Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по творческой работе**

Дисциплина: Основы алгоритмизации и программирования

Тема: Автоматизированное рабочее место

Выполнил работу

студент группы РИС-22-1б

Рыжков Н.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2023

**Описание проекта**

Цель проекта - создание автоматизированного рабочего места оператора АЗС с помощью языка программирования C++ и фреймворка Qt. Программа должна обеспечивать возможность заправки транспортных сред, корректировки цен на топливо, просмотра истории заправок, а также отправки уведомлений на электронную почту при малом количестве топлива в резервуаре и разрыве счетчиков между чеками по резервуару.

Функциональные требования

Заправка

Программа должна обеспечивать возможность заправки транспортных средств. При заправке оператор должен иметь возможность указать следующие параметры:

- Рабочее место

- Вид топлива

- Количество топлива

- Тип оплаты

Корректировка цен на топливо

Программа должна обеспечивать возможность корректировки цен на топливо. Оператор должен иметь возможность изменять цены на топливо в соответствии с текущим рыночными условиями.

Просмотр истории заправок

Программа должна обеспечивать возможность просмотра истории заправок. Оператор должен иметь возможность просмотреть список всех заправок, выполненных на АЗС, а также информацию о каждой заправке, включая дату и время, рабочее место, вид топлива, количество топлива и тип оплаты.

Отправка уведомлений на электронную почту

Программа должна обеспечивать возможность автоматической отправки уведомлений на электронную почту при малом количестве топлива в резервуаре и разрыве счетчиков между чеками по резервуару.

Нефункциональные требования

Интерфейс пользователя

Программа должна иметь интуитивно понятный и простой интерфейс пользователя. Оператор должен легко находить необходимые функции и выполнять операции без дополнительной подготовки.

Производительность

Программа должна обеспечивать высокую производительность и быстрый отклик на запросы оператора. Все операции должны выполняться мгновенно, без задержек и простоев.

Технические требования

Язык программирования

Программа должна быть написана на языке программирования C++.

Фреймворк

Программа должна использовать фреймворк Qt для создания графического интерфейса и обработки пользовательских действий.

Операционная система

Программа должна работать на операционных системах Windows.

Разработка

Этап 1: Проектирование

На этом этапе необходимо разработать дизайн программы и определить ее функциональные и нефункциональные требования.

Этап 2: Разработка интерфейса

На этом этапе необходимо разработать графический интерфейс программы с помощью фреймворка Qt.

Этап 3: Разработка базы данных

На этом этапе необходимо разработать базу данных для хранения информации о заправках.

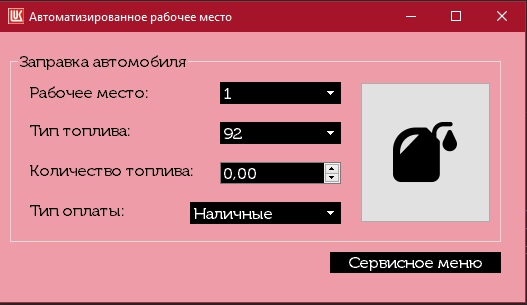
Этап 4: Разработка функциональности

На этом этапе необходимо разработать функциональность программы, включая заправку, корректировку цен на топливо, просмотр истории заправок и отправку уведомлений на электронную почту.

Этап 5: Тестирование

На этом этапе необходимо протестировать программу на соответствие требованиям и выявить, исправить ошибки.

**Дизайн рабочего окна**



**Листинг программы**

#include "mainwindow.h"

#include <QApplication>

#include <QLocale>

#include <QTranslator>

#include <QFontDatabase>

#include <QDialog>

#include <QFormLayout>

#include <QLineEdit>

#include <QPushButton>

#include <QMessageBox>

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

{

QApplication a(argc, argv);

QString fontPath = "MuseoCyrl500.otf";

int id = QFontDatabase::addApplicationFont(fontPath);

if (id != -1) {

QStringList fontFamilies = QFontDatabase::applicationFontFamilies(id);

if (!fontFamilies.isEmpty()) {

QFont font(fontFamilies.at(0));

QApplication::setFont(font);

}

}

QTranslator translator;

const QStringList uiLanguages = QLocale::system().uiLanguages();

for (const QString &locale : uiLanguages) {

const QString baseName = "ARM\_" + QLocale(locale).name();

if (translator.load(":/i18n/" + baseName)) {

a.installTranslator(&translator);

break;

}

}

MainWindow w;

w.show();

return a.exec();

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QFile>

#include <QTextStream>

#include <QStringList>

#include <QDebug>

#include <QDateTime>

#include <QMessageBox>

#include <QStandardItemModel>

#include <QProcess>

#include <QThread>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

#include <QScrollBar>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

struct Refueling {

int check\_number;

