгалтер рассчитывет заработную плату для сотрудников, прибыль за квартал и за год.

Системный администратор устанавливает программы, настраивает их и исправляет возникшие ошибки.

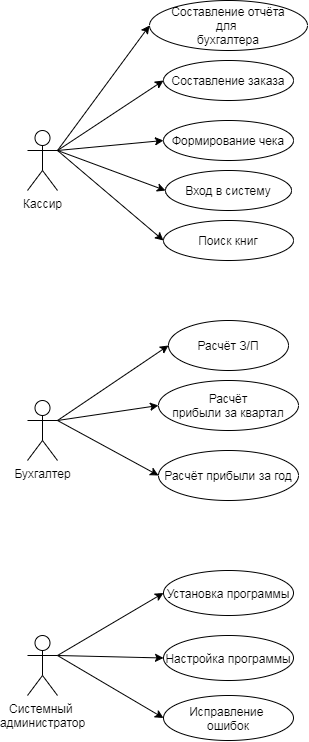


Рис.1 Диаграмма прецедентов.

## Выбор инструментов

### Язык программирования

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий/Язык | Java | C++ | C# |
| Участие в корпоративном проекте | 10 | 7 | 8 |
| Простота сопровождения | 8 | 7 | 8 |
| Наличие библиотек | 10 | 5 | 8 |
| Наличие документации на рускком языке | 7 | 7 | 7 |
| Скорость разработки | 8 | 7 | 8 |
| Итого баллов | 43 | 33 | 39 |

Таблица 1. Выбор языка по критериям.

По результатам сравнения выиграл язык программирования Java, но я выберу язык программирования C++ т.к. я хочу разработать десктоп приложение, используя QT.

В разработке приложений под Ubuntu используются в основном языки программирования Си или C++. Данный курсовой проект написан на C++.

C++ - компилируемый статически типизированный язык программирования. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C. Как в Си и в C++, точкой входа в программу является функция main.

### Среда разработки программного обеспечения

**Qt** — кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++.

Есть также «привязки» ко многим другим языкам программирования: Python - PyQt, PySide; Ruby - QtRuby; Java - Qt Jambi; PHP - PHP-Qt и другие.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML

Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы

Средой разработки является [Qt Creator](https://ru.wikipedia.org/wiki/Qt_Creator), которая включает [редактор кода](https://ru.wikipedia.org/wiki/Редактор_кода), справку, графические средства Qt Designer и возможность отладки приложений. Qt Creator может использовать [GCC](https://ru.wikipedia.org/wiki/GCC) или Microsoft VC++ в качестве компилятора и [GDB](https://ru.wikipedia.org/wiki/GDB) в качестве отладчика. Для Windows-версий библиотека комплектуется компилятором, заголовочными и объектными файлами [MinGW](https://ru.wikipedia.org/wiki/MinGW).

### База данных

В разработке приложений под Ubuntu в основном в качестве базы данных используется **SQLite**. Причины использования:

1. **Минимальные затраты ресурсов.** Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера: база данных SQLite представляет собой обычный файл. Когда БД не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.
2. **Оптимизация для одного пользователя.** С базой данных взаимодействует только наше приложение. Поэтому можно обойтись без идентификации с именем пользователя и паролем.
3. **Надежность и быстрота.** Базы данных SQLite поддерживают транзакции баз данных. Кроме того, операции чтения и записи данных реализуются на оптимизированном коде С.

## **Проектирование сценария**

В данном разделе приведен сценарий использования программы пользователем.

При запуске приложения открывается экран для ввода логина и пароля. После ввода логина и пароля кассира программа проверяет наличие файла order.txt, если файл найден открывается окно с таблицей и двумя кнопками при нажатии на кнопку производится проверка заказа (в заказе может быть больше книг, чем на складе) и выводится в таблицу.

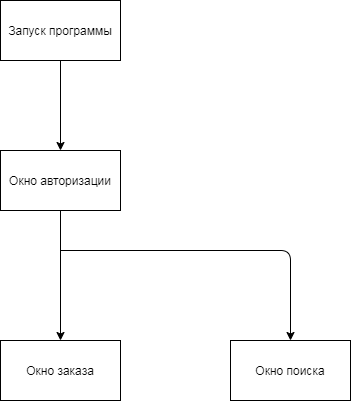


Рис2. Сценарий использования программы.

## Диаграммы классов

В данном разделе предоставлены диаграммы классов (Рисунок 3).

