СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день, стремительно развиваются информационные технологии, которые улучшают условия жизни людей. Значительно повышается уровень знаний, в связи с этим становится необходимым эффективно организовывать, сохранять и управлять доступом к ним.

Курсовая работа посвящена разработке приложения «Телефонный справочник организаций и жителей города», являющейся оболочкой для работы с базой данных телефонных абонентов и организаций. Приложение должно предоставлять пользователю средства для просмотра базы абонентов и организаций, ее редактирования и поиска по базе. Поэтому наиболее важными алгоритмами для проекта являются алгоритмы поиска и сортировки. Выбор подходящих алгоритмов поиска и сортировки основаны на простоте реализации и эффективности работы в программе.

В рамках данной работы необходимо овладение практическими навыками проектирования и разработки законченного, отлаженного и протестированного программного продукта с использованием методик объектно-ориентированного проектирования и языка высокого уровня C++. Закрепить теоретические знания, полученные при изучении курса “Основы алгоритмизации и программирования”. Также необходимо осуществить разработку удобного пользовательского интерфейса.

Помимо практического интереса, тема имеет широкие возможности для последующей модернизации проекта с применением навыков, полученных в ходе изучения курса “Компьютерное проектирование и языки программирования”.

Данная тема является актуальной, так как разработанный программный продукт предоставляет пользователю возможность получить информацию о всех номерах организаций и жителей города в доступной форме.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ аналогов программного средства

Справочник организаций и жителей вашего города с наиболее полной информацией о каждой организации, службе и компании поможет пользователю быстро найти нужный адрес, номер телефона и другую информацию.

Ни одна информационная печатная продукция не может справиться с объёмом данных, а тем более с их постоянными изменениями. Номер телефона может измениться, компания может переехать или изменить режим работы, а в бумажных справочниках по-прежнему будет содержаться неактуальная информация. Но при этом аналогичные приложения уже технически и визуально устарели. Пример такого телефонного справочника приведет на рисунке 1.1:

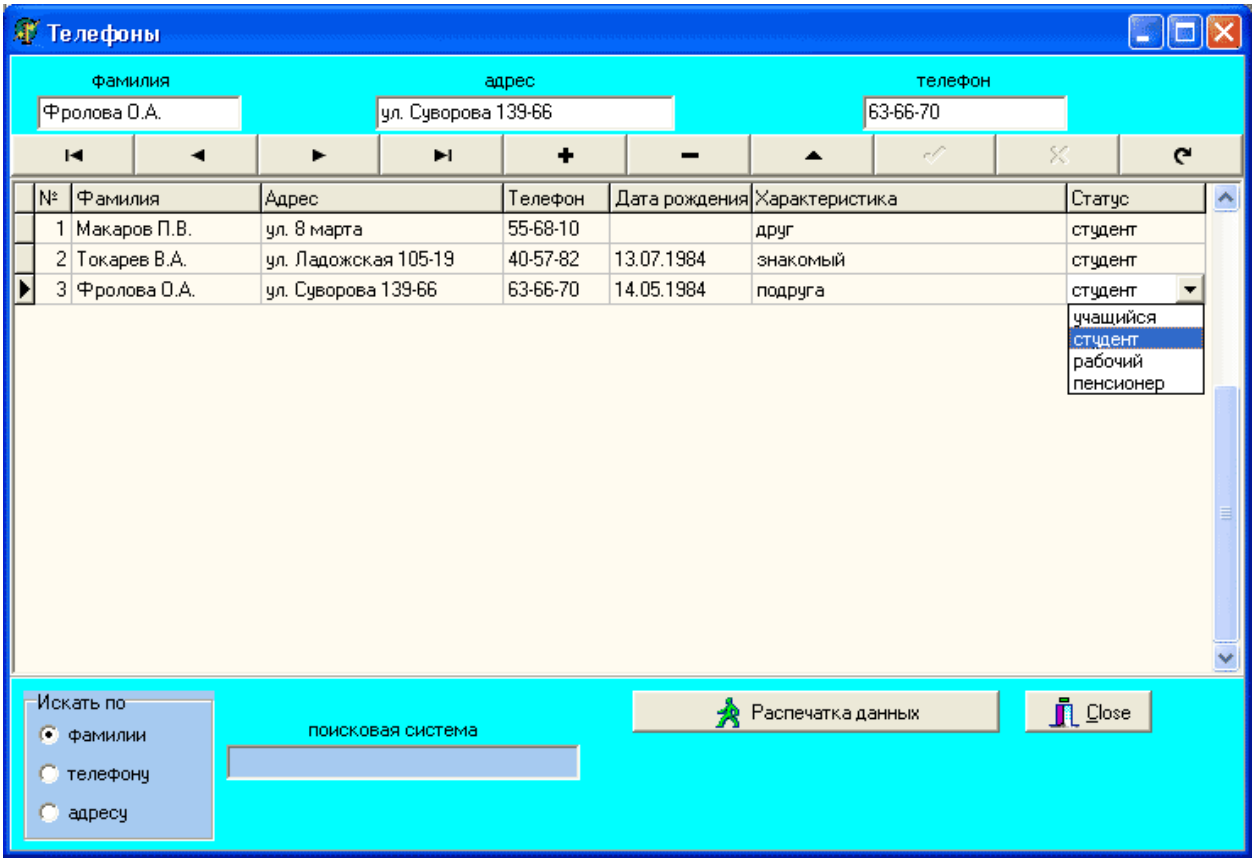

Рисунок 1.1 – Скриншот устаревшего приложения “Справочник”

Новое приложение справочник помогает избежать этого. В интернете обновить данные значительно легче, как и добавить в список новую организацию, только что открывшуюся. Это особенно актуально, ведь бизнес разрастается очень быстро. А новый и удобный интерфейс позволит пользователю быстрее сориентироваться в приложении.

Нередки случаи, когда необходимо срочно найти номер телефона или адрес конкретной компании. На поиски в бумажном носителе может уйти масса времени, но приложение справочник облегчает поиски и экономит время. Пользователь найдёт нужные службы в городе и информацию по ним, прямо не выходя из дома.

Исходя из всего этого можно сделать вывод, что найти нужный номер телефона или адрес пользователю удастся в несколько кликов. Больше не нужно тратить время на поиски ресурса со справочной информацией, всё что вам нужно, будет в одном приложении.

1.2 Постановка задачи

Программа должна иметь графический интерфейс, для удобного взаимодействия пользователя с программой.

В программе будут реализованы главные методы:

- Ввод данных.
- Сортировка данных.
- Поиск данных.
- Вывод данных
- Хранение данных
- Редактирование данных

Для реализации данного программного обеспечения используется объектно-ориентированный язык программирования C++, среда разработки Visual Studio 2019. Для реализации графического интерфейса используется кроссплатформенная IDE – Qt.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения ее следует разбить на функциональные блоки. Такой подход упростит понимание системы, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость системы в будущем путем добавления новых блоков.

2.1 Модуль основной работы приложения

Модуль основной работы приложения предназначен для манипуляций с данными и взаимодействием с ними вне зрения пользователя.

