# Аннотация

В выпускной квалификационной работе рассматривается программное обеспечение для работы с устройством, взаимодействующим с RFID-метками

Целью данной работы является разработка программного обеспечения для работы с УРМ. RFID-метка – бесконтактное устройство, позволяющее хранить некоторый объем информации, соответствующей описываемому предмету.

Выпускная квалификационная работа по теме «Программное обеспечение по обработке данных с устройства, взаимодействующего с RFID-метками» состоит из пояснительной записки на 65 страницу текстового документа, содержащей 4 раздела, 33 рисунка, 8 таблиц, заключение, список используемых сокращений, списка используемых источников, включающего 8 наименований, и два приложения.

Ключевые слова: RFID, RFID-считыватель, Программное обеспечение, база данных.

# Реферат

В аналитической части рассматриваются способы автоматизации процесса инвентаризации, обзор используемых технологий хранения информации и аналоги программного обеспечения на рынке.

В проектно-расчётной части описаны разработка программного обеспечения, спецификация специальных команд, основной алгоритм работы программного обеспечения, используемая база данных, разработанные функции.

В экспериментальной части демонстрируется тестирование разработанного программного обеспечения с программно-аппаратного устройством.

В экономической части дано технико-экономическое обоснование выбранного способа инвентаризации, описаны используемые технологии, показан сетевой график, дана оценка продолжительности работ и расчёт затрат на их проведение, приведён расчёт экономической эффективности и срока окупаемости, выполненных работ.

|  |  |
| --- | --- |
| Перечень сокращений | 7 |
| Введение | 8 |
| 1. Аналитический обзор | 9 |
| 1.1. Автоматизация процесса инвентаризации | 9 |
| 1.2. Описание характеристик, разрабатываемого ПО | 11 |
| 1.3. Сравнительный анализ имеющихся на рынке программных обеспечений | 11 |
| 1.4. Используемый язык программирования и среда разработки | 12 |
| 2. Проектно-расчётная часть | 13 |
| 2.1. Схема взаимодействия пользователя с ПО и УРМ | 13 |
| 2.2. Протокол управления между УРМ и ПК | 14 |
| 2.3. Структура базы данных | 16 |
| 2.4. Описание средств, используемых в разработке ПО | 17 |
| 2.5. Обзор кода программного обеспечения | 18 |
| 2.6. Описание функционала ПО | 46 |
| 3. Экспериментальная часть | 55 |
| 3.1. Тестирование работы ПО с УРМ в условиях внедрения | 55 |
| 4. Экономическая часть | 57 |
| 4.1. Технико-экономическое обоснование | 57 |
| 4.2. Аналитическая записка | 58 |
| 4.3. Сетевой график хода выполнения работ | 58 |
| 4.4. Расчёт затрат на проведение работ и составление сметы расходов | 60 |
| 4.5. Расчёт экономической эффективности и срока окупаемости проведённых работ | 61 |
| Заключение | 63 |
| Список используемых источников | 64 |
| Приложения | 65 |

# Оглавление

# Перечень сокращений

УРМ – программно-аппаратного устройство для работы с RFID-метками.

ПК – персональный компьютер

ПО – программное обеспечение

БД – база данных

РТУ МИРЭА – Российский технический университет московский институт радиотехники, электроники и автоматики.

RFID – Радиочастотная идентификация (Radio Frequency Identification).

CSV – Comma-Separated Values – значения, разделённые запятыми.

ЭВМ – Электронно-вычислительная машина.

# Введение

Инвентаризация относится к эффективным методам наблюдения и контроля за сохранностью имущества в филиале РТУ МИРЭА г. Фрязино. Для сокращения времени проведения инвентаризации и обеспечения достоверности данных по результатам, в организации было принято решение автоматизировать данный процесс с применением технологии RFID, так как она обладает рядом преимуществом по сравнению с методами, применяющими штрих-коды и QR-коды:

* Высокая скорость считывания и передачи данных.
* Быстрый поиск меток без прямой видимости.
* Безопасность и конфиденциальность сведений.
* Снижение влияния человеческого фактора.
* Устойчивость к агрессивным средам.

В процессе оптимизации данного процесса было разработано новое программно-аппаратное устройство (УРМ), работающее на частоте 13,56 МГц и взаимодействующее с метками семейства Mifare Classic 1K. Это устройство работает в двух режимах:

- в автономном оно может считывать данные с меток и выводить на экран;

- при подключении к персональному компьютеру (ПК) оно осуществляет передачу данных (записывает и считывает данные с метки).

Для взаимодействия УРМ с ПК потребовалось разработка программного обеспечения (ПО) со следующими характеристиками:

* ПО должно быть написанное на языке C++ в среде разработки Qt Creator и иметь графический интерфейс;
* ПО обеспечивает обмен информацией с УРМ через интерфейс RS-232;
* ПО взаимодействует с базой данных, в которой хранится информация об устройствах, имеющих наклеенные RFID-метки.

В этой работе даётся описание созданного программного обеспечения, удовлетворяющего выше описанным характеристикам.

Во время внедрения УРМ в лабораторном корпусе филиала РТУ МИРЭА в г. Фрязино, проводилась проверка работоспособности ПО, которая показала положительные результаты. На основании этого началась реализация процесса автоматизации инвентаризации, а именно наклеивание RFID-меток на всё оборудование лабораторного корпуса и формирование единой базы данных, состоящей из записей, соответствующих оборудованию, оснащённому RFID-метками.

В экономическом расчёте проведено сравнение созданного ПО с готовыми программными продуктами, в результате чего выявлено, что данное ПО экономически эффективно.

# 1. Аналитический обзор

# 1.1. Автоматизация процесса инвентаризация

На данный момент автоматизация процесса инвентаризации проводится с использованием трёх технологий хранения информации:

1. Штрих-коды.
2. QR-коды.
3. Радиочастотные метки (RFID-метки).

Сам же процесс автоматической инвентаризации сводится к трём этапам:

1. Этап печати этикеток, на которых находятся штрих-коды, QR-коды, RFID-метки, в соответствии с выбранной технологией хранения информации и дальнейшая маркировки необходимых объектов материальных ценностей.
2. Этап самого процесса инвентаризации. Ответственное лицо с помощью специального оборудования считывает информацию с различных этикеток.
3. Этап выгрузки и анализ проведённой инвентаризации. Данные с устройства, использованного во время инвентаризации, переносятся на компьютер, на котором установлено необходимое программное обеспечение и в этом ПО происходит обработка полученных данных.

В статье «Современные способы автоматизации процесса инвентаризации» [8] утверждается, что растёт количество компаний, предпочитающих автоматизацию процесса инвентаризации ручному способу учёта имущества, так как, в настоящее время, отличительной чертой успешного ведения бизнеса является внедрение передовых программных разработок и технических средств, которые значительно повышают производительность и качество работы.

В качестве примеров систем, разработанных для автоматизации процесса инвентаризации, можно привести технические средства и ПО компании «Саотрон» (Saotron), программный продукт «Инвентаризации имущества» аудиторской фирмы МКПЦН, модуль «Электронная инвентаризация» корпорации «Парус», программное обеспечение «Клеверенс: Инвентаризация имущества» от компании Клеверенс Софт.

Автоматизация — это трудоемкий и затратный процесс, так как для её проведения необходимо запланировать расходы на стоимость оборудования, программного обеспечения и её внедрения.

Одним из важных параметров, при выборе технологии автоматизации инвентаризации, является время её проведения. В статье «Решение для инвентаризации товаров на складе на RFID метках» [9] было проведено сравнение затраченного времени на инвентаризацию при использовании разных технологий. Оказалось, что инвентаризация при помощи технологии RFID выполняется в 40 раз быстрее, чем по штрих-кодам, и от 100 до 200 раз быстрее, чем вручную.

Основным принципом современной инвентаризационной технологии является нанесение штрих-кода на единицу имущества, чтобы затем считать его и перенести в базу данных. В настоящее время активно используется технология «нового поколения» – метод автоматической идентификации объектов, в котором посредством радиосигналов считываются или записываются данные, хранящиеся в метках, прикрепляемых к объектам. Использование данной технологии позволяет не сканировать штрих-код каждой единицы, а сканировать сигнал всех радиочастотных меток в том или ином помещении сразу, что способствует более точной и быстрой обработке данных в процессе проведения инвентаризации, а также позволяет максимально исключить фактор человеческой ошибки, в результате чего повышается достоверность ведения бухгалтерского учёта.

# 1.2. Описание характеристик, разрабатываемого ПО

Разрабатываемое ПО должно обладать рядом характеристик:

* Иметь графический интерфейс.
* Управлять УРМ для записи и считывания данных с RFID – меток.
* Добавлять, удалять и редактировать данные в встроенной базе данных, экспортировать базу данных в CSV файл.

# 1.3. Сравнительный анализ имеющихся на рынке программных обеспечений

Множество компаний занимаются разработкой ПО, но не все из них имеют перечисленные в предыдущем пункте характеристиками. Программные продукты, созданные компанией «Саотрон», Корпорацией «Парус», ООО «Клеверенс Софт», имеют следующий схожий функционал:

* Могут управлять устройством для записи и считывания данных с меток.
* Отправлять базу данных на устройство для дальнейшей обработки на нём.
* Добавлять, удалять и редактировать строки в базе данных.
* Выгружать сформированную базу данных в файлы различных форматов.
* Обрабатывать базу данных, полученную с устройства и выдавать отчёт по окончании обработки.

Сравнение по наличию необходимого функционала разработанного программного обеспечения с аналогами программных продуктов, перечисленных выше компаний показано в таблице 1.