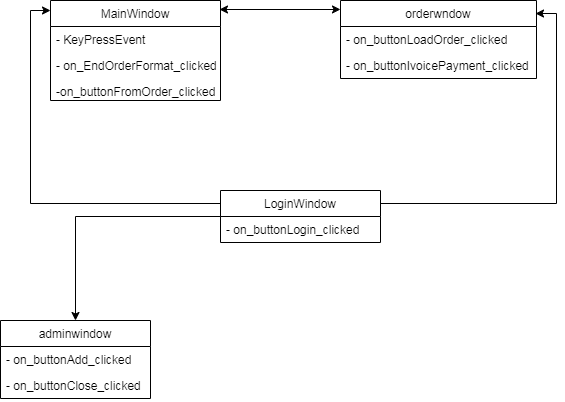


Рис.3 Диаграмма классов.

## Описание главного модуля.

Главный модуль состоит из функции main. Он выполняется при запуске приложения.

**Листинг 1. Главный модуль.**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

w.startMainWindow();

return a.exec();

}

## Описание спецификаций к модулям

Всего разработано 4 модуля:

* Окно для авторизации сотрудников (loginwindow)

//Считывание данных из заказа для формирования счёта (orderwindow)

* Окно формирования счёта
* Поиск книг в базе данных для формирования счёта (mainwindow)
* Окно админа для редактирования базы данных (adminwindow)

## Описание модулей

**Окно для авторизации сотрудников** – позволяет ввести данные для входа в систему. На экране находится поле для ввода логина и пароля сотрудника.

**Окно формирования счёта –** позволяет загрузить данные из заказа и провести проверку достаточно ли книг на складе (в заказе может быть запрос на большее количество книг, чем есть на складе). Окно имеет две кнопки и таблицу куда загружается книги из заказа, цену и количество.

**Поиск книг в базе данных для формирования счёта –** позволяет кассиру сформировать счёт для оплаты. Окно имеет поле для ввода названия книги, таблицу, где будет отображаться книга и кнопку для формирования счёта.

**Окно админа для редактирования базы данных –** Позволяет редактировать базу данных изменяя в ней значения, добавления и удаления данных**.** Окно имееттаблицу, содержащую базу данных и кнопки для её редактирования.

**Листинг 2. Нахождение книг из заказа и запись из в таблицу**

void orderwindow::on\_buttonLoadOrder\_clicked()

{

QString id\_lib, title\_lib, author\_lib, price\_lib, count\_lib;

QStandardItem \*item\_id, \*item\_title, \*item\_author, \*item\_price, \*item\_count;

QString path = \_\_FILE\_\_;

db\_lib\_order.setDatabaseName(formPath(path) + "/lib.db");

db\_lib\_order.open();

QSqlQuery lib(db\_lib\_order);

lib.exec("SELECT book\_id, title, author, price, count FROM books");

std::string line;

int open\_search\_window = 0;

for (int table\_row = 0; lib.next();)

{

id\_lib = lib.value(0).toString(); title\_lib = lib.value(1).toString(); author\_lib = lib.value(2).toString();

price\_lib = lib.value(3).toString(); count\_lib = lib.value(4).toString();

std::fstream order\_file("order.txt");

while (getline(order\_file, line))

{

QString str = QString::fromStdString(line);

if (line == "order has been paid")

{

QMessageBox msgBox;

msgBox.setWindowTitle("Message");

msgBox.setText("The order was marked as already paid. Please put a new file with the order.");

msgBox.exec();

this->close();

MainWindow \*mainwindow;

mainwindow = new MainWindow(this);

mainwindow->show();

open\_search\_window = 1;

break;

}

else if (str.indexOf(title\_lib) == 0 && (std::stoi(count\_lib.toStdString()) - std::stoi(priceString(line))) >= 0)

{

item\_id = new QStandardItem(id\_lib);

item\_title = new QStandardItem(title\_lib);

item\_author = new QStandardItem(author\_lib);

item\_price = new QStandardItem(price\_lib);

item\_count = new QStandardItem(count\_lib);

model\_order\_window->setItem(table\_row, 0, item\_id);

model\_order\_window->setItem(table\_row, 1, item\_title);

model\_order\_window->setItem(table\_row, 2, item\_author);

model\_order\_window->setItem(table\_row, 3, item\_price);

model\_order\_window->setItem(table\_row, 4, item\_count);

massTitles[table\_row] = title\_lib;

massSubDb.push\_back(std::stoi(count\_lib.toStdString()));

massSubOrder.push\_back(std::stoi(priceString(line)));