Таким образом модуль основной работы приложения должен следующие функции:

- Сортировка данных
- Поиска данных.
- Редактирование данных.

2.2 Модуль хранения данных

Модуль хранения данных предназначен для хранения данных о жителях и организациях города.

Главный метод базы данных будет выполнять следующую функцию:

- Хранение данных

2.3 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предназначен для взаимодействия пользователя с компьютером, основываясь на представлении и визуализации данных.

Для разрабатываемого проекта создается удобный и понятный пользовательский интерфейс, для реализации которого используется кроссплатформенная IDE – Qt.

Основные методы реализованные в модуле пользовательского интерфейса:

- Ввод данных.
- Вывод данных.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемой программы.

3.1 Описание классов приложения

3.1.1 Contact

Класс содержит в себе все основные данные справочника.

Поля:

- `string _id` – уникальный id.
- `String _phoneNumber` – телефонный номер жителя или организации.
- `string _country` – название страны.
- `string _city` – название города, деревни и т.д.
- `string _address` – адрес жителя или организации.
- `string _comment` – комментарий.

Методы:

- `void setPhoneNumber(string phoneNumber)` – инициализация переменной `_phoneNumber`.
- `void setCountry(string country)` – инициализация переменной `_country`.
- `void setCity(string city)` – инициализация переменной `_city`.
- `void setAddress(string address)` –инициализация переменной `_address`.
- `void setComment(string comment)` - инициализация переменной `_comment`.
- `void setFileId()` - инициализация переменной `_id` из модуля хранения данных.
- `string getPhoneNumber()` – возвращает `_phoneNumber`.
- `string getCountry()` – возвращает `_country`.
- `string getCity()` – возвращает `_city`.
- `string getAddress()` – возвращает `_address`.
- `string getComment()` – возвращает `_comment`.
- `string setId()` - инициализирует переменную `_id` сгенерированным уникальным id.

– `string getId()` – возвращает `_id`.

– `Contact ()` – конструктор по умолчанию.

– `Contact(string id, string number, string country, string city, string address, string comment)` - конструктор инициализации.

– `~ Contact ()` – деструктор.

3.1.2 PersonalContact: virtual public Contact

Класс содержит основную информацию, которая относится только к жителям.

Поля:

– `string _firstname` – имя жителя.

– `string _lastname` – фамилия жителя.

Методы:

– `void setFirstname(string firstname)` – инициализация переменной `_firstname`.

– `void setLastname(string lastname)` – инициализация переменной `_lastname`.

– `string getFullName()` - возвращает `_firstname` и `_lastname`.

– `string getLastName()` – возвращает `_firstname`.

– `string getFirstName()` – возвращает `_lastname`.

– `PerosnalContact()` – конструктор по умолчанию.

– `PerosnalContact(string firstname, string lastname)` – конструктор инициализации.

– `~PerosnalContact ()` – деструктор.

3.1.3 OrganisationContact: virtual public Contact

Класс содержит основную информацию, которая относится только к организациям.

Поля:

– `string _nameOrg` – название организации.

Методы:

– `void setNameOrg(string nameOrg)` – инициализация переменной `_nameOrg`.

– `string getNameOrg` - возвращает `_nameOrg`.

– `OrganisationContact ()` – конструктор по умолчанию.

– `OrganisationContact (string nameOrg)` – конструктор инициализации.

– `~OrganisationContact ()` – деструктор.

3.1.4 ContactRepository

Класс хранит всю информацию о контактах и взаимодействует с ней

Поля:

– `vector <OrganisationContact> vectorOrgContact` – вектор, хранящий объекты типа `OrganisationContact.`

– `vector <PersonalContact> vectorPerContact` – вектор, хранящий объекты типа `PersonalContact.`

– `Vector <Contact> vector` – вектор, хранящий объекты типа `Contact.`

Методы:

– `void CreateContact(Contact a)` – создает объект типа `Contact.`

– `void CreateOrgContact(OraganisationContact a)` - создает контакт типа `OraganisationContact.`

– `void CreatePerContact(PersonalContact a)` – создает контакт типа `PersonalContact.`

– `OrganisationContact searchOrgContact(string search, unsigned long long i)` - ищет нужный контакт по ключевому слову `search`, если найдет – возвращает объект типа `OrganisationContact.`

– `PersonalContact searchPerContact(string search, unsigned long long i)` – ищет нужный контакт по ключевому слову `search`, если найдет – возвращает объект типа `OrganisationContact.`

– `void Delete (string id, int category)` – удаляет контакт с определенным id.

– `Void searchContactToUpdate (string search)` - находит контакт, который пользователь собрался обновить.

– `int getVecSizeContact()` – возвращает общее количество элементов, которые содержатся в векторах `vectorPerContact` и `vectorOrgContact`

– `Int getVecSizePerContact()` – возвращает количество элементов, которые содержатся в векторе `vectorPerContact`

– `Int getVecSizeOrgContact()` – возвращает количество элементов, которые содержатся в векторе `vectorOrgContact`

– `String searchContac(string search)` – ищет необходимый контакт по его id.

– `String numVecId(int a)` – возвращает id контакта, на который указал пользователь.

– `bool idCheck(string id)` – возвращает `true`, если контакт существует.

– `PersonalContact PerContactToTable(int i)` – возвращает объект из вектора `vectorPerContact`.

– `OrganisationContact OrgContactToTable(int i)` – возвращает объект из вектора `vectorOrgContact`.

– `PersonalContact getPerObj(string id)` – возвращает объект с заданным id.

– `OrganisationContact getOrgObj(string id)` – возвращает объект с заданным id

– `void updPerObj(string id, PersonalContact upd )` – принимает объект и id, обновляет объект с таким же id.

– `void updOrgObj(string id, OrganisationContact upd )` – принимает объект и id, обновляет объект с таким же id.

– `void writeFilePerContact()` – записывает данные вектора `vectorPerContact` в файл.

– `void writeFileOrgContact()` – Записывает данные вектора `vectorOrgContact` в файл.

– `void readFileOrgContact()` – Считывает данные из файла и записывает в вектор `vectorOrgContact`.

– `void readFilePerContact()` – Считывает данные из файла и записывает в вектор `vectorPerContact`.

– `bool searchContactNumberOrg(string search)` – Возвращает `True` если находит контакт с номером, как `search`.

– `bool searchContactNumberPer(string search)` – Возвращает `True` если находит контакт с номером, как `search`.

– `string checkNumber(string number)` – проверяет введенный пользователем номер на наличие недопустимых символов и их общее допустимое количество.

– `bool searchOrgContactBool(string search, unsigned long long i)`- возвращает `True`, если находит объект типа `OrganisationContact`.

– `unsigned long long searchOrgContactPosition(string search, unsigned long long i)`- возвращает номер позиции под которым находится объект типа `OrganisationContact`.

– `bool searchPerContactBool(string search, unsigned long long i)`- возвращает `True`, если находит объект типа `PersonalContact`.

– `unsigned long long searchPerContactPosition(string search, unsigned long long k)` - возвращает номер позиции под которым находится объект типа `PersonalContact`.

3.1.5 MainWindow: public QMainWindow

Класс для реализации графического интерфейса и взаимодействия с ним.

