Разработка виртуального помощника для фрилансеров на C++

Техническое задание

Листов 42

Пермь 2025

Оглавление

[1. Введение 3](#_heading=h.1fob9te)

[2. Основания для разработки 3](#_heading=h.3znysh7)

[3. Назначение 3](#_heading=h.2et92p0)

4. [Т](#_heading=h.3dy6vkm)ребования 4

5. Стадии и этапы разработки 5

6. [Порядок контроля и приемки 42](#_heading=h.2s8eyo1)

1. **Введение**
   1. **Наименование программы**

Наименование программы: “Разработка виртуального помощника для фрилансеров на C++”.

**1.2. Краткая характеристика области применения**

Виртуальный помощник предназначен для упрощения работы фрилансеров, позволяя управлять задачами и проектами через специализированное приложение на компьютере.

1. Основания для разработки виртуального помощника

**2.1. Основания для проведения разработки**

“Разработка виртуального помощника для фрилансеров на C++” разрабатывается в соответствии творческим заданием по разработке автоматизированного рабочего места специалиста. Задание утверждено доцентом кафедры ИТАС Поляковой О. А.

Согласно требованиям Поляковой О. А., именуемой заказчиком, исполнители, Гладков Я. А., Ушаков А. А. и Килин С. В., обязаны разработать систему не позднее 20.05.2025.

1. ****Назначение****
   1. ****Основная цель****

**Разработанный виртуальный помощник для фрилансеров предназначен для автоматизации планирования, учета рабочего времени и контроля финансов. Система предоставляет:**

* **Минимальные требования к ресурсам компьютера.**
* **Контроль рабочего времени.**
* **Управление финансами (учет доходов и расходов).**
  1. ****Решаемые проблемы****
* Ручное ведение учета времени и задач.
* Отсутствие структуры в работе фрилансера.
* Потеря данных при сбоях системы.

****3.3 Области применения****

* Индивидуальные фрилансеры
* Небольшие команды удаленных сотрудников
* Студенты и самозанятые

****3.4 Функциональные возможности****

****Таблица 1.****

|  |  |
| --- | --- |
| **Компонент** | **Функция** |
| **Статистика** | **Предоставляет подробную статистику по выполненным, отмененным и активным задачам.** |
| **Удалить задачу** | **Переносит задачу в раздел отмененных.** |
| **Задача выполнена** | **Переносит задачу в раздел выполненных.** |
| **Добавить задачу** | **Открывает окно для добавления новой задачи с последующим отображением в поле активных задач.** |

1. ****Требования**** 
   1. ****Функциональные требования****

* **Автоматическое распределение задач по приоритетам и срокам.**
* **Генерация статистики (рабочее время, доходы, завершенные проекты).**

****4.2 Технические требования****

*****Аппаратные компоненты*****

****Таблица 2.****

|  |  |
| --- | --- |
| **Компонент** | **Спецификация** |
| **Язык** | **C++ (Qt 6.5+)** |
| **Интерфейс** | **QML + Widgets** |
| **Сборка** | **CMake** |

****4.3 Эксплуатационные требования****

* **ПК: x86\_64, 2 ГБ ОЗУ, 200 МБ дискового пространства**
* **ОС: Windows 10+, Ubuntu 22.04+, macOS 12+**
* **Автономная работа: до 30 дней без обновлений**