Таблица 1 – Сравнение ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Особенности | Возможность модификации | Возможность  работы с БД | Возможность записи и считывание данных с RFID-меток | Возможность экспорта базы данных |
| Программное  обеспечение |
| Разрабатываемое ПО | + | + | + | + |
| RADIO Scaner Srv | - | + | + | + |
| Cleverence RFID | - | + | + | + |
| Парус - Бюджет 10 | - | + | + | + |

Таблица 1 показывает, что разработанное ПО имея схожие с аналогами возможности, обладает преимуществами в том, что при модификации УРМ и появлении у него новых функций, приложение так же может быть модифицировано в соответствии с новым функционалом УРМ.

# 1.4. Используемый язык программирования и среда разработки

Для проекта был выбран язык программирования С++ и среда разработки Qt, которая является кроссплатформенным инструментарием разработки ПО на различных языках программирования. Важным преимуществом Qt является хорошо продуманный, логичный и стройный набор классов, предоставляющий программисту очень высокий уровень абстракции. Благодаря этим свойствам, код получается стройнее и проще, логичнее и понятнее, чем аналогичный по функциональности код, например, MFC. Его легче поддерживать и развивать. Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанная с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая содержит редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений.

# 2. Проектно-расчётная часть

# 2.1. Схема взаимодействия пользователя с ПО и УРМ

Взаимодействие пользователя с созданным программным обеспечением происходит при помощи приложения, которое, взаимодействуя с УРМ, позволяет:

* считать данные с RFID-метки;
* записать данные на RFID-метку;
* работать со встроенной базой данных.

Схема взаимодействия показана на рисунке 1.

Рисунок 1 – Взаимодействие пользователя, приложения, УРМ и метки

# **2.2. Протокол управления между УРМ и ПК**

Данный пункт содержит описание набора команд, с помощью которых программа - клиент позволяет работать с RFID-метками. Они показаны в таблице 2.

Таблица 2 – Описание команд

|  |  |  |  |
| --- | --- | --- | --- |
| Название команды | Синтаксис | Описание команды | Возвращаемое значение |
| Запись данных | w[address]:[data] | Записывает данные data начиная с адреса address | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение данных | r[address]:[number] | Считывание определённого числа number данных начиная с адреса address | При успешном завершении возвращает считанные данные. В случае ошибки вернёт код ошибки. |
| Запись версии | v:[data] | Запись версии data метки | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение версии | v | Считывает версию метки | При успешном завершении возвращает считанную версию. В случае ошибки вернёт код ошибки. |
| Запись инвентарного номера | i:[data] | Записывает инвентарный номер data объекта | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение инвентарного номера | i | Считывает инвентарный номер объекта | При успешном завершении возвращает считанный инвентарный номер. В случае ошибки вернёт код ошибки. |
| Запись комментария | c:[data] | Записывает описание data прибора на метку | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение комментария | c | Считывает описание прибора с метки | При успешном завершение возвращает считанный комментарий. В случае ошибки вернёт код ошибки. |
| Блокировка карты | b | Переключает метку в состояние «Активная», для выполнения нескольких команд | В случае ошибки вернёт код ошибки |
| Разблокировка карты | ub | Переключает метку в состояние «Неактивная». (Использовать эту команду только в случае если метка в состояние «Активная») | В случае ошибки вернёт код ошибки |
| Cправка команд | h | Показывает сведения о командах | Выводит информацию о всех командах |

# 2.3. Структура базы данных

В качестве базы данных был выбран продукт SQLite, представляющий собой библиотеку, в которой компонуется программа, и SQLite становится составной частью программы. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных в единственном стандартном файле на том компьютере, на котором исполняется программа.

Так как каждая метка содержит в себе идентификатор и описание, то база данных состоит из двух столбцов:

* инвентарный\_номер - integer primary key, который имеет целочисленный тип данных и является уникальным значением для каждой записи;
* описание – varchar, который имеет строковый тип данных переменой длины.

# 2.4. Описание средств, используемых в разработке ПО

Для разработки программы использовалась среда разработки: Qt Creator 4.9.1 (Community). Данный программный продукт имеет лицензию GNU GPL, что означает, что он является свободным для использования.

Для разработки программы использовался язык программирования C++.

В коде используются библиотеки, предоставляемые Qt Creator:

* QMainWindow - предоставляет главное окно приложения.
* QtSerialPort/QSerialPort - предоставляет набор основных методов и свойств для доступа к ресурсам последовательных портов.
* QtSerialPort/QSerialPortInfo - позволяет получить информацию о имеющихся в системе последовательных портах.
* QString - предоставляет строку символов Unicode.
* QByteArray - предоставляет массив байт.
* QTextCodec - обеспечивает преобразования между кодировками текста.
* QMessageBox - предоставляет целую серию статических методов, с помощью которых можно создавать окна сообщений.
* QObject - это базовый класс для всех объектов Qt.
* QtSql/QtSql - помогает обеспечить интеграцию БД.
* QTableView - реализует табличное представление, которое отображает элементы из модели.
* QApplication - руководит управляющей логикой графического пользовательского интерфейса и основными настройками.

# 2.5. Обзор кода программного обеспечения

При разработке ПО были написаны:

* проектный файл DP.pro;
* заголовочный файл dbfacade.h – для работы с базой данных;
* заголовочный файл mainwindow.h – для работы с УРМ и графическим интерфейсом;
* файл реализации dbfacade.cpp;
* файл реализации main.cpp;
* файл реализации mainwindow.cpp;
* файл формы mainwindow.ui.

Проектный файл DP.pro, автоматически создаётся Qt Creator. В нем находятся вся необходимая информация для того, чтобы qmake смог собрать проект. Исходный код представлен ниже.

**Начала кода**

#-------------------------------------------------

#

# Project created by QtCreator 2019-05-16T11:58:46

#

#-------------------------------------------------

QT += core gui # Включено по умолчанию

QT += serialport # Добавляем так как используем библиотеку QtSerialPort

QT += sql # Добавляем так как используем библиотеку QtSql

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets # Включено по умолчанию

TARGET = DP # Имя проекта

TEMPLATE = app # Создаёт make-файл для сборки приложения.

# The following define makes your compiler emit warnings if you use

# any feature of Qt which has been marked as deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS # Передать опции для компилятора.

# You can also make your code fail to compile if you use deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

CONFIG += c++11 # Задаёт опции которые должен использовать компилятор.

SOURCES += \ # Подключённые к проекту файлы реализации

dbfacade.cpp \

main.cpp \

mainwindow.cpp

HEADERS += \ # Подключённые к проекту заголовочные файлы

dbfacade.h \

mainwindow.h

FORMS += \ # Подключённые к проекту файлы формы

mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

**Конец кода**

В заголовочном файле dbfacade.h находится код для создания и взаимодействия с базой данных. Исходный код представлен ниже.

**Начала кода**

#ifndef DBFACADE\_H

#define DBFACADE\_H

#include <QObject>

#include <QtSql/QtSql>

class **QTableView**;

class **QSqlTableModel**;

class **DBFacade** : public QObject

{

Q\_OBJECT

public:

**DBFacade**(QTableView\* tags, QObject \*parent = nullptr);

virtual ~***DBFacade***();

void **addTag**(QString id, QString com);

// добавляет

void **repTag**(QString id, QString com);

// заменяет

void **remTag**(QString id);

// удаляет

bool **checkTagID**(QString id);

// проверяет наличие id в базе данных

void **expTable**();

// экспортирует

protected:

QSqlDatabase m\_db;

// база данных

QSqlQuery \*m\_query;

// запрос к базе

QSqlRecord m\_rec;

// строка таблицы (ответ на запрос)

QSqlTableModel \*m\_tagsModel;

// модель таблицы

};

#endif // DBFACADE\_H

**Конец кода**

В заголовочном файле mainwindow.h находится код для взаимодействия с графическим интерфейсом и УРМ. Исходный код представлен ниже.

**Начало кода**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtSerialPort/QSerialPort>

#include <QtSerialPort/QSerialPortInfo>

#include <QString>

#include <QByteArray>

#include <QTextCodec>

#include <QMessageBox>

namespace Ui

{

class MainWindow;

}

class DBFacade;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_WriteDataOnCard\_clicked();

//Функция, отвечающая за отправку запроса на запись данных на метку

void on\_ReadDataOnCard\_clicked();

//Функция, отвечающая за отправку запроса на чтение данных с метки

void on\_DevCon\_clicked();

//Функция, отвечающая за подключение к устройству

void on\_DevDis\_clicked();

//Функция, отвечающая за отключение от устройства

void on\_tagAdd\_clicked();

//Функция, отвечающая за запись данных в базу данных

void on\_tagRem\_clicked();

//Функция, отвечающая за удаление данных из базы данных

void on\_ExpTagTab\_clicked();

//Функция, отвечающая за вывод данных из базы данных в csv файл

void ReadData();

//Функция, отвечающая за чтение данных с порта

bool CheckDigID(QString id);

//Функция, отвечающая за проверку введённого инвентарного номера на наличие букв

private:

Ui::MainWindow \*m\_ui;

DBFacade \*m\_db;

QSerialPort \*thisPort = nullptr;

QTextCodec \*codecC = QTextCodec::codecForName( "CP1251" );

QTextCodec \*codecU = QTextCodec::codecForName( "UTF-8" );

};

#endif // MAINWINDOW\_H

**Конец кода**

В файле реализации dbfacade.cpp находится код, в котором объявляется объект класса DBFacade и идёт дальнейшая работа с ним, а также написана реализация методов класса DBFacade. Исходный код представлен ниже.