Поля:

– `Ui::MainWindow *ui` – указатель на графическую оболочку.

– `QStandardItemModel *model` – указатель на модель.

– `QStandardItemModel *csvModel` – указатель на модель.

– `ContactRepository *repository` – указатель на класс `ContactRepository`.

Методы:

– `void countContacts()` – выводит количество контактов из базы данных на пользовательский интерфейс.

– `void SaveAsClear(QString filename)` – сохранение данных из таблицы.

– `void contactClear()` – очистка введенных данных в виджете.

– `OrganisationContact BuildOrgContact(OraganisationContact *oContact)` – выводит данные объекта в таблицу.

– `PersonalContact BuildPerContact (PersonalContact *pContact)` - выводит данные объекта в таблицу.

– `void setColummDataPer(string firstname)` – выводит в таблицу данные о контакте жителя.

– `void setColummDataOrg(string firstname)` – выводит в таблицу данные о контакте организации.

– `void ClearTable()` – очищает таблицу.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Метод getPerObj (string id) класса ContactRepository.

Шаг 1. Начало.

Шаг.2 Проходим по всему вектору `vectorPerContact`.

Шаг 3. Рассматриваем каждый id в векторе `vectorPerContact`.

Шаг 4. Если id равен id в векторе `vectorPerContact`, то возвращаем этот объект вектора `vectorPerContact`.

Шаг 5. Конец

4.2 Метод getOrgObj (string id) класса ContactRepository.

Шаг 1. Начало.

Шаг.2 Проходим по всему вектору `vectorOrgContact`.

Шаг 3. Рассматриваем каждый id в векторе `vectorOrgContact`.

Шаг 4. Если id равен id в векторе `vectorOrgContact`,
то возвращаем этот объект вектора `vectorOrgContact`.

Шаг 5. Конец.

4.3 Метод contactClear() класса MainWindow.

Шаг 1. Начало.

Шаг 2. Очищаем форму `nameOrglineEdit_9`.

Шаг 3. Очищаем форму `numberOrglineEdit_10`.

Шаг 4. Очищаем форму `countryOrglineEdit_11`.

Шаг 5. Очищаем форму `cityOrglineEdit_12`.

Шаг 6. Очищаем форму `adresOrglineEdit_3`.

Шаг 7. Очищаем форму `commentOrglineEdit_4`.

Шаг 8. Очищаем форму `nameCitizlineEdit_3`.

Шаг 9. Очищаем форму `lastnameCitizlineEdit_4`.

Шаг 10. Очищаем форму `numberCitizrlineEdit_5`.

Шаг 11. Очищаем форму `contryCitizlineEdit_6`.

Шаг 12. Очищаем форму `cityCitizlineEdit_7`.

Шаг 13. Очищаем форму `adresCitizlineEdit_3`.

Шаг 14. Очищаем форму `commentCitizlineEdit_4`.

Шаг 15. Конец.

4.4 Метод idCheck (string id) класса ContactRepository.

Шаг 1. Начало.
Шаг.2 Проходим по всему вектору `vectorOrgContact`.
Шаг 3. Сравниваем id с id в каждом объекте вектора `vectorOrgContact`.
Шаг 4. Если id равен id в векторе `vectorOrgContact`, то возвращаем `true`.
Шаг.5 Если нет, то проходим по всему вектору `vectorPerContact`.
Шаг 6. Сравниваем id с id в каждом объекте вектора `vectorPerContact`.
Шаг 7. Если id равен id в векторе `vectorPerContact`, то возвращаем true.
Шаг 8. Возвращаем `false`.
Шаг 9. Конец.

4.5 Метод setId() класса Contact.

Шаг 1. Начало.
Шаг 2. Идем по пунктам.
Шаг 3. Объявляем переменную `Cheak` типа `ContactRepository`.
Шаг 4. Объявляем переменную `generateId` типа `string`.
Шаг 5. Объявляем переменную `a` типа `int` и присваиваем ей случайное число [1000000000, 9999999999].
Шаг 6. Переводим `a` в тип `string` и присваиваем переменной `generateId`.
Шаг 7. Присваиваем символ ‘C’ для 0 элемента строки `generateId`.
Шаг 8. Объявляем переменную `cheaker` типа `bool`.
Шаг 9. Присваиваем значение к `cheaker`, которое вернет функция `idCheck()` исходя из значения переменной `generateId`.
Шаг 10. Возвращаем значение строки `generateId`, если `cheaker` равен false, иначе возвращаемся к пункту 2.
Шаг 11. Конец.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Для тестирования и отработки возможных ошибок разработанного приложения был использован Qt Creator.

5.1 Тестирование допустимости ввода определенных переменных

При вводе пользователем некорректных символов телефонного номера и превышением допустимого количества цифр (более 20-ти), пользователю будет выведено сообщение предупреждения (рисунок 5.1.1).

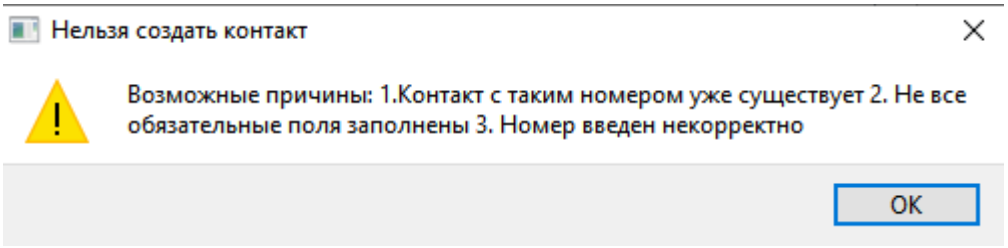

Рисунок 5.1.1 – Сообщение при вводе некорректного телефонного номера

5.2 Тестирование работы приложения при создании контакта с одинаковым номером телефона

Если пользователь решил создать контакт, номер которого существует, будет выведена предупреждение (рисунок 5.2.1).

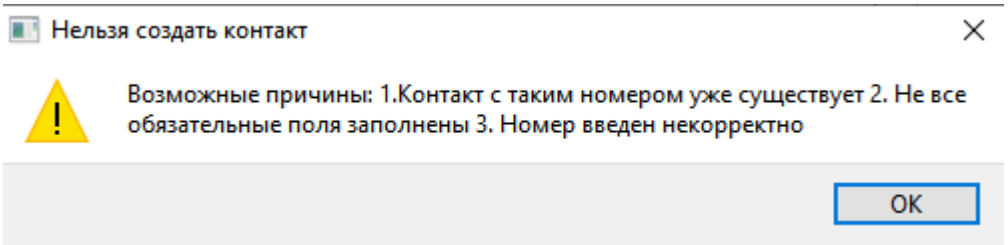

Рисунок 5.2.1 – Предупреждение при создании контакта с одинаковым номером телефона

5.3 Тестирование работы приложения при создании или обновлении контакта без заполнения полей, отмеченных как обязательные

Если пользователь не заполнил поля, отмеченные как обязательные, при создании или обновлении контакта, ему выведется соответствующее

предупреждение (рисунок 5.3.1).