1. ****Этапы разработки****

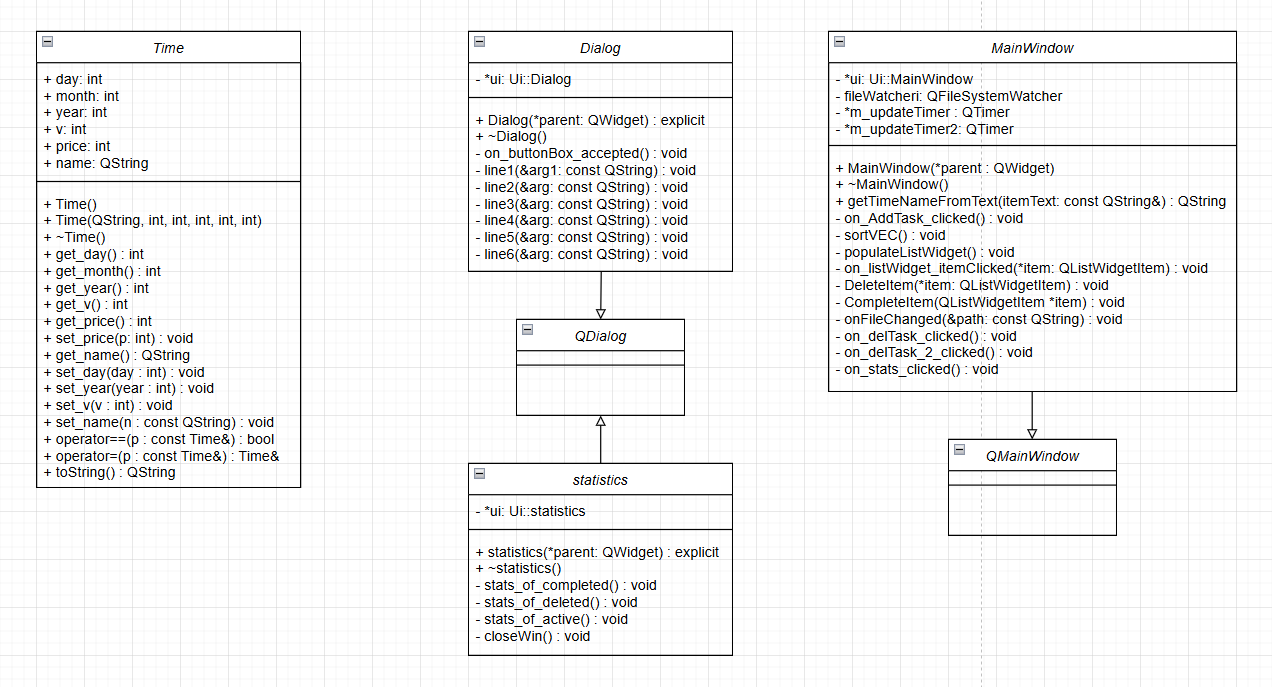
****5.1 Техническое планирование****

* **Разработка UML-диаграмм классов и состояний**
* **Создание прототипов интерфейса в Qt Designer**

****5.2 Реализация****

* **Написание кода**
* **Создание интерфейса**
* **После этого было выполнено тестирование ПО: проверка стабильности работы и корректности передачи данных.**

**UML-диаграммы классов**

****

Программа в Qt

Класс dialog.h

#ifndef DIALOG\_H

#define DIALOG\_H

#include "mainwindow.h"

#include <QDialog>

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

Q\_OBJECT

public:

explicit Dialog(QWidget \*parent = nullptr);

~Dialog();

signals:

void accepted();

private slots:

void on\_buttonBox\_accepted();

void line1(const QString &arg1);

void line2(const QString &arg);

void line3(const QString &arg);

void line4(const QString &arg);

void line5(const QString &arg);

void line6(const QString &arg);

private:

Ui::Dialog \*ui;

};

#endif // DIALOG\_H

Класс mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#ifndef TIME

#define TIME

#include <QMainWindow>

#include <QTextEdit>

#include <Time.h>

#include <QListWidgetItem>

#include <QList>

#include <QFileSystemWatcher>

class Time

{

public:

int day;

int month;

int year;

int v;

int price;

QString name;

public:

Time();

Time(QString, int, int, int, int, int);

~Time();

int get\_day() const;

int get\_month() const;

int get\_year() const;

int get\_v() const;

int get\_price() const;

void set\_price(int p);

QString get\_name()const;

void set\_day(int day);

void set\_month(int month);

void set\_year(int year);

void set\_v(int v);

void set\_name(const QString n);

friend std::fstream& operator>>(std::fstream&, Time&);

friend std::fstream& operator<<(std::fstream&, const Time&);

bool operator==(const Time& p);

Time& operator=(const Time& p);

QString toString() const;

};

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

QString getTimeNameFromText(const QString& itemText);

private slots:

void on\_AddTask\_clicked();

void sortVEC();

void populateListWidget();

void on\_listWidget\_itemClicked(QListWidgetItem \*item);

void DeleteItem(QListWidgetItem \*item);

void CompleteItem(QListWidgetItem \*item);

void onFileChanged(const QString &path);

void on\_delTask\_clicked();

void on\_delTask\_2\_clicked();

void on\_stats\_clicked();

private:

Ui::MainWindow \*ui;

QFileSystemWatcher fileWatcher;

QTimer \*m\_updateTimer;

QTimer \*m\_updateTimer2;

};

#endif // MAINWINDOW\_H

#endif

Класс statistics.h

#ifndef STATISTICS\_H

#define STATISTICS\_H

#include <QDialog>

namespace Ui {

class statistics;

}

class statistics : public QDialog

{

Q\_OBJECT

public:

explicit statistics(QWidget \*parent = nullptr);

~statistics();

private slots:

void stats\_of\_completed();

void stats\_of\_deleted();

void stats\_of\_active();

void closeWin();

private:

Ui::statistics \*ui;

};

#endif // STATISTICS\_H

**dialog.cpp**

**#include "dialog.h"**

**#include "ui\_dialog.h"**

**#include <QFile>**

**#include <QTextStream>**

**Time temp;**

**Dialog::Dialog(QWidget \*parent)**

**: QDialog(parent)**

**, ui(new Ui::Dialog)**

**{**

**ui->setupUi(this);**

**connect(ui->lineEdit, &QLineEdit::textChanged, this, &Dialog::line1);**

**connect(ui->lineEdit\_2, &QLineEdit::textChanged, this, &Dialog::line2);**

**connect(ui->lineEdit\_3, &QLineEdit::textChanged, this, &Dialog::line3);**

**connect(ui->lineEdit\_4, &QLineEdit::textChanged, this, &Dialog::line4);**

**connect(ui->lineEdit\_5, &QLineEdit::textChanged, this, &Dialog::line5);**

**connect(ui->lineEdit\_6, &QLineEdit::textChanged, this, &Dialog::line6);**

**}**

**Dialog::~Dialog()**

**{**

**delete ui;**

**}**

**void Dialog::on\_buttonBox\_accepted()**

**{**

**QFile file("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt");**

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

**QTextStream out(&file);**

**out << "Название: " << temp.get\_name() << '\n';**

**if(temp.get\_day() < 10)**

**{**

**if(temp.get\_month() < 10)**

**out << "Дата сдачи: " << '0' << temp.get\_day() << ".0" << temp.get\_month() << '.' << temp.get\_year() << '\n';**

**else**

**out << "Дата сдачи: " << '0' << temp.get\_day() << '.' << temp.get\_month() << '.' << temp.get\_year() << '\n';**

**}**

**if(temp.get\_month() < 10)**

**out << "Дата сдачи: " << temp.get\_day() << ".0" << temp.get\_month() << '.' << temp.get\_year() << '\n';**

**out << "Дата сдачи: " << temp.get\_day() << '.' << temp.get\_month() << '.' << temp.get\_year() << '\n';**

**out << "Время на выполнение: " << temp.get\_v() << '\n';**

**out << "Стоимость работы: " << temp.get\_price() << '\n';**

**out << '\n';**

**file.close();**

**}**

**void Dialog::line1(const QString &arg1)**

**{**

**temp.set\_name(arg1);**

**}**

**void Dialog::line2(const QString &arg)**

**{**

**QVariant myVariant = arg;**

**int myInt = myVariant.toInt();**

**temp.set\_day(myInt);**

**}**

**void Dialog::line3(const QString &arg)**

**{**

**QVariant myVariant = arg;**

**int myInt = myVariant.toInt();**

**temp.set\_month(myInt);**

**}**

**void Dialog::line4(const QString &arg)**

**{**

**QVariant myVariant = arg;**

**int myInt = myVariant.toInt();**

**temp.set\_year(myInt);**

**}**

**void Dialog::line5(const QString &arg)**

**{**

**QVariant myVariant = arg;**

**int myInt = myVariant.toInt();**

**temp.set\_v(myInt);**

**}**

**void Dialog::line6(const QString &arg)**

**{**

**QVariant myVariant = arg;**

**int myInt = myVariant.toInt();**

**temp.set\_price(myInt);**

**}**

**main.cpp**

**#include "mainwindow.h"**

**#include <QPalette>**

**#include <QApplication>**

**#include <QIcon>**

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

**{**

**QApplication a(argc, argv);**

**MainWindow w;**

**QPalette pal = w.palette();**

**QIcon icon("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\ICON.ico");**

**w.setWindowIcon(icon);**

**pal.setColor(QPalette::Window, QColor(22, 51, 62));**

**w.setPalette(pal);**

**w.setAutoFillBackground(true);**

**w.setWindowTitle("Планировщик");**

**w.show();**

**return a.exec();**

**}**

**mainwindow.cpp**

#include "mainwindow.h"