++table\_row;

break;

}

else if (std::stoi(count\_lib.toStdString()) - std::stoi(priceString(line)) < 0)

{

QMessageBox msgBox;

msgBox.setWindowTitle("Message");

msgBox.setText("The number of books in the order is more than in stock: " + title\_lib);

msgBox.exec();

}

}

if (open\_search\_window == 1)

break;

order\_file.close();

}

model\_order\_window->setHeaderData(0, Qt::Horizontal, "book id", Qt::DisplayRole);

model\_order\_window->setHeaderData(1, Qt::Horizontal, "title book", Qt::DisplayRole);

model\_order\_window->setHeaderData(2, Qt::Horizontal, "author", Qt::DisplayRole);

model\_order\_window->setHeaderData(3, Qt::Horizontal, "price", Qt::DisplayRole);

model\_order\_window->setHeaderData(4, Qt::Horizontal, "count books", Qt::DisplayRole);

ui->orderTableView->setModel(model\_order\_window);

ui->orderTableView->resizeRowsToContents();

ui->orderTableView->resizeColumnsToContents();

}

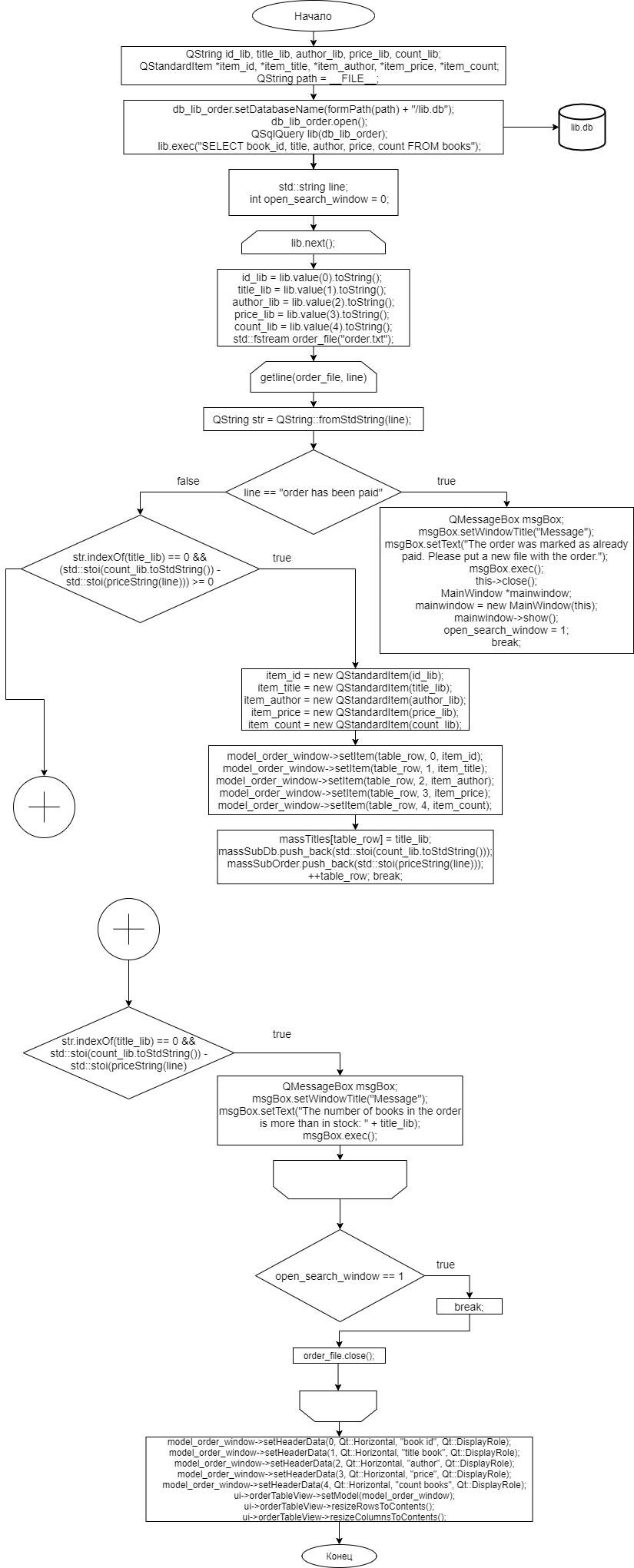


Рис. 4. Блок-схема. on\_buttonLoadOrder\_clicked.