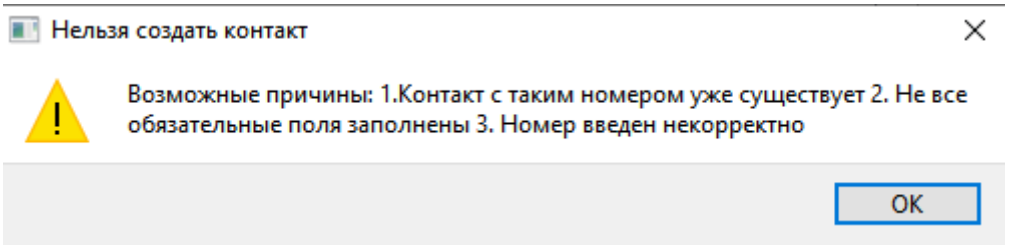

Рисунок 5.3.1 – Предупреждение, при отсутствии заполнения обязательных полей в виджете

5.4 Тестирование работы приложения попытке редактирования контакта, когда ни один не был создан

Если пользователь решил обновить контакт, когда ни один из контактов не было создан, то выведется сообщение об ошибке (рисунок 5.4.1).

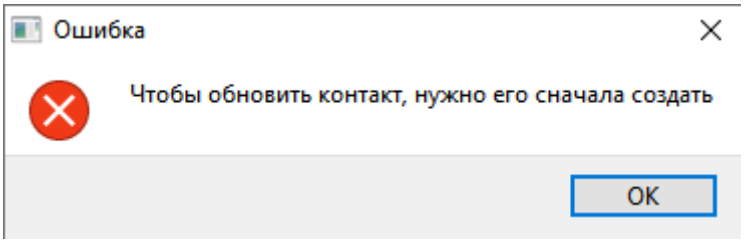

Рисунок 5.4.1 – Загрузка несуществующей базы данных с контактами.

5.5 Тестирование работы приложения попытке поиска по данным несуществующего контакта

Если пользователь ввел в строку поиска данные контакта, которого не существует, то выведется сообщение о результате поиска.

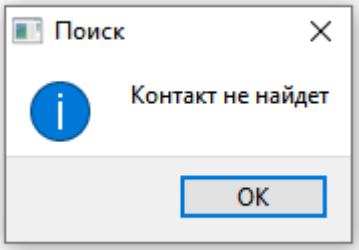

Рисунок 5.5.1 – Информационное окно при отсутствии совпадений поиска.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо открыть файл «ContactBook.exe». После этого откроется основное окно программы (рисунок 6.1).

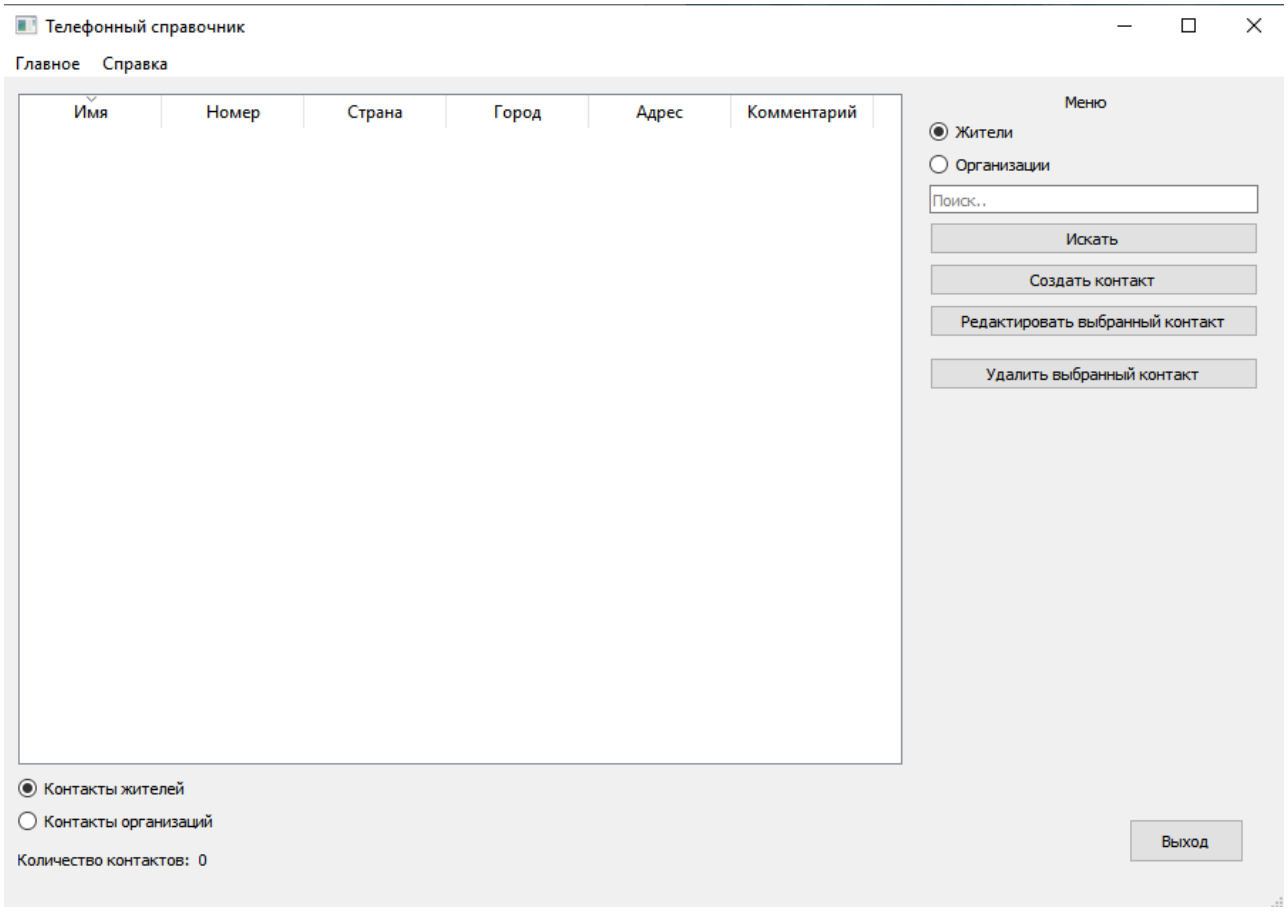

Рисунок 6.1 – Основное окно приложения

При нажатии на кнопку «Создать контакт» появляется виджет (рисунок 6.2). В нем пользователь может выбрать чей контакт он собирается создать. Затем пользователь вводит необходимую информацию для создания контакта. После создания контакта виджет исчезает.