**Начало кода**

#include "dbfacade.h"

#include <QTableView>

DBFacade::**DBFacade**(QTableView\* tags, QObject \*parent)

: QObject(parent)

{

m\_db = QSqlDatabase::addDatabase("QSQLITE");

//Активизируем драйвер

m\_db.setDatabaseName("DataBaseForDProject");

//Устанавливаем имя даты базы

m\_db.open();

//Открываем базу данных

m\_query = new QSqlQuery(m\_db);

//Создаем объект класса QSqlQuery для использования команд SQL

if (false == m\_db.tables().contains("tags"))

m\_query->exec("create table tags (Инвентарный\_номер integer primary key, Описание varchar)");

//Передаем команду на создание таблицы с именем tags со столбцами Инвентарный\_номер, Описание

m\_tagsModel = new QSqlTableModel(this, m\_db);

//Класс QSqlTableModel позволяет отображать данные в табличной и иерархической форме

m\_tagsModel->*setTable*("tags");

//Устанавливает таблицу базы данных, с которой работает модель

m\_tagsModel->*select*();

//Вызов select() производит заполнение данными.

m\_tagsModel->*setEditStrategy*(QSqlTableModel::OnFieldChange);

//Производит запись после того, как пользователь перейдёт к другой ячейке таблицы

tags->*setModel*(m\_tagsModel);

//Устанавливает модель

tags->setEditTriggers(QAbstractItemView::NoEditTriggers);

//Блокирует ячейки таблицы из экрана приложения

}

**//**Деструктор

DBFacade::~***DBFacade***()

{

delete m\_query;

}

//Функция, отвечающая за добавление строк

void DBFacade::**addTag**(QString id, QString com)

{

m\_query->first();

m\_query->prepare("INSERT INTO tags (Инвентарный\_номер, Описание) VALUES ('"+id+"','"+com+"')");

//Подготавливает SQL запрос к выполнению

m\_query->exec("INSERT INTO tags (Инвентарный\_номер, Описание) VALUES ('"+id+"','"+com+"')");

//Отправляем запрос

m\_tagsModel->*select*();

//Заполнение данными

}

//Функция, отвечающая за замену строк

void DBFacade::**repTag**(QString id, QString com)

{

m\_query->prepare("REPLACE INTO tags (Инвентарный\_номер, Описание) VALUES ('"+id+"','"+com+"')");

//Подготавливает SQL запрос к выполнению

m\_query->exec("REPLACE INTO tags (Инвентарный\_номер, Описание) VALUES ('"+id+"','"+com+"')");

//Отправляем запрос

m\_tagsModel->*select*();

}

//Функция, отвечающая за проверку совпадающих идентификаторов

bool DBFacade::**checkTagID**(QString id)

{

m\_query->exec(("SELECT Инвентарный\_номер FROM tags WHERE Инвентарный\_номер =")+ id);

{

while (m\_query->next())

{

return true;

}

}

return false;

}

//Функция, отвечающая за удаление строки

void DBFacade::**remTag**(QString id)

{

m\_query->exec(("DELETE FROM tags WHERE Инвентарный\_номер = ") + id);

//Удаляем из таблицы строку с нужным id

m\_tagsModel->*select*();

}

//Функция, отвечающая за экспорт базы данных

void DBFacade::**expTable**()

{

QString textData;

//Создаём строку textData

int rows = m\_tagsModel->*rowCount*();

//Создаём целочисленную переменную rows и присваиваем ей кол-во строк в базе данных

int columns = m\_tagsModel->*columnCount*();

//Создаём целочисленную переменную columns и присваиваем ей кол-во столбцов в базе данных

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

textData += m\_tagsModel->*data*(m\_tagsModel->*index*(i,j)).toString();

textData += ";";

}

textData += "\n";

}

//Создаём два цикла по кол-ву строк и столбцов и заполняем переменную textData

//информацией из строк и столбцов, далее ставим ;

//и переходим на новую строку с помощью управляющего символа \n

QFile csvFile("DataBaseTag.csv");

//Создаем файл в формате csv с названием "DataBaseTag.csv"

if(csvFile.*open*(QIODevice::WriteOnly | QIODevice::Truncate)) // Если файл открыт для записи или перезаписи, то

{

QTextStream out(&csvFile);

//Создаем объект класса QTextStream out

out << textData;

//И передаем ей переменую textData

csvFile.*close*();

//Закрываем файл

}

}

**Конец кода**

В файле реализации main.cpp находится код, в котором объявлены объекты класса QApplicationи MainWindow, а также установлена кодировка CP1251. Исходный код представлен ниже.

**Начало кода**

#include "mainwindow.h"

#include <QApplication>

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

{

QTextCodec\* codec\_cp = QTextCodec::codecForName("CP1251");

//Создаём переменную codec\_cp и устанавливаем кодировку CP1251

QTextCodec::setCodecForLocale(codec\_cp);

//Делаем её локальной

QApplication a(*argc*, argv);

//Создаём объект класса QApplication, который управляет нашими окнами и виджетами.

MainWindow w;

//Создаём свой виджет.

w.show();

// Пока мы не вызовем метод show, виджет скрыт.

return a.exec(); //exec, обрабатывает события от пользователя,

// графической системы и операционной системы

// и передаёт их конкретным виджетам.

}

**Конец кода**

В файле реализации mainwindow.h находится код, в котором объявлен объект класса MainWindow и SerialPort и идёт дальнейшая работа с объектами, а также написана реализация методов класса MainWindow. Исходный код представлен ниже.

**Начало кода**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "dbfacade.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

m\_ui(new Ui::MainWindow),

thisPort(new QSerialPort(this))

{

m\_ui->setupUi(this);

m\_db = new DBFacade(m\_ui->tagView, this);

//Считываем активные последовательные порты

foreach (const QSerialPortInfo &info, QSerialPortInfo::availablePorts())

//Отправляем эти данные в comboBox

m\_ui->DevComboBox->addItem(info.portName());

//По сигналу readyRead вызывает функцию ReadData

connect(thisPort, SIGNAL(readyRead()),this,SLOT(ReadData()));

}

**//**Деструктор

MainWindow::~***MainWindow***()

{

delete m\_ui;

}

//Функция, отвечающая за подключение к устройству

void MainWindow::**on\_DevCon\_clicked**()

{

//Выбираем имя нашего последовательного порта в выпадающем списке

if (thisPort->portName() != m\_ui->DevComboBox->currentText())

{

thisPort->*close*();

thisPort->setPortName(m\_ui->DevComboBox->currentText());

}

//Устанавливаем настройки порта

thisPort->setBaudRate(QSerialPort::Baud4800);

thisPort->setDataBits(QSerialPort::Data8);

thisPort->setParity(QSerialPort::NoParity);

thisPort->setStopBits(QSerialPort::OneStop);

thisPort->setFlowControl(QSerialPort::NoFlowControl);

thisPort->*open*(QSerialPort::ReadWrite);

//Если порт открыт

if (thisPort->isOpen())

{

//Вывод текста в всплывающее окно

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством установлено");

}

else //или

{

//Вывод текста в всплывающее окно

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством не установлено");

}

}

//Функция, отвечающая за отключение от устройства

void MainWindow::**on\_DevDis\_clicked**()

{

//Если порт открыт

if(thisPort->isOpen())

{

//Закрываем

thisPort->*close*();

// Вывод текста в MessageBox

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством разорвано");

}

else //или

{

//Вывод текста в MessageBox

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством не установлено");

}

}

//Функция, отвечающая за отправку запроса на запись данных на метку

void MainWindow::**on\_WriteDataOnCard\_clicked**()

{

m\_ui->DevTagBrowser->clear();

//Если порт открыт

if(thisPort->isOpen())

{

if (m\_ui->tagID->text().isEmpty() || m\_ui->tagCom->toPlainText().isEmpty())

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Поле инвентарного номера и/или поле описания пусто->заполните их");

return;

}

if (m\_ui->tagID->text().length() > 19)

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Инвентарный номер не может быть больше 19 цифр");

return;

}

if (!CheckDigID(m\_ui->tagID->text()))

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Инвентарный номер должен состоять только из цифр");

return;

}

if (m\_ui->tagCom->toPlainText().length() > 742)

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Поле описания слишком большое, максимум 742 символа");

return;

}

//Блокировка метки

thisPort->write("b");

thisPort->write("\xd");

//Создаём и обнуляем

QByteArray dataOutI = nullptr;

QByteArray dataOutC = nullptr;

//Присваиваем dataOutI значение полученные из tagID

//Присваиваем dataOutC значение полученные из tagCom

dataOutI = "i:"+m\_ui->tagID->text().toLocal8Bit();

dataOutC = "c:"+m\_ui->tagCom->toPlainText().toLocal8Bit();

//Записываем данные в порт

for (int i = 0; i < dataOutI.count(); ++i)

{

thisPort->putChar(dataOutI[i]);

}

thisPort->write("\xd");

//Записываем данные в порт

for (int i = 0; i < dataOutC.count(); ++i)

{

thisPort->putChar(dataOutC[i]);

}

thisPort->write("\xd");

//Разблокировка метки

thisPort->write("ub");

thisPort->write("\xd");

}

else //или

{

//Вывод текста в MessageBox

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством не установлено");

}

}

//Функция, отвечающая за отправку запроса на чтение данных с метки

void MainWindow::**on\_ReadDataOnCard\_clicked**()

{

m\_ui->DevTagBrowser->clear();