## Описание тестовых наборов модулей.

В некоторых модулях производится проверка данных на корректность, чтобы сообщить пользователю о проблеме и предотвратить аварийное завершение или некорректную работу программы.

**Тест 1. Проверка на наличие файла заказа**

Если файл заказа не был положен в папку build-Book\_shop –Desktop-Release, то программа выведет сообщение, что файл с заказом не был найден (Рис. 5).

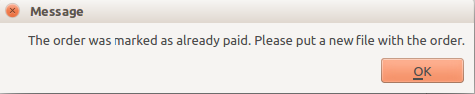


Рис.5. Файл с содержанием заказа не был найден

**Тест 2. Проверка не был ли ранее оплачен заказ**

Если кассир выполнил заказ и забыл, что он был у него оплачен, то программа выведет сообщение и не оплатит заказ (Рис. 6).

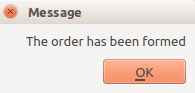


Рис.6. Сообщение о том, что заказ был оплачен

**Тест 3. Проверка правильный ли логин и пароль ввёл сотрудник**

Если сотрудник ввёл неправильный логин и пароль, то ему будет выведено сообщение о том, что логин или пароль был введён некорректно.

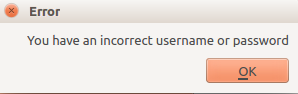


Рис.7.Неправильный логин или пароль

## Описание применения средств отладки

В ходе написания курсового проекта при попытке запустить программу были получены ошибки.

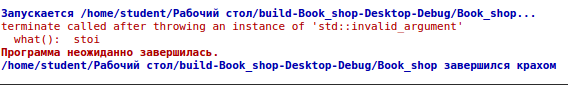


Рис.8. Сообщение об ошибке

При проверке кода были исправлены найденные ошибки, в результате при запуске программы ошибок не было C:\Users\P1-18\Desktop\ТЕСТЫ\14.png

Рис.9. Успешная сборка

## Анализ оптимальности использования памяти и быстродействия

В дальнейшем планируется оптимизировать код. При анализе оптимальности использования памяти, утечки памяти не были обнаружены.

# Эксплуатационная часть

## **Руководство оператора**

### **Назначение программы**

Разработанное приложение позволяет сформировать счёт и хранить его в формате текстового файла

### **Условия выполнения программы**

Операционная система: Ubuntu

Оперативная память: 512 Мб.

Доступного места на диске: 200 Мб.

Путь к базе данных должен задаваться в программе вручную

Файл заказа должен быть текстовиком и должен лежать в папке build-Project-DesktopRelease

### **Выполнение программы**

При запуске программы откроется окно для авторизации сотрудника в котором надо ввести логин и пароль для входа под учётной записью.

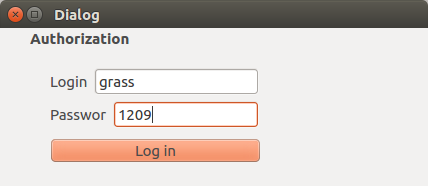


Рис1. Окно авторизации

При вводе логина и пароля кассира, откроется окно заказов если файл заказа в нужной папке

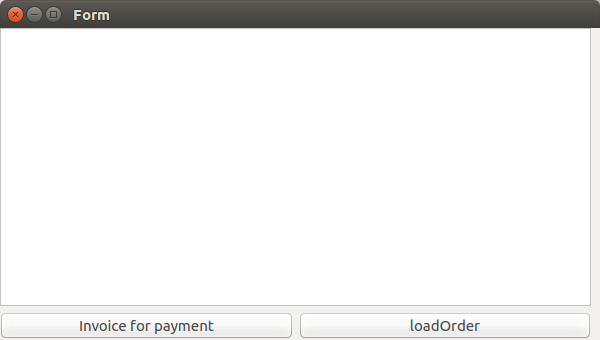


Рис.2 Окно формирования заказа

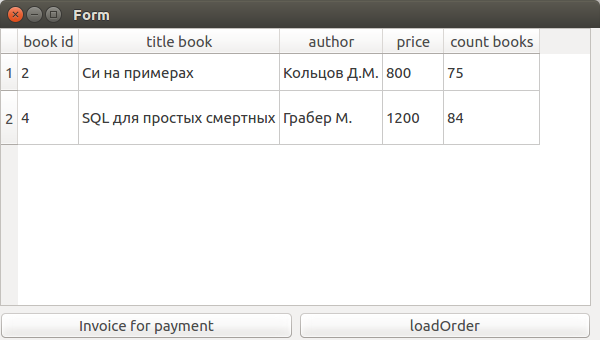
При нажатии на кнопку loadOrder в таблицу загружается содержание заказа.