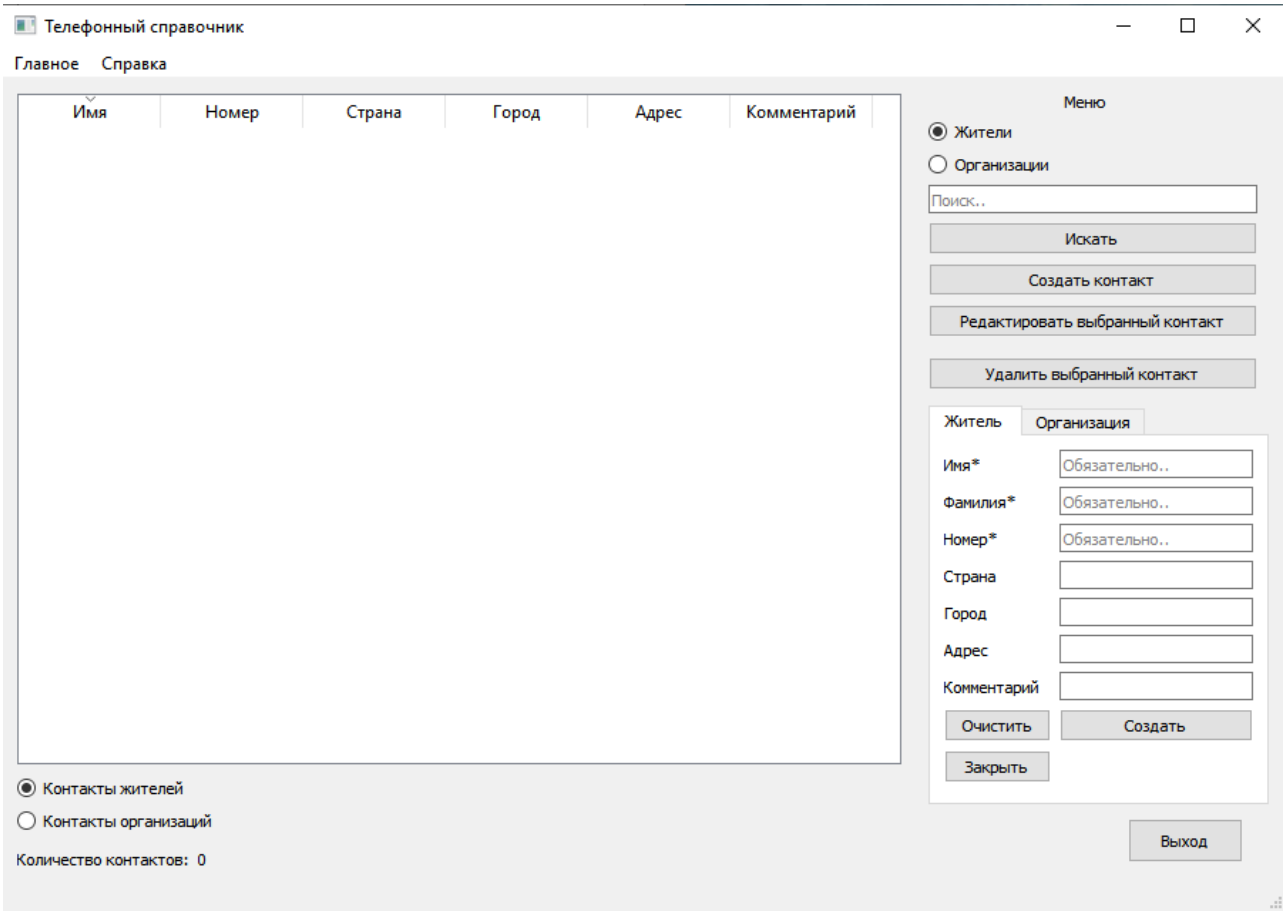

Рисунок 6.2 – Основное окно приложения при нажатии кнопки “Создать контакт”

Для удаления необходимого контакта, достаточно выбрать любой столбец нужной строки, либо на номер строки и нажать “Удалить выбранный контакт” (рисунок 6.3).

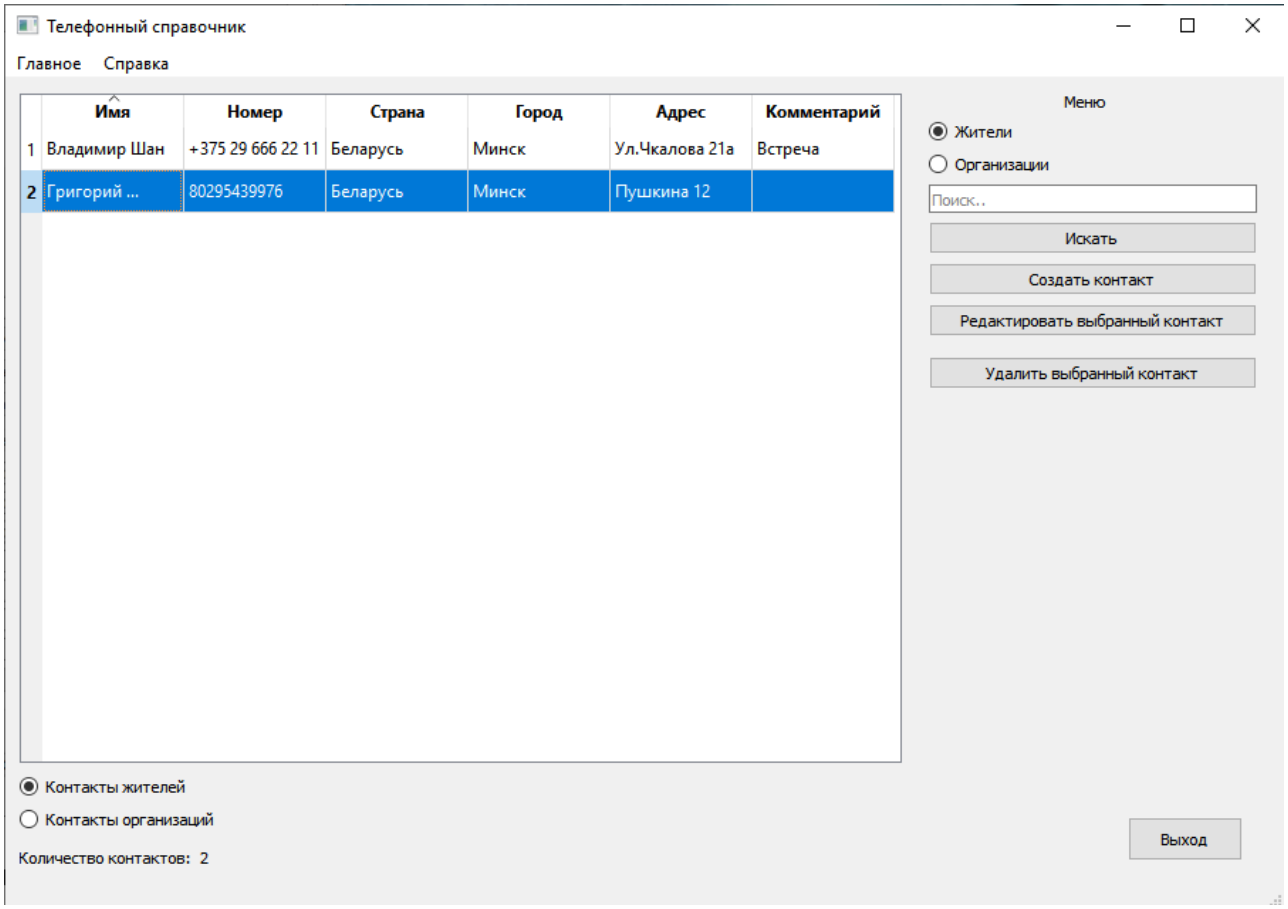

Рисунок 6.3 – Выбор нужного контакта для удаления

Для перехода между категориями контактов, а именно между “жителем” и “организацией” можно воспользоваться специальными кнопками (рисунок 6.4).