//Если порт открыт

if(thisPort->isOpen())

{

//Блокируем

thisPort->write("b");

thisPort->write("\xd");

//Отправляем на устройство i\xd

thisPort->write("i");

thisPort->write("\xd");

thisPort->write("c");

thisPort->write("\xd");

//Разблокируем

thisPort->write("ub");

thisPort->write("\xd");

}

else //или

{

//Вывод текста в MessageBox

QMessageBox::information(nullptr, "Внимание", "Соединение с устройством не установлено");

}

}

//Функция, отвечающая за запись данных в базу данных

void MainWindow::**on\_tagAdd\_clicked**()

{

//Если поля tagID и tagCom пусто, то return

if (m\_ui->tagID->text().isEmpty() || m\_ui->tagCom->toPlainText().isEmpty())

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Поле инвентарного номера и/или поле описания пусто->заполните их");

return;

}

if (m\_ui->tagID->text().length() > 19)

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Инвентарный номер не может быть больше 19 цифр");

return;

}

if (!CheckDigID(m\_ui->tagID->text()))

{

QMessageBox::information(nullptr, "Внимание", "Инвентарный номер должен состоять только из цифр");

return;

}

if (m\_ui->tagCom->toPlainText().length() > 742)

{

//Вызов окна с информацией

QMessageBox::information(nullptr, "Внимание", "Поле описания слишком большое, максимум 742 символа");

return;

}

//Если функция checkTagID вернул true

if (m\_db->checkTagID(m\_ui->tagID->text()))

{

//Создаём оповещение если строка с введённым инвентарным номером уже существует

//и предлагает перезаписать строку

int n = QMessageBox::warning(nullptr,

"Внимание",

"Строка с введённым инвентарным номером уже существует."

"\nЖелаете ли вы перезаписать строку?",

"Да",

"Нет",

QString(),

0,

1

);

//Если нажали да, то вызываем функцию repTag и передаем данные из полей и return

if(!n)

{

m\_db->repTag(m\_ui->tagID->text(), m\_ui->tagCom->toPlainText());

return;

}

//Если нажали нет, то return

if(n)

{

return;

}

}

//Вызываем функцию addTag и передаём ей параметры полей

m\_db->addTag(m\_ui->tagID->text(), m\_ui->tagCom->toPlainText());

}

//Функция, отвечающая за удаление данных из базы данных

void MainWindow::**on\_tagRem\_clicked**()

{

if (m\_ui->tagID->text().isEmpty()) //Если поле tagID пусто, то return

{

QMessageBox::information(nullptr, "Внимание", "Поле инвентарного номера пусто->заполните его");

return;

}

//Если функция checkTagID вернул false

if (!m\_db->checkTagID(m\_ui->tagID->text()))

{

QMessageBox::information(nullptr, "Внимание", "Строки с данным инвентарным номером нет");

return;

}

//Вызываем функцию remTag и передаем ей параметр поля

m\_db->remTag(m\_ui->tagID->text());

}

//Функция, отвечающая за вывод даных из базы данных в csv файл

void MainWindow::**on\_ExpTagTab\_clicked**()

{

m\_db->expTable();

//Вызов метода expTable для экспорта данных

QMessageBox::information(nullptr, "Внимание", "Файл создан"); //Оповещение о создании файла

}

//Функция, отвечающая за проверку введённого инвентарного номера на наличие букв

bool MainWindow::**CheckDigID**(QString id)

{

for (int i = 0; i < id.count(); ++i)

{

if (!id[i].isDigit()) //Если i-ый элемент не цифра, то возвращаем false

return false;

}

return true;

}

//Функция, отвечающая за чтение данных с порта

void MainWindow::**ReadData**()

{

//Создаём и обнуляем

QByteArray dataIn = nullptr;

//Присваиваем dataIn данные полученные с порта

dataIn = thisPort->readAll();

//Создаём переменную strf и перекодируем её из cp1251 в юникод

QString strf = codecC->toUnicode(dataIn);

//Переменой dataIn присваиваем значения строки strf, перекодируя из юникода в utf-8

dataIn = codecU->fromUnicode(strf);

//Выводим на экран данные полученные с порта

m\_ui->DevTagBrowser->insertPlainText(dataIn);

}

**Конец кода**

Файл формы mainwindow.ui создаётся с использованием инструмента Qt Designer. Qt Designer – это инструмент для проектирования и создания графических пользовательских интерфейсов из компонентов Qt. Он является встроенным в Qt Creator, но может запускаться отдельно.

Виджеты и формы, созданные с помощью Qt Designer, интегрированы с управляющим кодом, использующим механизм сигналов и слотов Qt, который позволяет установить поведение графических элементов при взаимодействии с ними. Все свойства, установленные в Qt Designer, можно изменить динамически внутри кода.

Разработанный интерфейс сохраняется в файл с расширением ui, который подключается к создаваемой программе с помощью специальных методов библиотеки Qt.

На рисунке 2 изображены использованные объекты и соответствующие им классы.

Рисунок 2 – Таблица использованных элементов

Ниже приведён код, находящийся в файле mainwindow.ui.

**Начало кода.**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1090</width>

<height>730</height>

</rect>

</property>

<property name="windowTitle">

<string>DProject</string>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="QPushButton" name="WriteDataOnCard">

<property name="geometry">

<rect>

<x>20</x>

<y>660</y>

<width>195</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Записать на метку</string>

</property>

</widget>

<widget class="QLabel" name="label\_2">

<property name="geometry">

<rect>

<x>20</x>

<y>400</y>

<width>120</width>

<height>20</height>

</rect>

</property>

<property name="text">

<string>Ввод описания: </string>

</property>

</widget>

<widget class="QLineEdit" name="tagID">

<property name="geometry">

<rect>

<x>20</x>

<y>380</y>

<width>195</width>

<height>20</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>20</x>

<y>350</y>

<width>181</width>

<height>20</height>

</rect>

</property>

<property name="text">

<string>Ввод инвентарного номера:</string>

</property>

</widget>

<widget class="QPushButton" name="ReadDataOnCard">

<property name="geometry">

<rect>

<x>120</x>

<y>10</y>

<width>95</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Считать данные

с метки</string>

</property>

</widget>

<widget class="QPushButton" name="tagAdd">

<property name="geometry">

<rect>

<x>20</x>

<y>610</y>

<width>95</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Добавить

в базу данных</string>

</property>

</widget>

<widget class="QPushButton" name="tagRem">

<property name="geometry">

<rect>

<x>120</x>

<y>610</y>

<width>95</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Удалить

из базы данных</string>

</property>

</widget>

<widget class="QTableView" name="tagView">

<property name="geometry">

<rect>

<x>230</x>

<y>50</y>

<width>840</width>

<height>595</height>

</rect>

</property>

<property name="sortingEnabled">

<bool>false</bool>

</property>

<attribute name="horizontalHeaderMinimumSectionSize">

<number>395</number>

</attribute>

<attribute name="horizontalHeaderDefaultSectionSize">

<number>395</number>

</attribute>

<attribute name="horizontalHeaderHighlightSections">

<bool>false</bool>

</attribute>

<attribute name="verticalHeaderMinimumSectionSize">

<number>70</number>

</attribute>

<attribute name="verticalHeaderDefaultSectionSize">

<number>70</number>

</attribute>

<attribute name="verticalHeaderHighlightSections">

<bool>false</bool>

</attribute>

</widget>

<widget class="QTextEdit" name="tagCom">

<property name="geometry">

<rect>

<x>20</x>

<y>425</y>

<width>195</width>

<height>180</height>

</rect>

</property>

</widget>

<widget class="QComboBox" name="DevComboBox">

<property name="geometry">

<rect>

<x>20</x>

<y>10</y>

<width>95</width>

<height>35</height>

</rect>

</property>

</widget>

<widget class="QPushButton" name="DevDis">

<property name="geometry">

<rect>

<x>120</x>

<y>50</y>

<width>95</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Отсоединиться

от устройства</string>

</property>

</widget>

<widget class="QPushButton" name="DevCon">

<property name="geometry">

<rect>

<x>20</x>

<y>50</y>

<width>95</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Соединиться

с устройством</string>

</property>

</widget>

<widget class="QTextBrowser" name="DevTagBrowser">

<property name="geometry">

<rect>

<x>20</x>

<y>90</y>

<width>195</width>

<height>255</height>

</rect>

</property>

</widget>

<widget class="QLabel" name="label\_3">

<property name="geometry">

<rect>

<x>610</x>

<y>20</y>

<width>70</width>

<height>20</height>

</rect>

</property>

<property name="text">

<string>База Данных </string>

</property>

</widget>

<widget class="QPushButton" name="ExpTagTab">

<property name="geometry">

<rect>

<x>550</x>

<y>660</y>

<width>195</width>

<height>35</height>

</rect>

</property>

<property name="text">

<string>Экспорт базы данных в CSV файл</string>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1090</width>