Рис.3 Содержание заказа

Затем нажимаем на кнопку Invoice for payment для того, чтобы сформировать конечный счёт.

Если файл заказа не был найден, то откроется окно поиска книг в базе данных благодаря которому можно будет сформировать счёт на основе искомых книг.

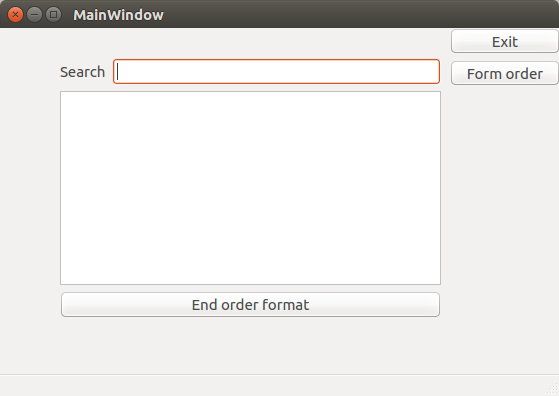


Рис4. Окно поиска

Если вы ходите сформировать заказ в программе но он при нажатии на кнопку Form order выводит диалоговое окно которое содержит сообщение: “You did not put the order file in the folder”.

### Сообщение оператору

**Причина:** файл заказа не был добавлен папку build-Project-DesktopRelease.

**Действия программы:** Программа выводит сообщение “You did not put the order file in the folder”

**Действия оператора**: Положить файл с заказом с именем order.txt

Так же из окна поиска можно открыть окно заказа нажав на кнопку Form order.

## To-Do лист

* Улучшить дизайн программы
* Доработать окно администратора
* В окне поиска при добавлении нескольких одинаковых книг создавался бы ещё один столбец который бы считал количество одинаково добавленных книг

# Заключение

Данный курсовой проект был разработан в целях автоматизации формирования счёта. Данный проект может быть использован в учебных целях для обучения программированию на qt.

# Литература

Язык программирования

https://ru.wikipedia.org/wiki/C%2B%2B

Среда разработки программного обеспечения

# Приложение

Приложение 1. Листинг 1. main.cpp

#include "mainwindow.h"

#include "loginwindow.h"

//Выполнил: Андрей Слепов и Митюшин Пётр

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

LoginWindow login\_window;

login\_window.open();

return a.exec();

}

Приложение 2. Листинг 2. mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "loginwindow.h"

#include <QTime>

#include <QDate>

#include <fstream>

#include <QDebug>

#include <string>

//Выполнил: Андрей Слепов и Митюшин Пётр

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

orderWindow = new orderwindow;

}

MainWindow::~MainWindow()

{

delete ui;

}

//Примечание разработчика сформировать константу пути и прибавлять в //дальнейшем название бд

QString formPath(QString path) //Формирование пути для того, чтобы //определить, где лежит база данных

{

int index\_slesh;

QString new\_path;

for (index\_slesh = path.length()-1; path[index\_slesh] != '/'; --index\_slesh) { // избавляемся от названия файла

;

}

for (int index\_new\_path = 0; index\_new\_path != index\_slesh; ++index\_new\_path) { //формируем строку без названия файла

new\_path += path[index\_new\_path];

}

return new\_path;

}

//Таблица вне функции, чтобы хранить две переменные

QStandardItemModel \*model = new QStandardItemModel;

//Connect database with books

QSqlDatabase db\_lib = QSqlDatabase::addDatabase("QSQLITE", "lib");

void MainWindow::keyPressEvent(QKeyEvent \*event) //Метод при нажатии кнопки //на клавиатуре читает данные из бд в таблицу

{

QString book\_id, title, author, price, count;

QStandardItem \*item\_id, \*item\_title, \*item\_author, \*item\_price, \*item\_count;

//Осуществляем запрос

QString path = \_\_FILE\_\_; //Местонахождение файла

db\_lib.setDatabaseName(formPath(path) + "/lib.db");

db\_lib.open();