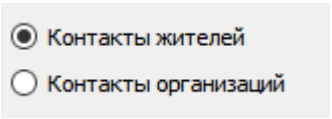

Рисунок 6.4 – Кнопки перехода между разделами контактов

Для поиска данных, пользователь может воспользоваться строкой поиска (рисунок 6.5), перед этим выбрав категорию контакта, в которой будет происходить сам поиск. Если поиск успешный – в таблице будут выведены данные контакта, если нет – пользователь увидит соответствующее уведомление от программы.

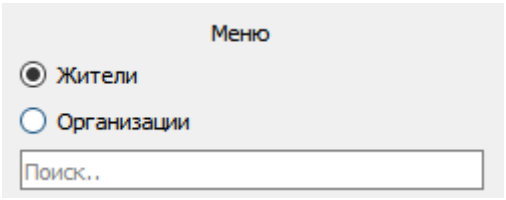

Рисунок 6.5 – Строка поиска по выбранному контакту

Чтобы редактировать контакт, пользователь должен выбрать его в таблице, затем нажать на кнопку “Редактировать выбранный контакт”, после чего он может изменить появившиеся в отдельном виджете данные. (Рисунок 6.6)

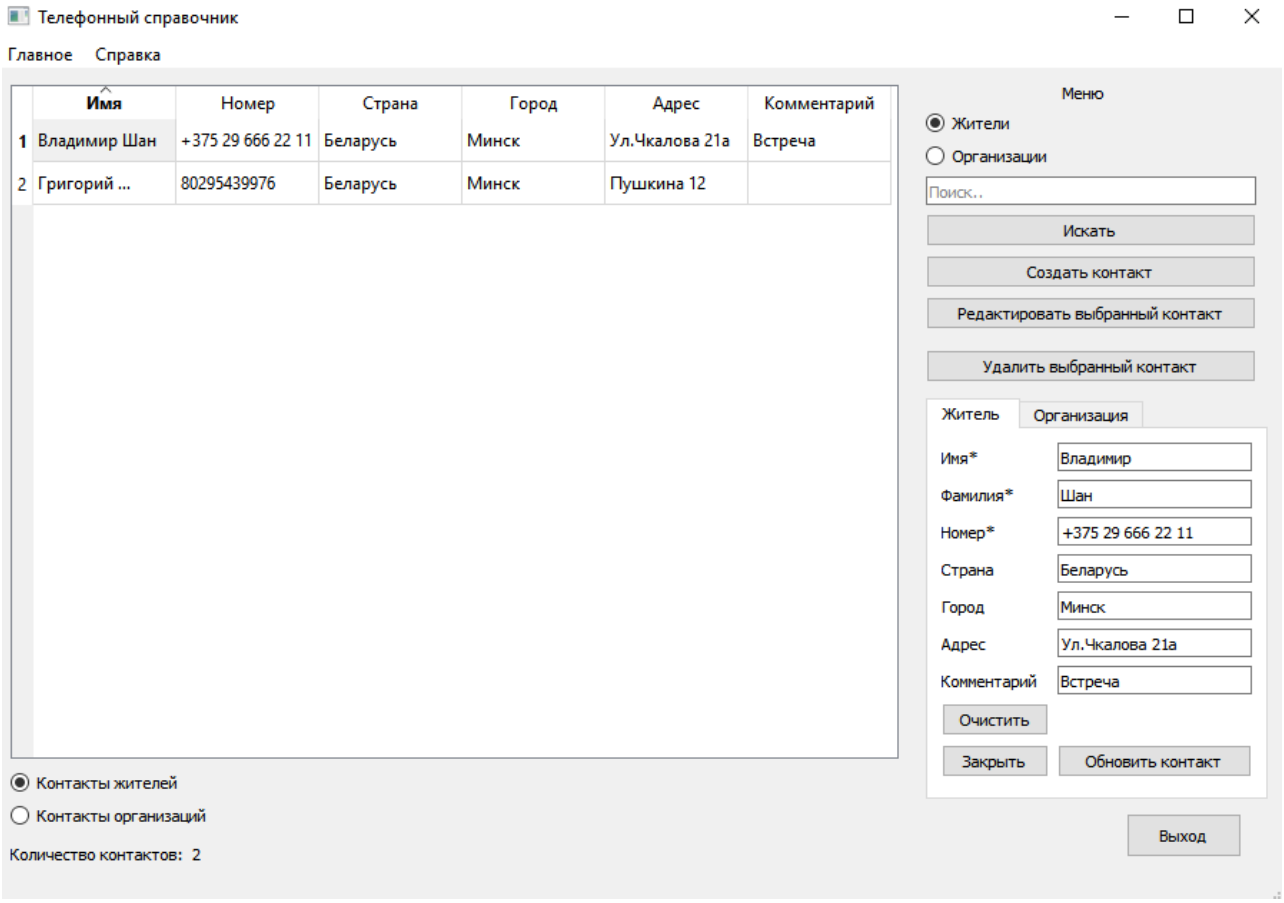

Рисунок 6.6 – Редактирование контакта

Чтобы реализовать изменение контакта, которые пользователь ввел, он должен нажать на кнопку “Обновить контакт”, после нажатия виджет исчезнет и контакт обновиться (рисунок 6.7)

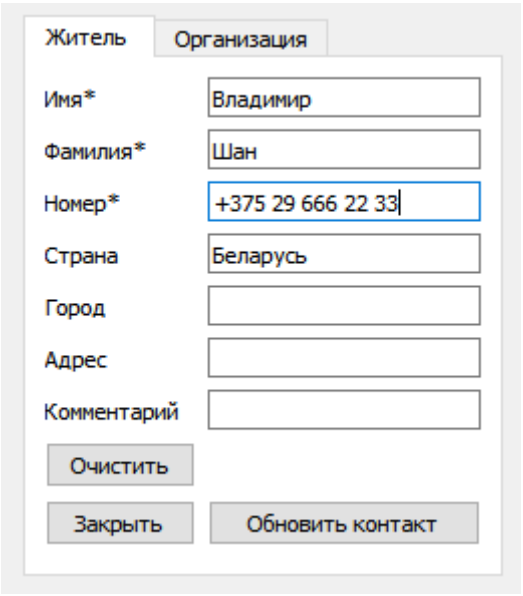

Рисунок 6.7 – Кнопка обновление контакта

Виджет для создания и редактирования контакта автоматически закрывается, но если пользователь передумал выполнять намеченные действия, а именно создание или редактирование контакта, то он может самостоятельно закрыть виджет, нажав кнопку “Закрыть” (рисунок 6.8)