#include "dialog.h"

#include "ui\_dialog.h"

#include "ui\_statistics.h"

#include "statistics.h"

#include <QTimer>

#include "ui\_mainwindow.h"

#include <QString>

#include <QTextStream>

#include <QFile>

#include <QListWidget>

#include <QTextEdit>

#include <vector>

#include <sstream>

#include <algorithm>

#include <QDebug>

#include <QStringList>

#include <QListWidget>

#include <QListWidgetItem>

#include <QFileSystemWatcher>

#include <QFont>

#include <QIcon>

QList<Time> MyList;

std::vector<Time> result;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QFileSystemWatcher \*fileWatcher = new QFileSystemWatcher(this);

QString filePath = "C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt";

fileWatcher->addPath(filePath);

connect(fileWatcher, &QFileSystemWatcher::fileChanged, this, &MainWindow::onFileChanged);

ui->listWidget->setSelectionMode(QAbstractItemView::ExtendedSelection);

connect(ui->listWidget, &QListWidget::itemClicked, this, &MainWindow::on\_listWidget\_itemClicked);

connect(ui->delTask, &QPushButton::clicked, this, &MainWindow::on\_delTask\_clicked);

connect(ui->delTask\_2, &QPushButton::clicked, this, &MainWindow::on\_delTask\_2\_clicked);

sortVEC();

populateListWidget();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_AddTask\_clicked()

{

Dialog \*Win = new Dialog;

QPalette pal = Win->palette();

QIcon icon("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\ICON.ico");

Win->setWindowIcon(icon);

pal.setColor(QPalette::Window, QColor(22, 51, 62));

Win->setPalette(pal);

Win->setAutoFillBackground(true);

Win->setWindowTitle("Добавление задачи");

Win->show();

}

void MainWindow::populateListWidget() {

ui->listWidget->clear();

for (const Time& time : MyList) {

QListWidgetItem\* item = new QListWidgetItem();

item->setText(time.toString());

ui->listWidget->addItem(item);

}

}

bool proverca(const Time& a, const Time& b) {

if (a.get\_price() != b.get\_price()) {

return a.get\_price() > b.get\_price();

}

if (a.get\_year() != b.get\_year()) {

return a.get\_year() < b.get\_year();

}

if (a.get\_month() != b.get\_month()) {

return a.get\_month() < b.get\_month();

}

if (a.get\_day() != b.get\_day()) {

return a.get\_day() < b.get\_day();

}

return a.get\_v() > b.get\_v();

}

void sortirovka(std::vector<Time>& times) {

bool swapped;

for (size\_t i = 0; i < times.size() - 1; i++) {

swapped = false;

for (size\_t j = 0; j < times.size() - i - 1; j++) {

if (proverca(times[j + 1], times[j])) {

std::swap(times[j], times[j + 1]);

swapped = true;

}

}

if (!swapped) break;

}

}

void MainWindow::sortVEC()

{

result.clear();

QFile file("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt");

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

qDebug() << "Could not open file for reading:" << file.fileName();

return;

}

QTextStream in(&file);

in.setEncoding(QStringConverter::Utf8);

if(file.size() == 0)

{

file.close();

populateListWidget();

return;

}

QString name;

int day = 0, month = 0, year = 0, v = 0, price = 0;

bool validRecord = false;

while (!in.atEnd()) {

QString line = in.readLine();

if (line.startsWith("Название:")) {

name = line.mid(line.indexOf(":") + 2).trimmed();

validRecord = true;

} else if (line.startsWith("Дата сдачи:")) {

QString dateString = line.mid(line.indexOf(":") + 2).trimmed();

QStringList dateParts = dateString.split('.');

if (dateParts.size() == 3) {

bool ok;

day = dateParts[0].toInt(&ok);

if (!ok) {

qDebug() << "Invalid day format: " << dateParts[0];

validRecord = false;

continue;

}

month = dateParts[1].toInt(&ok);

if (!ok) {

qDebug() << "Invalid month format: " << dateParts[1];

validRecord = false;

continue;