QSqlQuery query(db\_lib);

query.exec("SELECT book\_id, title, author, price, count FROM books");

if (event->key() == Qt::Key\_Enter || event->key() == Qt::Key\_Return) //Если кнопка Enter была нажата

{

bool book\_not\_found = 0; //Переменная для проверки была ли книга //найдена

for (int table\_row = 0; query.next();) //Осуществляем проход по всей //базе данных

{

book\_id = query.value(0).toString(); //id книги из базы данных

title = query.value(1).toString(); //Название книги из базы //данных

author = query.value(2).toString(); //Автор книги из базы данных

price = query.value(3).toString(); //Цена книги из базы данных

count = query.value(4).toString(); //Количество книги из базы //данных

if (title == ui->lineEdit->text()) { //Если название из базы //данных совпадает с текством введённым в поле для ввода

//Отделения данных от представлений в виджетах, //обрабатывающих наборы данных

item\_id = new QStandardItem(book\_id); item\_title = new QStandardItem(title); item\_author = new QStandardItem(author);

item\_price = new QStandardItem(price); item\_count = new QStandardItem(count);

//Заполнение модели элементами по индексу

model->setItem(table\_row, 0, item\_id); model->setItem(table\_row, 1, item\_title); model->setItem(table\_row, 2, item\_author);

model->setItem(table\_row, 3, item\_price); model->setItem(table\_row, 4, item\_count);

book\_not\_found = 1; //Книга найдена

++table\_row; //строка таблицы

}

}

if (book\_not\_found == 0) { //Если книга не найдена выводим окно //сообщения

QMessageBox msgBox;

msgBox.setWindowTitle("Message"); //Заголовок сообщения

msgBox.setText("sorry book not found");//Содержание сообщения

msgBox.exec();

}

}

ui->searchTableView->setModel(model); //Установка модели в которая была //ранее заполнина

//Обновление размера таблицы

ui->searchTableView->resizeRowsToContents();

ui->searchTableView->resizeColumnsToContents();

}

void MainWindow::on\_EndOrderFormat\_clicked() //Метод формирования заказа

{

std::ofstream file("check.txt");

if (file.is\_open()) //Если файл открыт

{

LoginWindow lw;

file << "OOO \"Books shop\"\n";

QDate cd = QDate::currentDate();

QString date\_in\_file = cd.toString(); //Текущая дата

QTime ct = QTime::currentTime();

QString time\_in\_file = ct.toString(); //Текущее время

file << "Terminal: 00001\n"; //starting write in file

file << "Cashier: " << lw.cashiers\_replace.toStdString() << " " << date\_in\_file.toStdString() << " " << time\_in\_file.toStdString() << "\n";

QModelIndex index;

int summ\_order = 0, tabel\_column;

for (tabel\_column = 0; tabel\_column < model->rowCount(); ++tabel\_column)

{

index = model->index(tabel\_column, 1); // Индекс из модели. //Индекс названия книги

file << index.data().toString().toStdString() << "..............."; // Записываем название книги в счёт. в txt файл

index = model->index(tabel\_column, 3); // Индекс из модели. //Индекс стоимости книги

file << index.data().toString().toStdString(); // Записываем цену //в счёт

summ\_order += std::stoi(index.data().toString().toStdString()); // Подсчёт суммы стоимости всех книг

file << "\n";

}

file << "NDS 10/110\n";

file << "Count product in receipt: " << tabel\_column << "\n"; //Количество всех книг из модели

file << "Summ: " << summ\_order << "\n";

file << "Summ NDS 10/100: " << (summ\_order \* 20) / 100; //Подсчёт НДС //от общей суммы

file.close();

}

else //Если файл не открыт

{

QMessageBox errEntryBox;

errEntryBox.setWindowTitle("Error"); //Заголовок сообщения

errEntryBox.setText("File not open"); //Содержание сообщения

errEntryBox.exec(); //отображает диалоговое окно

}

}

void MainWindow::on\_buttonFormOrder\_clicked() //Метод открытия окна заказа

{

std::fstream order\_file("order.txt");

if (order\_file.is\_open()) { //Если файл открыт

this->close(); //Закрываем текущее окно

orderWindow->show(); //Открываем окно формирования заказа

} else {

QMessageBox errEntryBox;

errEntryBox.setWindowTitle("Message"); //Заголово сообщения

errEntryBox.setText("You did not put the order file in the folder"); //Содержание сообщения

errEntryBox.exec(); //отображает диалоговое окно

}

}

Приложение 3. Листинг 3. orderwindow.cpp

#include "mainwindow.h"