QDateTime time;

float check\_fuel1;

float check\_fuel2;

QString fuel\_type;

float fuel\_amount;

float fuel\_price;

float payment\_amount;

QString payment\_type;

QString workstation\_number;

};

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = 0);

~MainWindow();

private slots:

void on\_refuelButton\_clicked();

void on\_resizeButton\_clicked();

void on\_pricesButton\_clicked();

void on\_historyButton\_clicked();

private:

Ui::MainWindow \*ui;

double fuel\_prices[4] = { 45.5, 48.5, 52.0, 47.0 };

int check\_number = 1;

QStandardItemModel \*history\_model;

float check\_fuel1;

float check\_fuel2;

void saveRefuelingToFile(Refueling refueling);

void readRefuelingsFromFile();

float readNumbers(QString fuel\_type, float check\_fuel1);

void saveNumbers(QString fuel\_type, float check\_fuel2);

void sendoil(QString fuel\_type);

void sendoil2(QString fuel\_type);

bool raznica(QString fuel\_type);

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QIcon icon("icon.png"); // создание объекта иконки

setWindowIcon(icon);

QFile check\_file("check\_number.txt");

if (check\_file.open(QIODevice::ReadOnly | QIODevice::Text)) {

QTextStream in(&check\_file);

in >> check\_number;

}

check\_file.close();

history\_model = new QStandardItemModel(this);

history\_model->setColumnCount(10);

history\_model->setHeaderData(0, Qt::Horizontal, "Номер чека");

history\_model->setHeaderData(1, Qt::Horizontal, "Время");

history\_model->setHeaderData(2, Qt::Horizontal, "Начальный счетчик");

history\_model->setHeaderData(3, Qt::Horizontal, "Конечный счетчик");

history\_model->setHeaderData(4, Qt::Horizontal, "Тип топлива");

history\_model->setHeaderData(5, Qt::Horizontal, "Кол-во литров");

history\_model->setHeaderData(6, Qt::Horizontal, "Цена за литр");

history\_model->setHeaderData(7, Qt::Horizontal, "Стоимость заправки");

history\_model->setHeaderData(8, Qt::Horizontal, "Тип оплаты");

history\_model->setHeaderData(9, Qt::Horizontal, "Номер рабочего места");

ui->historyTableView->setModel(history\_model);

ui->historyTableView->horizontalHeader()->setStyleSheet("QHeaderView {color: rgb(0, 0, 0); }");

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_refuelButton\_clicked()

{

QString fuel\_type = ui->fuelTypeComboBox->currentText();

float fuel\_amount = ui->fuelAmountSpinBox->value();

QString payment\_type = ui->paymentTypeComboBox->currentText();

QString workstation\_number = ui->workstationComboBox->currentText();

float fuel\_price;

if (fuel\_type == "92") {

fuel\_price = fuel\_prices[0];

} else if (fuel\_type == "95") {

fuel\_price = fuel\_prices[1];

} else if (fuel\_type == "100") {

fuel\_price = fuel\_prices[2];

} else if (fuel\_type == "DT") {

fuel\_price = fuel\_prices[3];

} else {

QMessageBox::critical(this, "Ошибка", "Неверный тип топлива");

return;

}

check\_fuel1 = readNumbers(fuel\_type, check\_fuel1);

float payment\_amount = fuel\_amount \* fuel\_price;

check\_fuel2 = check\_fuel1 - fuel\_amount;

saveNumbers(fuel\_type, check\_fuel2);

if (check\_fuel2 < 100) {

sendoil(fuel\_type);

}

Refueling refueling;

refueling.check\_number = check\_number;

refueling.time = QDateTime::currentDateTime();

refueling.check\_fuel1 = check\_fuel1;

refueling.check\_fuel2 = check\_fuel2;

refueling.fuel\_type = fuel\_type;

refueling.fuel\_amount = fuel\_amount;

refueling.fuel\_price = fuel\_price;

refueling.payment\_amount = payment\_amount;

refueling.payment\_type = payment\_type;

refueling.workstation\_number = workstation\_number;

saveRefuelingToFile(refueling);

check\_number++;

QFile check\_file("check\_number.txt");

if (check\_file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&check\_file);

out << check\_number;

}

check\_file.close();