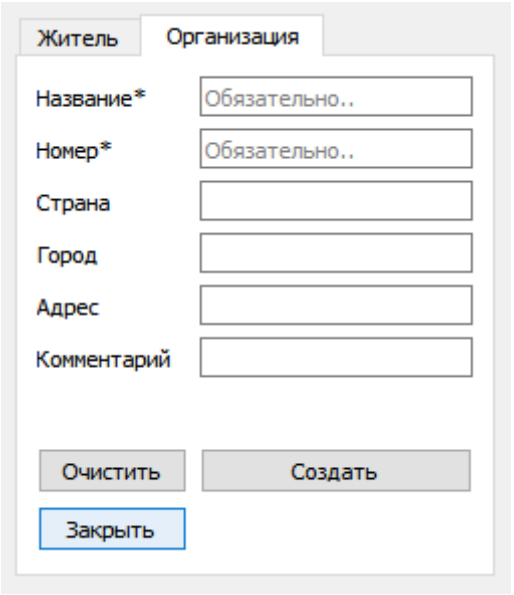

Рисунок 6.8 – Кнопка закрытия виджета

Минимальным количеством данных для создания контакта были отмечены обязательные поля, без их заполнения пользователь не может создать контакт (рисунок 6.9)

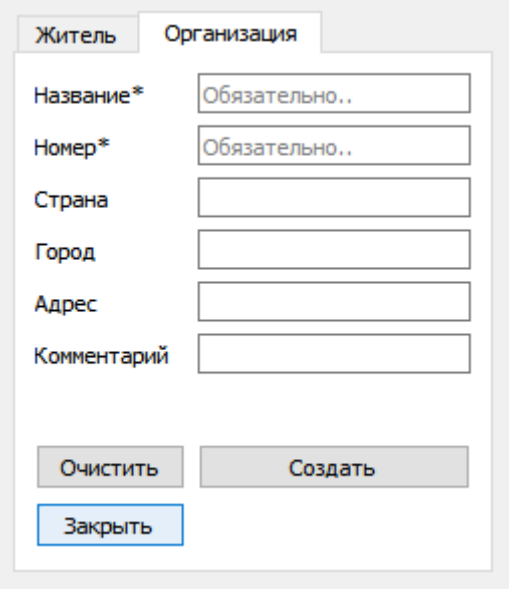

Рисунок 6.9 – Кнопки закрытия виджета

Для сортировки данных пользователю предложена удобная возможность: выбрать категорию сверху таблицы, по нужному критерию (рисунок 6.10)

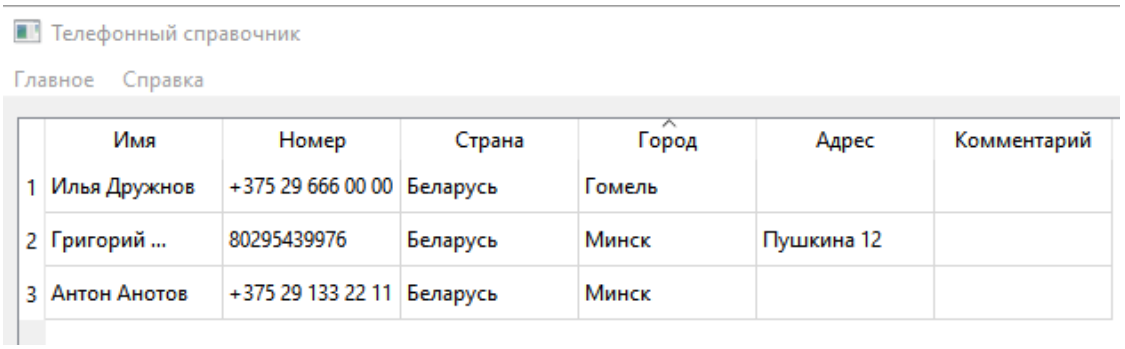

Рисунок 6.10 – Кнопки перехода между разделами контактов

При выходите из приложения пользователю нужно нажать на кнопку “Выход”, затем подтвердить выход в диалоговом окне (рисунок 6.11).

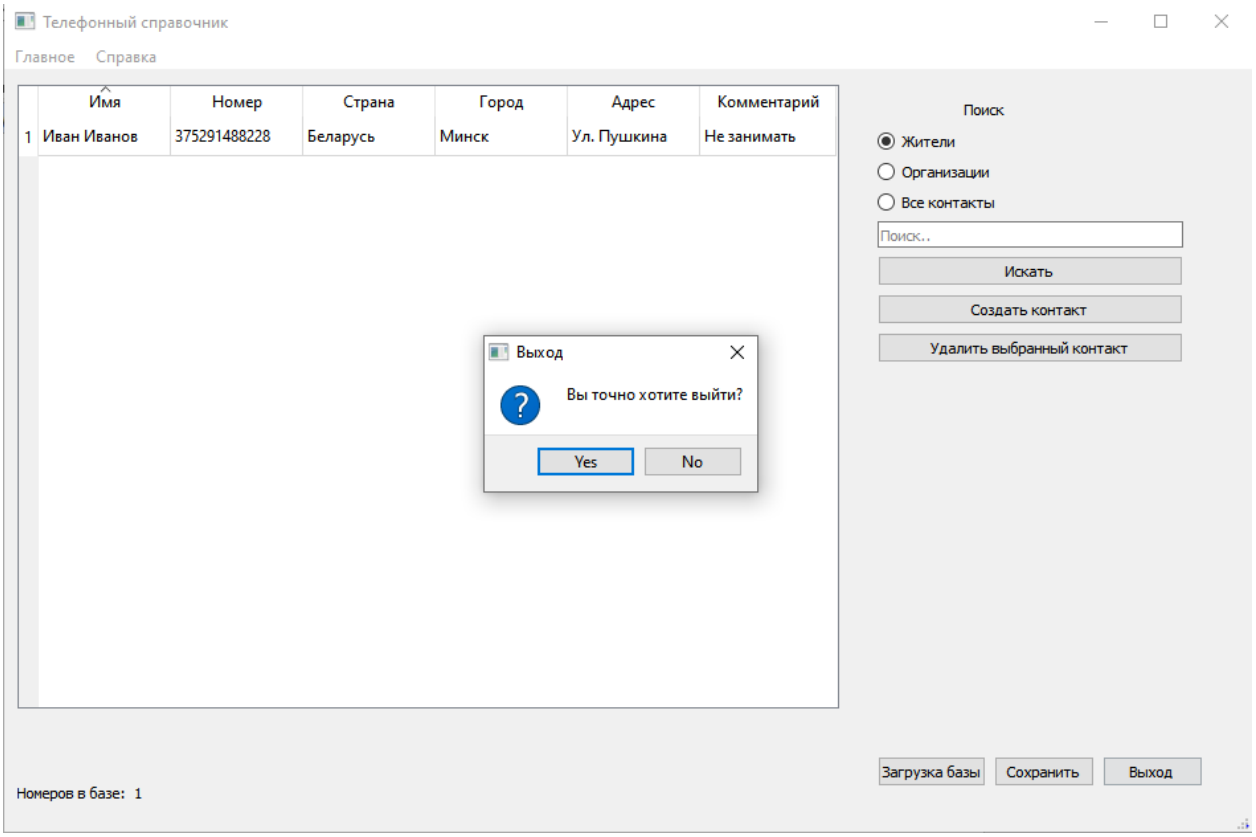

Рисунок 6.11 – Выход и диалоговое окно

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом была разработана работоспособная программа справочник с необходимым набором функций. Данный курсовой проект был разработан в соответствии с поставленными задачами, весь функционал был реализован в полном объеме.