#include "orderwindow.h"

#include "ui\_orderwindow.h"

#include <QStandardItemModel>

#include <QMessageBox>

#include <fstream>

#include <QTime>

#include <QDate>

#include <string>

//Выполнил: Митюшин Пётр

orderwindow::orderwindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::orderwindow)

{

ui->setupUi(this);

}

orderwindow::~orderwindow()

{

delete ui;

}

//Протестируй функцию

std::string priceString(std::string line) //Функция для того чтобы вернуть //фрагмент строки содержащий количество книг

{

unsigned int ind\_start\_price;

for (ind\_start\_price = line.length()-1; line[ind\_start\_price] != '|'; --ind\_start\_price) //нахождения последней границы из текстового файла заказа

;

std::string result\_str;

for (int ind\_form\_string = ind\_start\_price+1; line[ind\_form\_string] != '\0'; ++ind\_form\_string) {

result\_str += line[ind\_form\_string];

}

return result\_str;

}

const int sizeMassTitles = 1000;

QString \*massTitles = new QString[sizeMassTitles];

std::vector <int> massSubOrder, massSubDb;

QStandardItemModel \*model\_order\_window = new QStandardItemModel;

QSqlDatabase db\_lib\_order = QSqlDatabase::addDatabase("QSQLITE", "lib\_order"); //Активизация драйвера для использования sqlite

void orderwindow::on\_buttonInvoicePayment\_clicked() //При нажатии на кнопку //формируется счёт

{

std::ofstream order\_file("order.txt", std::ios::app); //Открываем файл //для добавления в конец строки, что заказ оплачен

order\_file << "order has been paid"; //Строка о том что заказ был оплачен

order\_file.close();

QDate cd = QDate::currentDate();

QString date\_in\_file = cd.toString(); //Текущая дата

QTime ct = QTime::currentTime();

QString time\_in\_file = ct.toString(); //Текущие время

std::fstream final\_order("final\_order.txt");

int summ\_for\_score = 0, account\_number = 0;

QString path = \_\_FILE\_\_; //Местонахождение текущего файла

db\_lib\_order.setDatabaseName(formPath(path) + "/lib.db"); //Путь до //проекта соединёный с названием базы данных

db\_lib\_order.open();

QSqlQuery lib\_num(db\_lib\_order);

final\_order << "Provider: " << "\n";

final\_order << "Invoice for payment: " << account\_number << '\n';

final\_order << "Number\t\tProducts\tPrice\n\n"; //symbyl number of order

QModelIndex index;

for (int number\_order = 0; number\_order < model\_order\_window->rowCount(); ++number\_order)

{

final\_order << number\_order << ". ";

index = model\_order\_window->index(number\_order, 1); //Индекс названия //книги

final\_order << index.data().toString().toStdString() << "..................";

index = model\_order\_window->index(number\_order, 3); //Индекс цены

//Записываем в файл коичество из заказа и количество умножаем на цену

final\_order << massSubDb[number\_order] << '\*' << index.data().toString().toStdString() << '=' << massSubDb[number\_order] \* std::stoi(index.data().toString().toStdString()) << '\n';

//Сумма всего заказа

summ\_for\_score += massSubDb[number\_order] \* std::stoi(index.data().toString().toStdString());//Умножаем количество на //цену

lib\_num.prepare("UPDATE books set count = :count WHERE title = :title"); //Находим книгу по названию и меняем её количество в базе данных

lib\_num.bindValue(":count", massSubDb[number\_order] - massSubOrder[number\_order]); //То чему будет ровно количество

lib\_num.bindValue(":title", massTitles[number\_order]); //То, откуда //мы берём название книги для её поиска

lib\_num.exec(); //Выполняя запрос из prepare

}

final\_order << "\nTotal: " << summ\_for\_score << '\n' << "Total vat amount: " << (summ\_for\_score \* 20)/100 + summ\_for\_score << '\n';

final\_order << date\_in\_file.toStdString() << ' ' << time\_in\_file.toStdString() << '\n'; //Сегодняшняя дата и время

final\_order.close(); //закрываем файл счёта

QMessageBox msgBox;

msgBox.setWindowTitle("Message"); //Заголовок сообщения

msgBox.setText("The order has been formed"); //Содержание сообщения

msgBox.exec(); //отображает диалоговое окно

this->close(); // Закрываем текущее окно

}

void orderwindow::on\_buttonLoadOrder\_clicked() //При нажатии на кнопку //происходит запись в таблицу книг из заказа