}

year = dateParts[2].toInt(&ok);

if (!ok) {

qDebug() << "Invalid year format: " << dateParts[2];

validRecord = false;

continue;

}

} else {

qDebug() << "Неверный формат даты: " << dateString;

validRecord = false;

continue;

}

} else if (line.startsWith("Время на выполнение:")) {

bool ok;

v = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid v format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

} else if (line.startsWith("Стоимость работы:")) {

bool ok;

price = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid price format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

if (validRecord) {

Time task(name, day, month, year, v, price);

result.push\_back(task);

validRecord = false;

}

}

}

MyList.clear();

for (const Time& time : result) {

MyList.append(time);

}

file.close();

sortirovka(result);

QFile file1("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt");

if (!file1.open(QIODevice::WriteOnly | QIODevice::Text)) {

qDebug() << "Не удалось открыть файл для записи!";

return;

}

QTextStream out(&file1);

out.setEncoding(QStringConverter::Utf8);

for (const Time& t : result)

{

out << "Название: " << t.name << '\n';

out << "Дата сдачи: " << QString("%1.%2.%3").arg(t.day, 2, 10, QChar('0')).arg(t.month, 2, 10, QChar('0')).arg(t.year) << '\n';

out << "Время на выполнение: " << t.v << '\n';

out << "Стоимость работы: " << t.price << '\n';

out << '\n';

}

file1.close();

populateListWidget();

}

void MainWindow::on\_listWidget\_itemClicked(QListWidgetItem \*item)

{

if (ui->listWidget->selectedItems().contains(item)) {

} else {

item->setSelected(true);

}

}

void MainWindow::DeleteItem(QListWidgetItem \*item)

{

if (!item) {

return;

}

QString itemText = item->text();

QString filePath = "C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt";

QFile file(filePath);

if (!file.open(QIODevice::ReadWrite | QIODevice::Text)) {

qDebug() << "Не удалось открыть файл для чтения/записи: " << filePath;

return;

}

QTextStream in(&file);

in.setEncoding(QStringConverter::Utf8);

QString fileContent;

while (!in.atEnd()) {

QString line = in.readLine();

fileContent.append(line);

fileContent.append("\n");

}

file.close();

QString patternString =

"Название: " + QRegularExpression::escape(getTimeNameFromText(itemText)) + ".\*?" +

"Дата сдачи: .\*?\\n" +

"Время на выполнение: .\*?\\n" +

"Стоимость работы: .\*?\\n" +

"(?:\\n|$)";

QRegularExpression pattern(patternString, QRegularExpression::DotMatchesEverythingOption | QRegularExpression::MultilineOption);

QRegularExpressionMatch match = pattern.match(fileContent);

QString matchedText = match.captured(0);

QFile delf("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\deletedTasks.txt");

delf.open(QIODevice::Append);

QTextStream outD(&delf);

outD << matchedText;

delf.close();

if (match.hasMatch()) {

fileContent.remove(match.capturedStart(), match.capturedLength());

} else {

qDebug() << "Элемент не найден в файле: " << itemText;

return;

}

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

qDebug() << "Не удалось открыть файл для записи: " << filePath;

return;

}

QTextStream out(&file);

out.setEncoding(QStringConverter::Utf8);

out << fileContent;

file.close();

}

QString MainWindow::getTimeNameFromText(const QString& itemText)

{

int nameStart = itemText.indexOf("Название: ") + QString("Название: ").length();

if (nameStart != -1) {

int nameEnd = itemText.indexOf("\n", nameStart);

if (nameEnd != -1) {

return itemText.mid(nameStart, nameEnd - nameStart).trimmed();

} else {

return itemText.mid(nameStart).trimmed();

}

}

return "";

}

void MainWindow::onFileChanged(const QString &path)

{

Q\_UNUSED(path);

sortVEC();

}

void MainWindow::on\_delTask\_clicked()

{

QListWidgetItem \*selectedItem = ui->listWidget->currentItem();

DeleteItem(selectedItem);

if (selectedItem)

DeleteItem(selectedItem);

}

void MainWindow::CompleteItem(QListWidgetItem \*item)

{

if (!item) {

return;

}

QString itemText = item->text();

QString filePath = "C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\tasks.txt";