QList<QStandardItem\*> row\_items;

row\_items.append(new QStandardItem(QString::number(refueling.check\_number)));

row\_items.append(new QStandardItem(refueling.time.toString()));

row\_items.append(new QStandardItem(QString::number(refueling.check\_fuel1)));

row\_items.append(new QStandardItem(QString::number(refueling.check\_fuel2)));

row\_items.append(new QStandardItem(refueling.fuel\_type));

row\_items.append(new QStandardItem(QString::number(refueling.fuel\_amount)));

row\_items.append(new QStandardItem(QString::number(refueling.fuel\_price)));

row\_items.append(new QStandardItem(QString::number(refueling.payment\_amount)));

row\_items.append(new QStandardItem(refueling.payment\_type));

row\_items.append(new QStandardItem(refueling.workstation\_number));

history\_model->appendRow(row\_items);

bool flazhok = raznica(fuel\_type);

if (flazhok == true) sendoil2(fuel\_type);

QMessageBox::information(this, "Успешно", "Заправка завершена");

}

int initialWidth = 525;

int initialHeight = 270;

bool isResized = false;

void MainWindow::on\_resizeButton\_clicked()

{

if (!isResized) {

resize(1115, 614);

isResized = true;

} else {

resize(initialWidth, initialHeight);

isResized = false;

}

}

void MainWindow::on\_pricesButton\_clicked()

{

fuel\_prices[0] = ui->price92SpinBox->value();

fuel\_prices[1] = ui->price95SpinBox->value();

fuel\_prices[2] = ui->price100SpinBox->value();

fuel\_prices[3] = ui->priceDTSpinBox->value();

QMessageBox::information(this, "Успешно", "Цены на топливо обновлены");

}

void MainWindow::on\_historyButton\_clicked()

{

readRefuelingsFromFile();

}

void MainWindow::saveRefuelingToFile(Refueling refueling)

{

QFile file("history.csv");

if (file.open(QIODevice::Append | QIODevice::Text)) {

QTextStream out(&file);

out << refueling.check\_number << ",";

out << refueling.time.toString("yyyy-MM-dd hh:mm:ss") << ",";

out << refueling.check\_fuel1 << ",";

out << refueling.check\_fuel2 << ",";

out << refueling.fuel\_type << ",";

out << refueling.fuel\_amount << ",";

out << refueling.fuel\_price << ",";

out << refueling.payment\_amount << ",";

out << refueling.payment\_type << ",";

out << refueling.workstation\_number << "\n";

}

file.close();

}

void MainWindow::readRefuelingsFromFile()

{

QFile file("history.csv");

if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {

history\_model->removeRows(0, history\_model->rowCount());

QTextStream in(&file);

while (!in.atEnd()) {

QString line = in.readLine();

QStringList fields = line.split(",");

QList<QStandardItem\*> row\_items;

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

row\_items.append(new QStandardItem(fields.at(i)));

}

history\_model->appendRow(row\_items);

}

}

file.close();

}

float MainWindow::readNumbers(QString fuel\_type, float check\_fuel1) {

QFile check(fuel\_type + ".txt");

if (check.open(QIODevice::ReadOnly | QIODevice::Text)) {

QTextStream in(&check);

in >> check\_fuel1;

}

check.close();

return check\_fuel1;

}

void MainWindow::saveNumbers(QString fuel\_type, float check\_fuel2) {

QFile file(fuel\_type + ".txt");

if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&file);

out << check\_fuel2 << "\n";

}

file.close();

}

void MainWindow::sendoil(QString fuel\_type) {

QFile file("sendmail/debug/replace.txt");

if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&file);

out << fuel\_type << "\n";

}

file.close();

QProcess::startDetached("sendmail/debug/konec.exe");

}

void MainWindow::sendoil2(QString fuel\_type) {

QFile file("sendmail/debug/replace.txt");

if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&file);

out << fuel\_type << "\n";

}

file.close();

QProcess::startDetached("sendmail/debug/project2.exe");

}

std::vector<std::string> split(const std::string& s, char delimiter) {

std::vector<std::string> tokens;

std::string token;

std::istringstream tokenStream(s);

while (std::getline(tokenStream, token, delimiter)) {

tokens.push\_back(token);

}

return tokens;

}

bool MainWindow::raznica(QString fuel\_type) {

std::string filename = "history.csv";

std::string str = fuel\_type.toStdString();

std::ifstream input(filename);

std::ifstream input1(filename);

std::string prover;

bool flag = true;

std::string line;

std::string line1;

while (std::getline(input, line)) {

std::vector<std::string> fields = split(line, ',');

if (fields[4] == str) {

prover = fields[2];

}

}

while (std::getline(input1, line1)) {

std::vector<std::string> fields1 = split(line1, ',');

if (fields1[4] == str) {

if (prover == fields1[3])

flag = false;

}

}

return flag;

}