<height>21</height>

</rect>

</property>

</widget>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>

**Конец кода.**

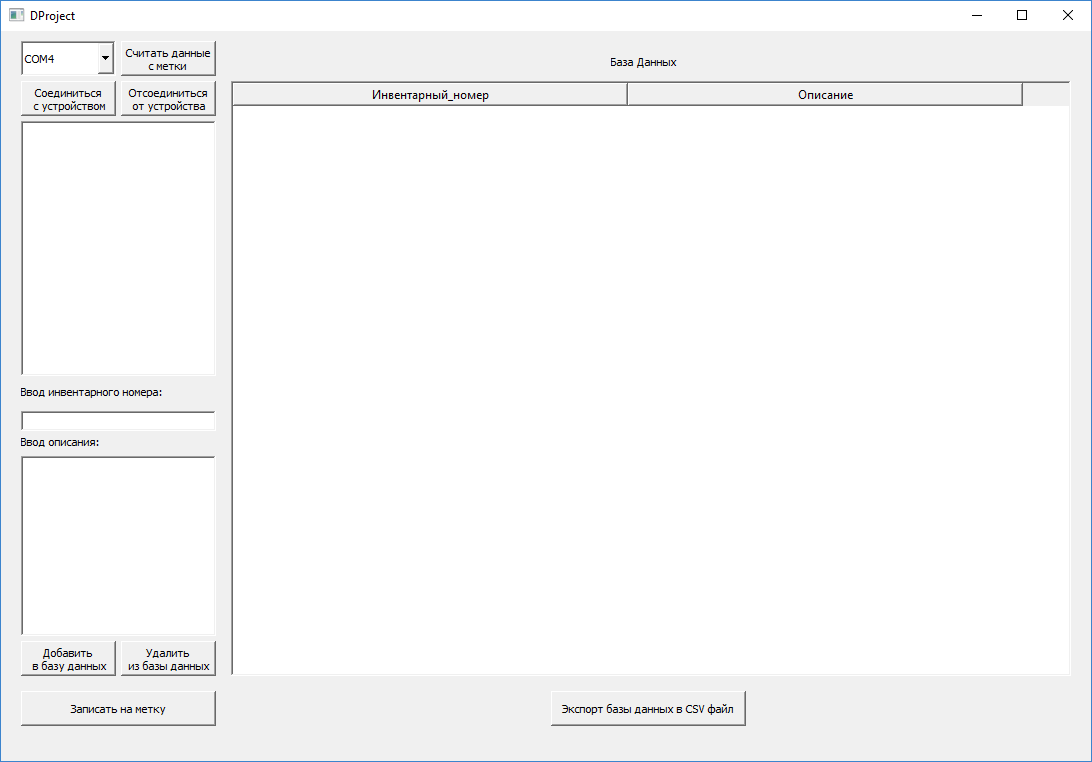
# 2.6. Описание функционала ПО

На рисунке 3 изображён общий вид приложения. Программу можно поделить на две части:

1. работа с RFID-метками;
2. работа с базой данных.

Рассмотрим часть программы, отвечающая за работу с метками. Программное обеспечение может:

* считать данные с метки;
* записать данные на метку.

Рисунок 3 – Общий вид приложения

Для начала работы нужно выбрать последовательный порт, через который подключено УРМ. На рисунке 4 изображён выпадающий список доступных для выбора портов.

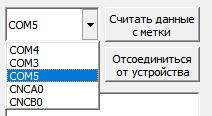


Рисунок 4 – Список портов

После выбора нужного порта следует подключиться к УРМ. На рисунке 5 изображён результат выполненного подключения.

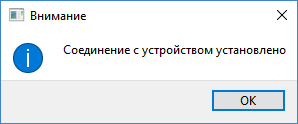


Рисунок 5 – Оповещение о подключении

Сделав все приготовления нужно приложить метку к УРМ и нажать кнопку «Считать данные с метки». На рисунке 6 изображён результат удачного считывания данных с метки.

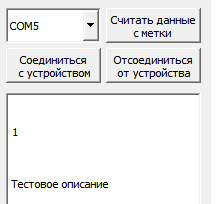


Рисунок 6 – Вывод данных в окно

Для записи данных на метку следует заполнить поля «Ввод инвентарного номера:» и «Ввод описания:». На рисунке 7 показан пример вводимых данных.

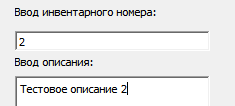


Рисунок 7 – Заполненные поля

После нажатия кнопок «Записать на метку» в окно выводится результат записи данных на метку. На рисунке 8 показан положительный вариант.

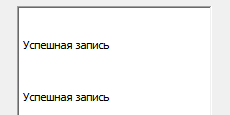


Рисунок 8 – Вывод об удачной записи

Рассмотрим часть программы, отвечающая за работу с базой данных.

Программное обеспечение может:

* добавлять строки в базу данных;
* редактировать строки в базе данных;
* удалять строки из базы данных;
* экспортировать базу данных в csv файл.

Для того, чтобы добавить новый элемент в базу данных нужно заполнить поля «Ввод инвентарного номера:» и «Ввод описания:». На рисунке 9 показан ввод данных.

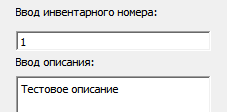


Рисунок 9 – Пример ввода данных

После нажать кнопку «Добавить в базу данных». На рисунке 10 показана база данных с новой строкой.

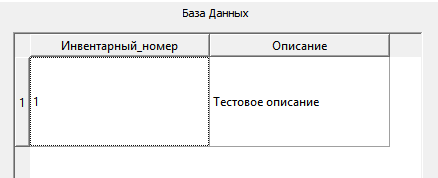


Рисунок 10 – Пример базы данных

Для редактирования строки нужно в поле «Ввод инвентарного номера:» ввести уже имеющийся в базе данных инвентарный номер, заполнить поле «Ввод описания:» и нажать на кнопку «Добавить в базу данных» (рисунок 11).

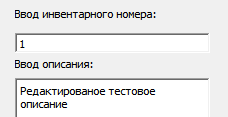


Рисунок 11 – Пример вводимых данных

На рисунке 12 показано оповещение о редактировании строки, в котором пользователь либо соглашается с перезаписью, либо отказывается от неё.

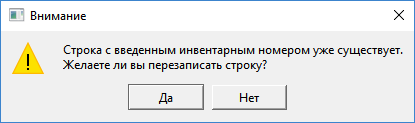


Рисунок 12 – Оповещение о перезаписи

На рисунке 13 представлен результат редактирования строки в базе данных.

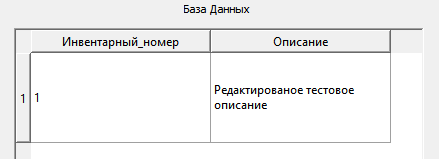


Рисунок 13 – Отредактированная строка в базе данных

Для удаления строки нужно заполнить поле «Ввод инвентарного номера:» и нажать на кнопку «Удалить из базы данных» (рисунок 14).



Рисунок 14 – Ввод инвентарного номера удаляемой строки

Результат удаления строки из базы данных показан на рисунке 15.

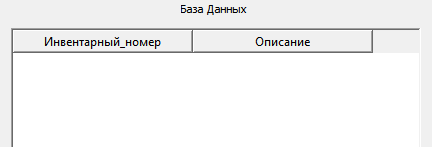


Рисунок 15 – База данных после удаления строки

Заново внесём в базу данных строку с инвентарным номером «1» и описанием «Редактированное текстовое описание». При нажатии на кнопку «Экспортировать базу данных в csv файл», программой будет создан файл DataBaseTag.csv в котором будет находиться вся база данных в формате «Инвентарный номер; описание». Этот файл можно использовать для дальнейших нужд инвентаризации. Например, распечатать часть или всю базу данных. Также появится оповещение о создание файла (рисунок 16).

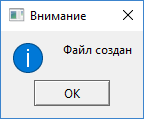


Рисунок 16 – Оповещение о создании файла

На рисунке 17 показан файл DataBaseTag.csv, открытый в программе Microsoft Office Excel.

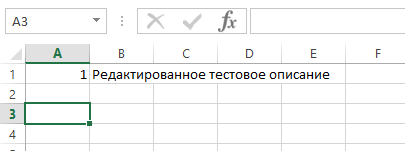


Рисунок 17 – Экспортированная база данных

Особенности УРМ и RFID-меток накладывают ряд ограничений при их применении, поэтому для зашиты пользователя от ошибочных действий были приняты соответствующие меры. Например, при какой-либо ошибки в работе приложения или УРМ появляется оповещение. Далее рассмотрим возможные ситуации, при которых могут возникать ошибки пользователя.

Предположим, что пользователь не соединился с УРМ или выбрал неправильный порт, тогда при нажатии на кнопки «Считать данные с метки», «Записать на метку», «Считать данные с метки», «Соединиться с устройством», «Отсоединиться от устройства», появляется оповещение, которое показано на рисунке 18.

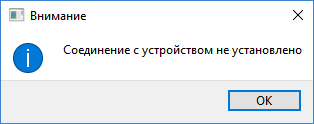


Рисунок 18 – Оповещение об отсутствии соединения.

Предположим, что соединение с УРМ прошло успешно, но при нажатии на кнопки «Считать данные с метки» и «Записать на метку», пользователь либо забыл приложить метку, либо после считывания или записи данных не убрал её. В этом случае появляется сообщение «Ошибка передачи данных» (рисунок 19).

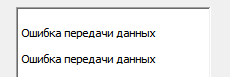


Рисунок 19 – Оповещение об ошибки при работе с меткой.

Если пользователь решил добавить новую строку в базу данных или записать данные на метку, но не заполнил поле инвентарного номера или поле описания, появляется оповещение, которое показано на рисунке 20.

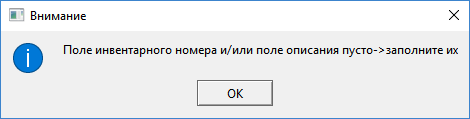


Рисунок 20 – Оповещение о пустых полях.

Предположим, что пользователь пытается удалить строку из базы данных, но не заполнил поле инвентарного номера, тогда при нажатии на кнопку «Удалить из базы данных», появляется оповещение, которое показано на рисунке 21.