QFile file(filePath);

if (!file.open(QIODevice::ReadWrite | QIODevice::Text)) {

qDebug() << "Не удалось открыть файл для чтения/записи: " << filePath;

return;

}

QTextStream in(&file);

in.setEncoding(QStringConverter::Utf8);

QString fileContent;

while (!in.atEnd()) {

QString line = in.readLine();

fileContent.append(line);

fileContent.append("\n");

}

file.close();

QString patternString =

"Название: " + QRegularExpression::escape(getTimeNameFromText(itemText)) + ".\*?" +

"Дата сдачи: .\*?\\n" +

"Время на выполнение: .\*?\\n" +

"Стоимость работы: .\*?\\n" +

"(?:\\n|$)";

QRegularExpression pattern(patternString, QRegularExpression::DotMatchesEverythingOption | QRegularExpression::MultilineOption);

QRegularExpressionMatch match = pattern.match(fileContent);

QString matchedText = match.captured(0);

QFile delf("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\completedTasks.txt");

delf.open(QIODevice::Append);

QTextStream outD(&delf);

outD << matchedText;

delf.close();

if (match.hasMatch()) {

fileContent.remove(match.capturedStart(), match.capturedLength());

} else {

qDebug() << "Элемент не найден в файле: " << itemText;

return;

}

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

qDebug() << "Не удалось открыть файл для записи: " << filePath;

return;

}

QTextStream out(&file);

out.setEncoding(QStringConverter::Utf8);

out << fileContent;

file.close();

}

void MainWindow::on\_delTask\_2\_clicked()

{

QListWidgetItem \*selectedItem1 = ui->listWidget->currentItem();

CompleteItem(selectedItem1);

if (selectedItem1)

DeleteItem(selectedItem1);

}

void MainWindow::on\_stats\_clicked()

{

statistics \*stats = new statistics;

QPalette pal = stats->palette();

QIcon icon("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\ICON.ico");

stats->setWindowIcon(icon);

pal.setColor(QPalette::Window, QColor(22, 51, 62));

stats->setPalette(pal);

stats->setAutoFillBackground(true);

stats->setWindowTitle("Статистика");

stats->show();

}

statistics.cpp

#include "statistics.h"

#include "ui\_statistics.h"

#include <QFile>

#include "mainwindow.h"

#include <vector>

statistics::statistics(QWidget \*parent)

: QDialog(parent)

, ui(new Ui::statistics)

{

ui->setupUi(this);

stats\_of\_completed();

stats\_of\_deleted();

stats\_of\_active();

connect(ui->closeB, &QPushButton::clicked, this, &statistics::closeWin);

}

statistics::~statistics()

{

delete ui;

}

void statistics::stats\_of\_completed()

{

std::vector<Time> com;

QFile comp("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\completedTasks.txt");

comp.open(QIODevice::ReadOnly);

QString name;

int day = 0, month = 0, year = 0, v = 0, price = 0;

bool validRecord = false;

while (!comp.atEnd()) {

QString line = comp.readLine();

if (line.startsWith("Название:")) {

name = line.mid(line.indexOf(":") + 2).trimmed();

validRecord = true;

} else if (line.startsWith("Дата сдачи:")) {

QString dateString = line.mid(line.indexOf(":") + 2).trimmed();

QStringList dateParts = dateString.split('.');

if (dateParts.size() == 3) {

bool ok;

day = dateParts[0].toInt(&ok);

if (!ok) {

qDebug() << "Invalid day format: " << dateParts[0];

validRecord = false;

continue;

}

month = dateParts[1].toInt(&ok);

if (!ok) {

qDebug() << "Invalid month format: " << dateParts[1];

validRecord = false;

continue;

}

year = dateParts[2].toInt(&ok);

if (!ok) {

qDebug() << "Invalid year format: " << dateParts[2];

validRecord = false;

continue;

}

} else {

qDebug() << "Неверный формат даты: " << dateString;

validRecord = false;

continue;

}

} else if (line.startsWith("Время на выполнение:")) {

bool ok;

v = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid v format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

} else if (line.startsWith("Стоимость работы:")) {

bool ok;

price = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid price format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

if (validRecord) {

Time task(name, day, month, year, v, price);

com.push\_back(task);

validRecord = false;

}

}

}

comp.close();

double sredPrice = 0;

double sredDayPrice = 0;

int totalPrice = 0;

int totalDays = 0;