{

QString id\_lib, title\_lib, author\_lib, price\_lib, count\_lib;

QStandardItem \*item\_id, \*item\_title, \*item\_author, \*item\_price, \*item\_count;

//Осуществляем запрос

QString path = \_\_FILE\_\_; //Местонахождение файла

db\_lib\_order.setDatabaseName(formPath(path) + "/lib.db"); //путь до базы //данных

db\_lib\_order.open();

QSqlQuery lib(db\_lib\_order);

lib.exec("SELECT book\_id, title, author, price, count FROM books"); //Осуществляем sql запрос

std::string line;

int open\_search\_window = 0;

for (int table\_row = 0; lib.next();)

{

//Читаем данные из базы данных

id\_lib = lib.value(0).toString(); title\_lib = lib.value(1).toString(); author\_lib = lib.value(2).toString();

price\_lib = lib.value(3).toString(); count\_lib = lib.value(4).toString();

std::fstream order\_file("order.txt"); //Открытие файла, чтобы начать //с первой строки

while (getline(order\_file, line)) //Построчное чтение из файла

{

QString str = QString::fromStdString(line); //переводим строку //типа string в строку типа QString

if (line == "order has been paid") //Если в файле который //содержит заказ встречается строчка, что заказ был оплачен

{

QMessageBox msgBox;

msgBox.setWindowTitle("Message"); //Заголовок сообщения

msgBox.setText("The order was marked as already paid. Please put a new file with the order."); //Содержание сообщения

msgBox.exec(); //отображает диалоговое окно

this->close(); //Закрывает текущее окно

MainWindow \*mainwindow;

mainwindow = new MainWindow(this);

mainwindow->show(); // Открываем окно поиска

open\_search\_window = 1;

break;

}

//Ищет подстроку в строке и определяет разницу между количеством //в базе данных и количеством в заказе

else if (str.indexOf(title\_lib) == 0 && (std::stoi(count\_lib.toStdString()) - std::stoi(priceString(line))) >= 0)

{

item\_id = new QStandardItem(id\_lib);

item\_title = new QStandardItem(title\_lib);

item\_author = new QStandardItem(author\_lib);

item\_price = new QStandardItem(price\_lib);

item\_count = new QStandardItem(count\_lib);

//Заполнение модели элементами по индексу

model\_order\_window->setItem(table\_row, 0, item\_id);

model\_order\_window->setItem(table\_row, 1, item\_title);

model\_order\_window->setItem(table\_row, 2, item\_author);

model\_order\_window->setItem(table\_row, 3, item\_price);

model\_order\_window->setItem(table\_row, 4, item\_count);

massTitles[table\_row] = title\_lib; //Название таблицы //добавляется в массив

massSubDb.push\_back(std::stoi(count\_lib.toStdString())); //Количество книг из базы данных добавляется в конец массив

massSubOrder.push\_back(std::stoi(priceString(line))); //Количество книг из заказа добавляется в конец массив

++table\_row; //строка таблицы

break;

}

else if (str.indexOf(title\_lib) == 0 && std::stoi(count\_lib.toStdString()) - std::stoi(priceString(line)) < 0) //Если //разность количества книг в заказе и в базе данных меньше нуля

{

//выводим сообщение, что в бд количества книг недостаточно

QMessageBox msgBox;

msgBox.setWindowTitle("Message"); //Заголовок сообщения

msgBox.setText("The number of books in the order is more than in stock: " + title\_lib);

msgBox.exec();

}

}

if (open\_search\_window == 1)

break;

order\_file.close();

}

// Название столбцов для tableView

model\_order\_window->setHeaderData(0, Qt::Horizontal, "book id", Qt::DisplayRole);

model\_order\_window->setHeaderData(1, Qt::Horizontal, "title book", Qt::DisplayRole);

model\_order\_window->setHeaderData(2, Qt::Horizontal, "author", Qt::DisplayRole);

model\_order\_window->setHeaderData(3, Qt::Horizontal, "price", Qt::DisplayRole);

model\_order\_window->setHeaderData(4, Qt::Horizontal, "count books", Qt::DisplayRole);

ui->orderTableView->setModel(model\_order\_window);

//Обновление размера таблицы

ui->orderTableView->resizeRowsToContents();

ui->orderTableView->resizeColumnsToContents();

}