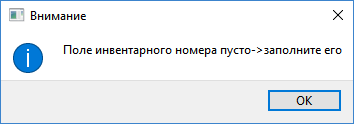


Рисунок 21 – Оповещение о пустом поле инвентарного номера.

Если пользователь пытается удалить строку с несуществующим в базе данных инвентарным номером, тогда при нажатии на кнопку «Удалить из базы данных», появляется оповещение, которое показано на рисунке 22.

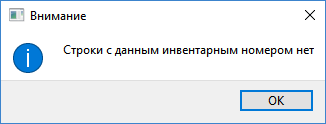


Рисунок 22 – Оповещение об отсутствии строки в базе данных.

Так как инвентарный номер может состоять только из цифр и иметь длину не более 19 символов, то при нарушении данных условий и нажатии на кнопки «Добавить в базу данных» и «Запись на метку» пользователь увидит оповещения об ошибке, показанные на рисунках 23 и 24.

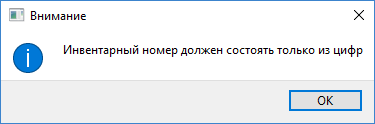


Рисунок 23 – Оповещение о наличие запрещённых символов.

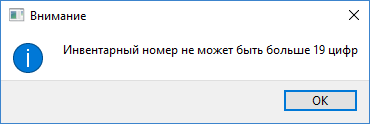


Рисунок 24 – Оповещение о превышении длины инвентарного номера.

Так как доступная память на RFID-метках ограничена, то длина описания не может превышать 742 символа. Поэтому при длине описания, превышающей ограничения, предусмотрено оповещение (рисунок 25), которое появляется при нажатии на кнопку «Добавить в базу данных» и «Запись на метку».

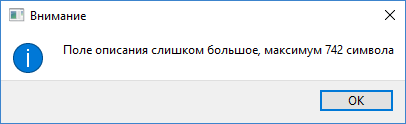


Рисунок 25 – Оповещение о превышении длины описания.

# 3. Экспериментальная часть

# 3.1 Тестирование работы ПО с УРМ в условиях внедрения

При внедрении ПО в лабораторном корпусе филиала РТУ МИРЭА в г. Фрязино было проведено тестирование взаимодействия приложенияс УРМ и RFID-метками.

Во время тестирования метку наклеили на источник питания (рисунок 26).

Рисунок 26 – Источник питания с наклеенной меткой

Для подключения УРМ к компьютеру, открыли приложение и выбрали последовательный порт (рисунке 27).

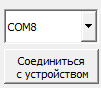


Рисунок 27 – Подключение порта

В приложении записали необходимые данные в формы для ввода (рисунок 28).

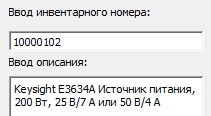


Рисунок 28 – Заполненные поля ввода

Нажав на кнопку «Добавить в базу данных», получили в базе данных добавленную строку (рисунок 29).

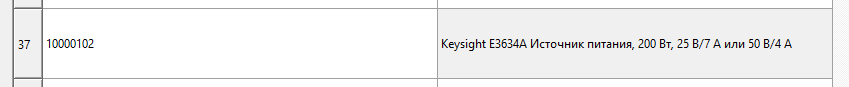


Рисунок 29 – Добавленная строка в базе данных

Нажав на кнопку «Записать на метку», записали данные на метку. Так как запись прошла успешно, приложение вывело сообщение, изображённое на рисунке 30.



Рисунок 30 – Успешный ответ

Если бы при записи произошёл сбой, то получили бы сообщение об ошибке (рисунок 31).



Рисунок 31 – Ответ ошибки

Для того, чтобы удостовериться в правильности записанных данных, считали их с метки, нажав на кнопку «Считать данные с метки» (рисунок 32).

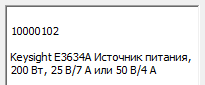


Рисунок 32 – Считанные данные

В результате внедрения программного обеспечения подтвердилась его работоспособность. Оно корректно взаимодействует с УРМ, даёт возможность записывать, считывать данные с RFID-метки, заполнять и редактировать базу данных.

# 4. Экономическая часть

# 4.1. Технико-экономическое обоснование

При автоматизации процесса инвентаризации в филиале РТУ МИРЭА г. Фрязино было разработано новое программно-аппаратного устройство для работы с RFID-метками. Для связи УРМ с ПК потребовалось программное обеспечение, имеющее графический интерфейс, с помощью которого можно управлять УРМ для записи и считывания данных с RFID – меток, а также добавлять, удалять и редактировать данные во встроенной базе данных, экспортировать базу данных в CSV файл.

Сравнение по наличию необходимого функционала программных продуктов, созданных компанией «Саотрон», корпорацией «Парус», ООО «Клеверенс Софт» показало, отсутствие возможности модификации этих приложений при расширении функций УРМ. Это послужило основанием для создания нового ПО, которое, имея схожий с аналогами функционал, можно усовершенствовать.

Разработанное программное обеспечение для работы с УРМ позволяет автоматизировать процесс инвентаризации, ускорить сбор и обработку полученных данных, сэкономить время сотрудников, кроме того имеет легкий и доступный к освоению интерфейс, что снижает квалификационные требования к проводящим инвентаризацию сотрудникам.

# 4.2. Аналитическая записка

В разработке программного обеспечения будут использоваться следующие технологии:

* Язык программирования C++. Используется для разработки программного обеспечения.
* Среда разработки Qt Creator. Содержит набор инструментов, используемых в разработке программного обеспечения.

Выбранные технологии распространяются на условиях лицензий свободного программного обеспечения, поэтому не требуют затрат на своё использование.

# 4.3. Сетевой график хода выполнения работ

Таблица 2 - Перечень работ

|  |  |
| --- | --- |
| Код работы | Наименование работы |
| 1-2 | Постановка проблемы |
| 2-3 | Анализ аналоговых решений и разработка уникальной концепции |
| 3-4 | Описание языка программирования и среды разработки |
| 4-5 | Проектировка и разработка программного обеспечения |
| 5-6 | Тестирование разработанного программного обеспечения и исправление ошибок |
| 6-7 | Внедрение решения в эксплуатацию |

Таблица 3 - Временная оценка продолжительности работ

|  |  |  |  |
| --- | --- | --- | --- |
| Код работы | Временная оценка деятельности работы (дни) | | |
| tmax | tmin | tij |
| 1-2 | 1 | 1 | 1 |
| 2-3 | 1 | 1 | 1 |
| 3-4 | 5 | 5 | 5 |
| 4-5 | 14 | 14 | 14 |
| 5-6 | 7 | 7 | 7 |
| 6-7 | 1 | 1 | 1 |

Таблица 4 - Расчёт сроков работ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Код работы | Продолжительность работы tij | Ранние сроки | | Поздние сроки | | Резервы времени |
| tрн | tро | tпн | tпо |
| 1-2 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2-3 | 1 | 1 | 2 | 1 | 2 | 0 |
| 3-4 | 5 | 2 | 7 | 2 | 7 | 0 |
| 4-5 | 14 | 7 | 21 | 7 | 21 | 0 |
| 5-6 | 7 | 21 | 28 | 21 | 28 | 0 |
| 6-7 | 1 | 28 | 29 | 28 | 29 | 0 |



Рисунок 33 –Сетевой график

Lкр = 1+1+5+14+7+1 = 29 дням

# 4.4. Расчёт затрат на проведение работ и составление сметы расходов

Затраты на внедрения разработанного программного продукта рассчитываются по формуле:

Квн = Рм + Рктс + Ртр + Рпо + Зфот + Рнач + Зэвм + Рн

Где:

Рм = 0 руб. – затраты на материал.

Рктс = 0 руб. – затраты на приобретение комплекса технических средств.

Ртр = 0 руб. – затраты на транспортировку, установку и наладку комплекса технических средств.

Рпо = Црпп + ΣЦпо = Црпп – т.к. для разработки использовалось свободно распространяющееся ПО учитывается только цена за разработанный программный продукт.

Зфот = Зосн \* Здоп \* Твн = Зосн \* Твн = 15 000 руб./мес. \* 2 ч. = (15 000 / 21 / 8) руб./ч. \* 2 ч. = 178,57 руб. – затраты на оплату работников, занимающихся внедрением проекта.

Рнач = Зфот \* Нсоц = 178,57 \* 30% = 178,57 \* 0,3 = 53,57 руб.

Зэвм = Тмвн \* *k*и \* *n* \* Cм-ч = 2 ч. \* 1 \* 1 \*60,18 руб./ч. = 120,36 руб. – затраты на эксплуатацию ЭВМ при внедрении решения.

Рн = Зфот \* *k*н = 0 руб. – накладные расходы.

Цена разработанного программного продукта состоит только из затрат на разработку. Продукт используется в нуждах филиала и не предполагается для продажи или получения выгоды. Рассчитывается по формуле:

Црпп = Зфот + Рнач + Зэвм

Зфот = Зосн \* Здоп \* Твн = Зосн \* Твн = 15 000 руб./мес. \* 29 дн. = 15 000 руб./мес. \*(29/21) мес. = 20 714, 28 руб.

Рнач = Зфот \* 30% = 6 214,28 руб.

Зэвм = Тмвн \* *k*и \* *n* \* Cм-ч = (29 \* 8) ч \* 1 \* 1 \* 6,18 руб./ч. = 1433,76 руб.

Получив все нужные нам значения, найдём цену разработанного продукта:

Црпп = Зфот + Рнач + Зэвм = 20 714,28 руб. + 6 214,28 руб. + 1433,76 руб. = 28 362,32 руб.