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

{

totalPrice += com[i].get\_price();

totalDays += com[i].get\_v();

}

sredPrice = totalPrice / com.size();

sredDayPrice = totalPrice / totalDays;

QString text;

text += "Количество выполненных задач: " + QString::number(com.size()) + "\n";

text += "Заработано всего: " + QString::number(totalPrice, 'f', 2) + "\n";

text += "Дней потрачено на работу: " + QString::number(totalDays) + "\n";

text += "Заработано в среднем за задачу: " + QString::number(sredPrice, 'f', 2) + "\n";

text += "Заработано в среднем в день: " + QString::number(sredDayPrice, 'f', 2) + "\n";

ui->textEdit->setText(text);

}

void statistics::stats\_of\_deleted()

{

std::vector<Time> del;

QFile delet("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\deletedTasks.txt");

delet.open(QIODevice::ReadOnly);

QString name;

int day = 0, month = 0, year = 0, v = 0, price = 0;

bool validRecord = false;

while (!delet.atEnd()) {

QString line = delet.readLine();

if (line.startsWith("Название:")) {

name = line.mid(line.indexOf(":") + 2).trimmed();

validRecord = true;

} else if (line.startsWith("Дата сдачи:")) {

QString dateString = line.mid(line.indexOf(":") + 2).trimmed();

QStringList dateParts = dateString.split('.');

if (dateParts.size() == 3) {

bool ok;

day = dateParts[0].toInt(&ok);

if (!ok) {

qDebug() << "Invalid day format: " << dateParts[0];

validRecord = false;

continue;

}

month = dateParts[1].toInt(&ok);

if (!ok) {

qDebug() << "Invalid month format: " << dateParts[1];

validRecord = false;

continue;

}

year = dateParts[2].toInt(&ok);

if (!ok) {

qDebug() << "Invalid year format: " << dateParts[2];

validRecord = false;

continue;

}

} else {

qDebug() << "Неверный формат даты: " << dateString;

validRecord = false;

continue;

}

} else if (line.startsWith("Время на выполнение:")) {

bool ok;

v = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid v format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

} else if (line.startsWith("Стоимость работы:")) {

bool ok;

price = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid price format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

if (validRecord) {

Time task(name, day, month, year, v, price);

del.push\_back(task);

validRecord = false;

}

}

}

delet.close();

double sredPrice = 0;

double sredDayPrice = 0;

int totalPrice = 0;

int totalDays = 0;

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

{

totalPrice += del[i].get\_price();

totalDays += del[i].get\_v();

}

sredPrice = totalPrice / del.size();

sredDayPrice = totalPrice / totalDays;

QString text1;

text1 += "Количество отмененных задач: " + QString::number(del.size()) + "\n";

text1 += "Не заработано: " + QString::number(totalPrice, 'f', 2) + "\n";

text1 += "Освобожденных дней: " + QString::number(totalDays) + "\n";

ui->textEdit\_2->setText(text1);

}

void statistics::stats\_of\_active()

{

std::vector<Time> act;

QFile activeTasks("C:\\Users\\Professional\\Desktop\\TvorcheskayaRabota\\Tasks.txt");

activeTasks.open(QIODevice::ReadOnly);

if(activeTasks.size() == 0)

{

activeTasks.close();

return;

}

QString name;

int day = 0, month = 0, year = 0, v = 0, price = 0;

bool validRecord = false;

while (!activeTasks.atEnd()) {

QString line = activeTasks.readLine();

if (line.startsWith("Название:")) {

name = line.mid(line.indexOf(":") + 2).trimmed();

validRecord = true;

} else if (line.startsWith("Дата сдачи:")) {

QString dateString = line.mid(line.indexOf(":") + 2).trimmed();

QStringList dateParts = dateString.split('.');

if (dateParts.size() == 3) {

bool ok;

day = dateParts[0].toInt(&ok);

if (!ok) {

qDebug() << "Invalid day format: " << dateParts[0];

validRecord = false;

continue;

}

month = dateParts[1].toInt(&ok);

if (!ok) {

qDebug() << "Invalid month format: " << dateParts[1];

validRecord = false;

continue;

}

year = dateParts[2].toInt(&ok);

if (!ok) {

qDebug() << "Invalid year format: " << dateParts[2];

validRecord = false;

continue;