Таблица 5 - Затраты на разработку и внедрение программного продукта

|  |  |  |
| --- | --- | --- |
| Наименование затрат | Условное обозначение | Значение |
| Стоимость разработки программного продукта | Рпо | 28 362,32 руб |
| Фонд оплаты труда | Зфот | 178,57 руб. |
| Отчисления во внебюджетные фонды | Рнач | 53,57 руб. |
| Затраты на эксплуатацию ЭВМ | Зэвм | 120,36 руб. |
| Итого | Квн | 28 714,82 руб. |

**4.5.** Расчёт экономической эффективности и срока окупаемости проведённых работ

Рассмотрим затраты на внедрение существующих программных продуктов и на внедрение своего программного продукта:

1. Разработанный продукт.

Суммарные затраты на разработку и внедрение были рассчитаны в предыдущем пункте и составили 28 714,82 руб.

1. «Клеверенс: Учёт имущества»

Затраты на внедрение «Клеверенс: Учёт имущества»

Квн = Рпо + Зфот + Рнач + Зэвм

Рпо = 46 050 руб.

Зфот = Зосн \* Здоп \* Твн = Зосн \* Твн = 15 000 руб./мес. \* 24 ч. = (15 000/21/8) \* 24 ч. = 2 142,85 руб. – затраты на оплату работников, занимающихся внедрением проекта.

Рнач = Зфот \* 30% = 2 142,85 руб. \* 0,3 = 642,85 руб. – отчисления во внебюджетные фонды с заработной платы работников.

Зэвм = Тмвн \* *k*и \* *n* \* Cм-ч = 24 ч. \*1 \*1 \* 6,18 руб./ч. = 148,32 руб. – затраты на эксплуатацию ЭВМ при внедрении проекта.

Таблица 6 - Затраты на внедрение «Клеверенс: Учёт имущества»

|  |  |  |
| --- | --- | --- |
| Наименование затрат | Условное обозначение | Значение |
| Стоимость программного продукта | Рпо | 46 050 руб. |
| Фонд оплаты труда | Зфот | 2 142,85 руб. |
| Отчисления во внебюджетные фонды | Рнач | 642,85 руб. |
| Затраты на эксплуатацию ЭВМ | Зэвм | 148,32 руб. |
| Итого | Квн | 48 984,02 руб. |

Таблица 7 - Сравнительная стоимость разрабатываемого программного продукта и существующих решений

|  |  |
| --- | --- |
| Продукт | Стоимость |
| Разрабатываемый программный продукт | 28 714,82 руб. |
| «Клеверенс: Учёт имущества» | 48 984,02 руб. |

Таблица показывает, что разработанное решение является экономически выгодно в 1,7 раза по сравнению с продуктом компании «Клеверенс: Учёт имущества».

# Заключение

Для инвентаризации с помощью RFID-меток было разработано программное обеспечения для работы с УРМ, которое соответствует поставленным требованиям и является экономически эффективным решением по сравнению с существующими аналогами на рынке ПО.

Во время внедрения ПО в лабораторный корпус филиала РТУ МИРЭА в г. Фрязино было проведено тестирование работы ПО, которое показало, что ПО успешно записывает и считывает данные с меток. Так же во время внедрения была создана база данных меток, наклеенных на приборы. Всего было обклеено 104 прибора. Поставленные задачи выполнены в полном объёме.

Таблица 8 – Соответствие полученных данных с ТЗ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Параметр | Значение по ТЗ | Фактическое значение | Соответствие |
| 1 | Язык программирования | C использованием языка C++ | C использованием языка C++ | соответствует |
| 2 | Среда разработки | С использованием  Qt Creator | С использованием  Qt Creator | соответствует |
| 3 | Передача данных между устройством и ПК | Интерфейс RS232 | Интерфейс RS232 | соответствует |
| 4 | Работа с базой данных | С использованием базы данных SQLite | С использованием базы данных SQLite | соответствует |

# Список литературы и интернет источников

* 1. Страуструп Б. Программирование. Принципы и практика с использованием C++ Вильямс, 2016. 1328 c.
  2. Саммерфилд М. Qt Профессиональное программирование. Символ-Плюс, 2011. 552 с.
  3. Макс Шлее. Qt 5.10 Профессиональное программирование на C++. — СПб.: БХВ-Петербург, 2018. — 1072 с.
  4. Документация по языку программирования С++ [Электронный ресурс] – Режим доступа: URL: <https://docs.microsoft.com/ru-ru/cpp/?view=vs-2019> (дата обращения 2019-05-22)
  5. Документация по фреймворку Qt [Электронный ресурс] – Режим доступа: URL: <https://doc.qt.io> (дата обращения 2019-05-22)
  6. Документация по базе данных SQLite. [Электронный ресурс] – Режим доступа: URL: <https://www.sqlite.org/lang.html> (дата обращения 2019-05-22)
  7. Клеверенс: Учёт имущества [Электронный ресурс] – Режим доступа: URL: <https://www.cleverence.ru/software/inventory-software/1C-ASSET-MANAGEMENT/> (дата обращения 2019-05-22)
  8. Современные способы автоматизации процесса инвентаризации [Электронный ресурс] – URL: https://cyberleninka.ru/article/v/sovremennye-sposoby-avtomatizatsii-protsessa-inventarizatsii (дата обращения 2019-02-22)
  9. Решение для инвентаризации товаров на складе на RFID метках [Электронный ресурс] – URL: <https://www.cleverence.ru/solutions/rfid-stock-taking/> (дата обращения 2019-02-22)

# Приложения

**Приложение А**

**Графический материал**

Таблица А.1 – Сравнение ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Особенности | Возможность модификации | Возможность  работы с БД | Возможность записи и считывание данных с RFID-меток | Возможность экспорта базы данных |
| Программное  обеспечение |
| Разрабатываемое ПО | + | + | + | + |
| RADIO Scaner Srv | - | + | + | + |
| Cleverence RFID | - | + | + | + |
| Парус - Бюджет 10 | - | + | + | + |

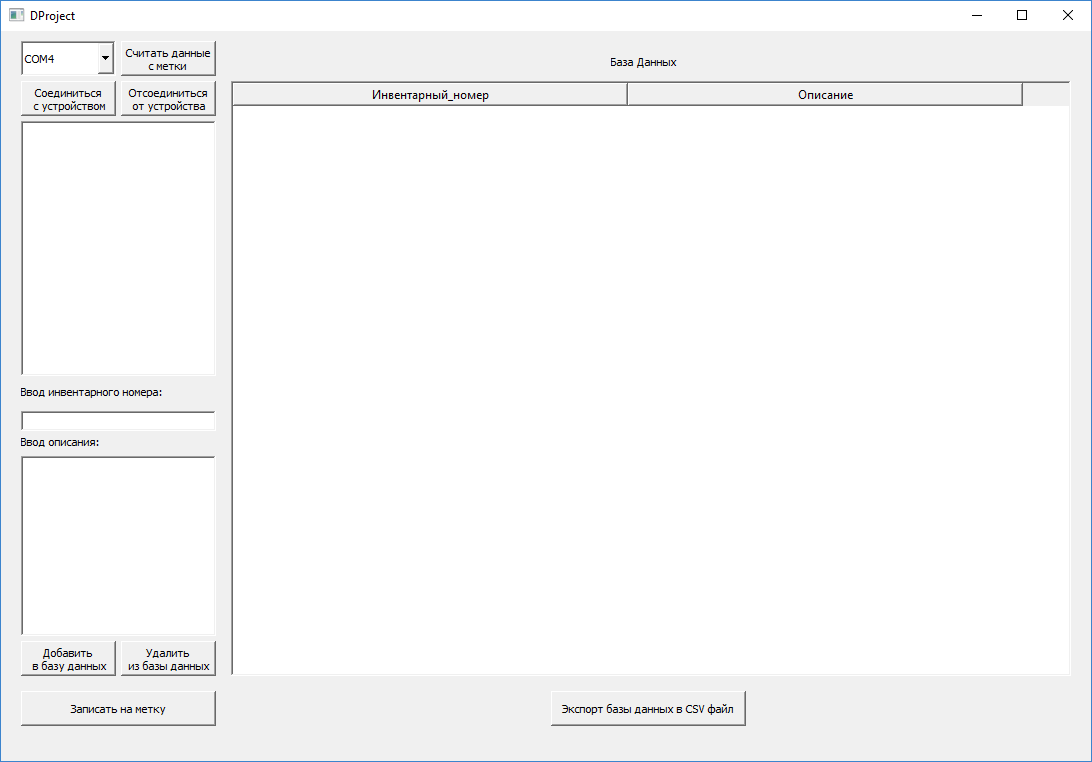
Рисунок А. 1 – Взаимодействие пользователя, приложения, устройство и метки.

Таблица А. 2 – Описание команд

|  |  |  |  |
| --- | --- | --- | --- |
| Название команды | Синтаксис | Описание команды | Возвращаемое значение |
| Запись данных | w[address]:[data] | Записывает данные data начиная с адреса address | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение данных | r[address]:[number] | Считывание определённого числа number данных начиная с адреса address | При успешном завершении возвращает считанные данные. В случае ошибки вернёт код ошибки. |
| Запись версии | v:[data] | Запись версии data метки | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение версии | v | Считывает версию метки | При успешном завершении возвращает считанную версию. В случае ошибки вернёт код ошибки. |
| Запись инвентарного номера | i:[data] | Записывает инвентарный номер data объекта | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение инвентарного номера | i | Считывает инвентарный номер объекта | При успешном завершении возвращает считанный инвентарный номер. В случае ошибки вернёт код ошибки. |
| Запись комментария | c:[data] | Записывает описание data прибора на метку | При успешном завершении возвращает код успешной записи. В случае ошибки вернёт код ошибки. |
| Чтение комментария | c | Считывает описание прибора с метки | При успешном завершение возвращает считанный комментарий. В случае ошибки вернёт код ошибки. |
| Блокировка карты | b | Переключает метку в состояние «Активная», для выполнения нескольких команд | В случае ошибки вернёт код ошибки |
| Разблокировка карты | ub | Переключает метку в состояние «Неактивная». (Использовать эту команду только в случае если метка в состояние «Активная») | В случае ошибки вернёт код ошибки |
| Cправка команд | h | Показывает сведения о командах | Выводит информацию о всех командах |



Рисунок А. 2 – Таблица использованных элементов

Рисунок А. 3 – Общий вид приложения.

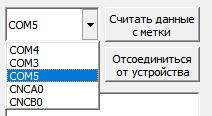
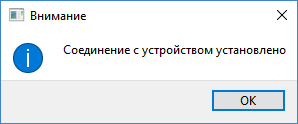


Рисунок А. 4 – Список портов.

Рисунок А. 5 – Оповещение о подключении.

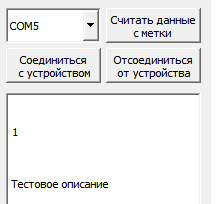


Рисунок А. 6 – Вывод данных в окно.

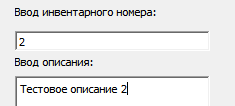


Рисунок А. 7 – Заполненные поля.

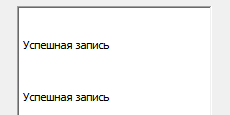


Рисунок А. 8 – Вывод об удачной записи.

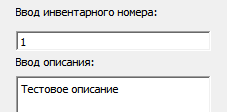


Рисунок А. 9 – Пример ввода данных.

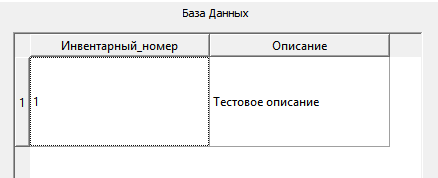


Рисунок А. 10 – Пример базы данных.

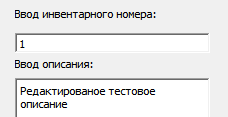


Рисунок А. 11 – Пример вводимых данных.

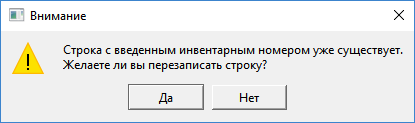


Рисунок А. 12 – Оповещение о перезаписи.

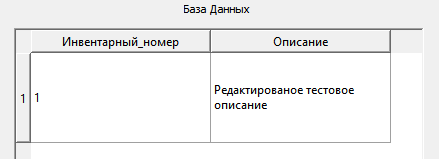


Рисунок А. 13 – Отредактированная строка в базе данных.



Рисунок А. 14 – Ввод инвентарного номера удаляемой строки.

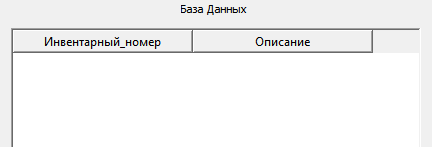


Рисунок А. 15 – База данных после удаления строки.

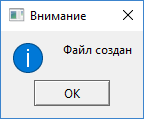
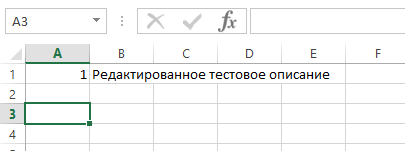
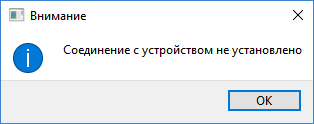
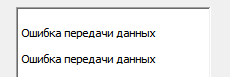
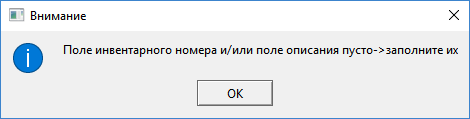


Рисунок А. 16 – Оповещение о создании файла.



Рисунок А. 17 – Экспортированная база данных.

Рисунок А. 18 – Оповещение об отсутствии соединения.

Рисунок А. 19 – Оповещение об ошибки при работе с меткой.

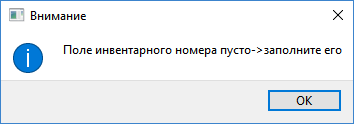
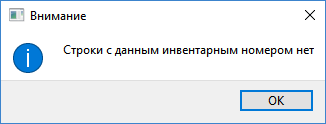
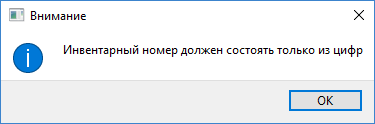
Рисунок А. 20 – Оповещение о пустых полях.

Рисунок А. 21 – Оповещение о пустом поле.

Рисунок А. 22 – Оповещение об отсутствии строки.

Рисунок А. 23 – Оповещение о наличии запрещённых символов.

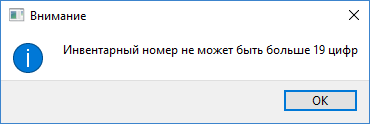


Рисунок А. 24 – Оповещение о превышении длины инвентарного номера.

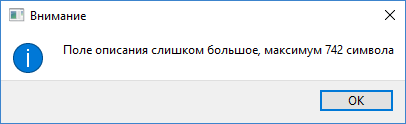
Рисунок А. 25 – Оповещение о превышении длины описания.



Рисунок А. 26 – Источник питания

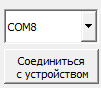
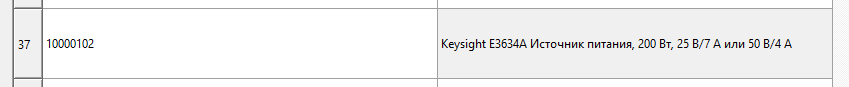


Рисунок А. 27 – Подключение порта

Рисунок А. 28 – Успешное добавление в базу данных

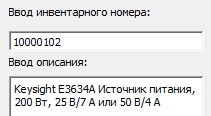


Рисунок А. 29 – Ввод и отправка данных



Рисунок А. 30 – Успешный ответ



Рисунок А. 31 – Ответ ошибки

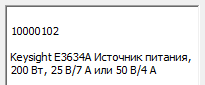
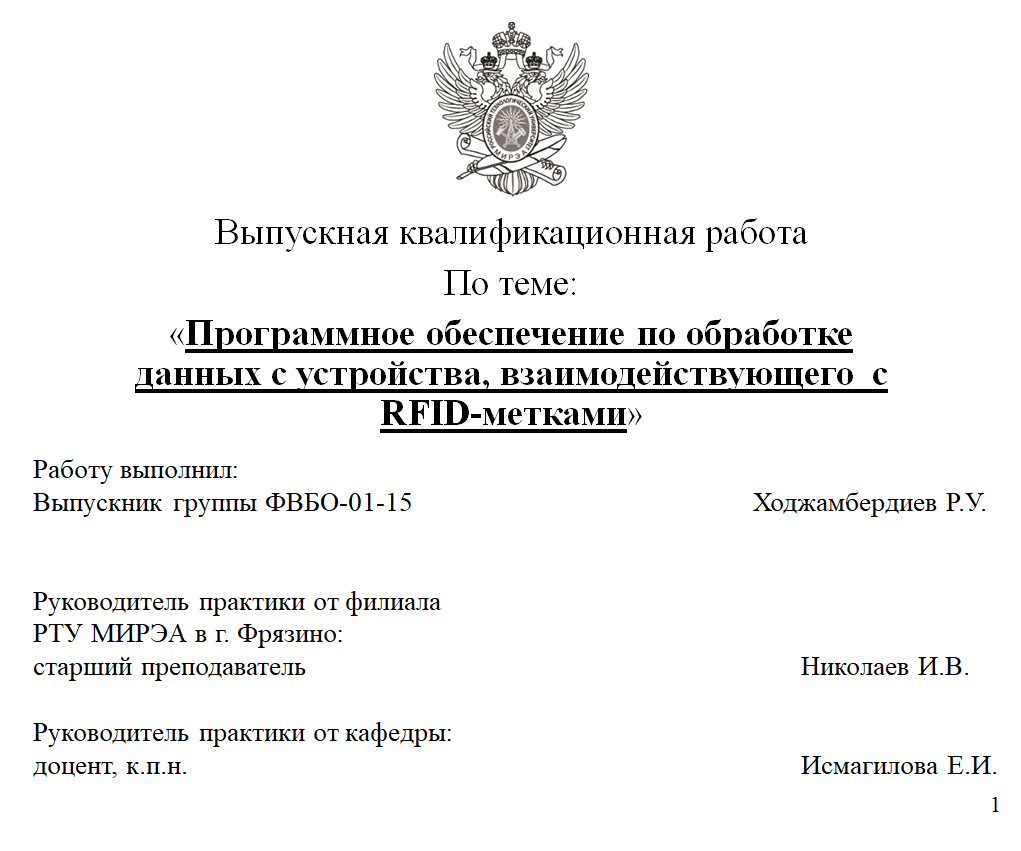


Рисунок А. 32 - Считанные данные



Рисунок А. 33 – Сетевой график

**Приложение Б**

**Презентация**