}

} else {

qDebug() << "Неверный формат даты: " << dateString;

validRecord = false;

continue;

}

} else if (line.startsWith("Время на выполнение:")) {

bool ok;

v = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid v format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

} else if (line.startsWith("Стоимость работы:")) {

bool ok;

price = line.mid(line.indexOf(":") + 2).trimmed().toInt(&ok);

if (!ok) {

qDebug() << "Invalid price format: " << line.mid(line.indexOf(":") + 2).trimmed();

validRecord = false;

continue;

}

if (validRecord) {

Time task(name, day, month, year, v, price);

act.push\_back(task);

validRecord = false;

}

}

}

activeTasks.close();

double sredPrice = 0;

double sredDayPrice = 0;

int totalPrice = 0;

int totalDays = 0;

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

{

totalPrice += act[i].get\_price();

totalDays += act[i].get\_v();

}

sredPrice = totalPrice / act.size();

sredDayPrice = totalPrice / totalDays;

QString text2;

text2 += "Количество задач: " + QString::number(act.size()) + "\n";

text2 += "Будет заработано: " + QString::number(totalPrice, 'f', 2) + "\n";

text2 += "Необходимо дней на работу: " + QString::number(totalDays) + "\n";

text2 += "Заработок за все задачи: " + QString::number(sredPrice, 'f', 2) + "\n";

text2 += "Будет заработано в среднем в день: " + QString::number(sredDayPrice, 'f', 2) + "\n";

ui->textEdit\_3->setText(text2);

}

void statistics::closeWin()

{

close();

}

Time.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "mainwindow.h"

#include <cmath>

#include <ctime>

#include <algorithm>

Time::Time() {

time\_t timer = time(0);

tm\* now = localtime(&timer);

year = now->tm\_year + 1900;

month = now->tm\_mon + 1;

day = now->tm\_mday;

v = 0;

name = " ";

price = 0;

}

Time::Time(QString name1, int day1, int month1, int year1, int v1, int price1)

{

name = name1;

day = day1;

month = month1;

year = year1;

v = v1;

price = price1;

}

Time::~Time() {}

QString Time::get\_name() const

{

return name;

}

int Time::get\_day() const

{

return day;

}

int Time::get\_month() const

{

return month;

}

int Time::get\_year() const

{

return year;

}

int Time::get\_v() const

{

return v;

}

int Time::get\_price() const

{

return price;

}

void Time::set\_day(int d) {

day = d;

}

void Time::set\_month(int m) {

month = m;

}

void Time::set\_year(int y) {

year = y;

}

void Time::set\_v(int V) {

v = V;

}

void Time::set\_name(QString n)

{

name = n;

}

void Time::set\_price(int p)

{

price = p;

}

bool Time::operator==(const Time& p)

{

if (day == p.day && month == p.month && year == p.year)

{

return true;

}

else {

return false;

}

}

Time& Time::operator=(const Time& p) {

day = p.day;

month = p.month;

year = p.year;

name = p.name;

v = p.v;

price = p.price;

return \*this;

}

QString Time::toString() const { //Метод для удобного преобразования в строку

if(day < 10)

{

if(month < 10) {

return QString("Название: %1\nДата сдачи: 0%2/0%3/%4\nВремя на выполнение: %5\nСтоимость: %6")

.arg(name)

.arg(day)

.arg(month)

.arg(year)

.arg(v)

.arg(price);

}

else {

return QString("Название: %1\nДата сдачи: 0%2/%3/%4\nВремя на выполнение: %5\nСтоимость: %6")

.arg(name)

.arg(day)

.arg(month)

.arg(year)

.arg(v)

.arg(price);

}

}

if(month < 10)

{

return QString("Название: %1\nДата сдачи: %2/0%3/%4\nВремя на выполнение: %5\nСтоимость: %6")

.arg(name)

.arg(day)

.arg(month)

.arg(year)

.arg(v)

.arg(price);

}

return QString("Название: %1\nДата сдачи: %2/%3/%4\nВремя на выполнение: %5\nСтоимость: %6")

.arg(name)

.arg(day)

.arg(month)

.arg(year)

.arg(v)

.arg(price);

}

1. Порядок приёма

Приемосдаточные испытания системы должны проводиться на кафедре ИТАС ПНИПУ в сроки с 22.05.2025 до указанных заказчиком.