В ходе разработки были изучены возможности языка программирования C++, а конкретно объектно-ориентированное программирование, а также получены навыки для проектирования и реализации графического пользовательского интерфейса в Qt.

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному средству, системное и функциональное проектирование, конструирование программного средства, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем планируется усовершенствование программы, а именно, усовершенствование графического интерфейса и расширения функционала, добавление разных виджетов с актуальной информацией по необходимым вопросам. Одна из главных целей на будущее – перевод в облачную систему хранения данных

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Герберт, Ш. Самоучитель C++/Ш. Герберт. Санкт-Петербург, 2003г. – 678 с.

[2] Страуструп, Б. Программирование. Принципы и практика использования C++/ Б. Страуструп, Вильямс, 2018 г. – 991 с.

[3] Стивен П. Язык программирования C++. Лекции и упражнения, 6-е изд. : Пер. с англ. – М. : ООО ”И.Д. Вильямс”, 2012. – 1248 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    list<Contact> ContactList;
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

contact.h

```
#ifndef CONTACT_H
#define CONTACT_H
#include <string>
#include <cstring>

using namespace std;

class Contact
{
public:
    Contact();
    Contact(string id, string number, string country, string city, string
address, string comment);

    void setPhoneNumber(string phoneNumber);
    string getPhoneNumber();

    void setCountry(string country);
    string getCountry();

    void setCity(string city);
    string getCity();

    void setAddress(string address);
    string getAddress();

    void setComment(string comment);
    string getComment();

    string setId();
    void setFileId(string id);
    string getId();

    ~Contact();
    friend class ContactRepository;

private:
    string _id;
```

```
    string _phoneNumber;
    string _country;
    string _city;
    string _address;
    string _comment;

};

#endif // CONTACT_H
```

contact.cpp

```
#include <iostream>
#include <clocale>
#include <cstring>
#include <string>
#include <contact.h>
#include <contactrepository.h>

using namespace std;

Contact::Contact()
{
    _id = setId();
    _phoneNumber = "";
    _country = "";
    _city = "";
    _address = "";
    _comment = "";
}

Contact::Contact(string id, string comment, string number, string country,
string city, string address):
    _id(id), _phoneNumber(number), _country(country), _city(city),
_address(address), _comment(comment) {};

void Contact::setPhoneNumber(string phoneNumber) {
    _phoneNumber = phoneNumber;
}
string Contact::getPhoneNumber() {
    return _phoneNumber;
}

void Contact::setCountry(string country) {
    this->_country = country;
}
string Contact::getCountry() {
    return _country;
}

void Contact::setCity(string city) {
    this->_city = city;
}
string Contact::getCity() {
    return _city;
}

void Contact::setAddress(string address) {
    this->_address = address;
}
string Contact::getAddress() {
```

```
    return _address;
}

void Contact::setComment(string comment) {
    this->_comment = comment;
}
string Contact::getComment() {
    return _comment;
}

string Contact::setId() {
    while(1){
   ContactRepository Check ;
    string generateId;
    int a = rand() % 9999999999 + 1000000000;
    generateId = std::to_string(a);
    generateId[0] = 'C';
  bool cheaker =  Check.idCheck(generateId);
   if(cheaker == false)
       return generateId;
    }
}
void Contact::setFileId(string id){
    this->_id = id;
}

string Contact::getId() {
    return _id;
}


Contact::~Contact() {};
```

contactrepository.h

```
#ifndef CONTACTREPOSITORY_H
#define CONTACTREPOSITORY_H

#include <contact.h>
#include <iostream>
#include <clocale>
#include <cstring>
#include <vector>
#include <personalcontact.h>
#include <organisationcontact.h>

using namespace std;

class ContactRepository
{
    private:
         vector <OrganisationContact> vectorOrgContact;

         vector <PersonalContact> vectorPerContact;

         vector <Contact> vector;
    public:
        void CreateContact(Contact a);

        void CreateOrgContact(OrganisationContact a);

        void CreatePerContact(PersonalContact a);
```

```
        void Delete(string id, int categoty);

        int getVecSizeContact();

        int getVecSizePerContact();

        int getVecSizeOrgContact();

        OrganisationContact searchOrgContact(string search, unsigned long long
i);

        bool searchOrgContactBool(string search, unsigned long long i);

        unsigned long long searchOrgContactPosition(string search, unsigned
long long i);

        PersonalContact searchPerContact(string search, unsigned long long i);

        bool searchPerContactBool(string search, unsigned long long i);

        unsigned long long searchPerContactPosition(string search, unsigned
long long k);

        string searchContact(string search);

        string searchContactToUpdate(string search);

        string numVecId(unsigned long long a);

        bool idCheck(string id);

        PersonalContact PerContactToTable(int i);

        OrganisationContact OrgContactToTable(int i);

        PersonalContact getPerObj(string id);

        OrganisationContact getOrgObj(string id);

        void updPerObj(string id, PersonalContact upd );

        void updOrgObj(string id, OrganisationContact upd );

        void writeFilePerContact();

        void writeFileOrgContact();

        void readFileOrgContact();

        void readFilePerContact();

        bool searchContactNumberOrg(string search);

        bool searchContactNumberPer(string search);

        string checkNumber(string number);

  friend class Contact;
  friend class OrganisationContact;
  friend class PersonalContact;
};
```

```
#endif // CONTACTREPOSITORY_H
```

contactrepository.cpp

```
#include <contactrepository.h>
#include <contact.h>
#include <iostream>
#include <clocale>
#include <string>
#include <cstring>
#include <vector>
#include <personalcontact.h>
#include <organisationcontact.h>
#include <iomanip>
#include <fstream>

void ContactRepository::CreateContact(Contact a)
{
    this->vector.push_back(a);
}
void ContactRepository::CreateOrgContact(OrganisationContact a){
    this->vectorOrgContact.push_back(a);
}

void ContactRepository::CreatePerContact(PersonalContact a){
    this->vectorPerContact.push_back(a);
}

int ContactRepository::getVecSizePerContact(){
    return vectorPerContact.size();
}

int ContactRepository::getVecSizeOrgContact(){
    return vectorOrgContact.size();
}

int ContactRepository::getVecSizeContact()
{
    return ( getVecSizePerContact() + getVecSizeOrgContact() );
}

string ContactRepository::numVecId(unsigned long long a)
{
    return vector[a].getId();
}

string ContactRepository::searchContact(string search)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getPhoneNumber() ))
        {
            return vectorOrgContact[i].getId();
        }
    }

    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if ((search == vectorPerContact[i].getPhoneNumber() ))
        {
            return vectorPerContact[i].getId();
```

```
        }
    }
    return "";
}

string ContactRepository::searchContactToUpdate(string search)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getPhoneNumber() ))
        {
            return search;
        }
    }

    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if ((search == vectorPerContact[i].getPhoneNumber() ))
        {
            return search;
        }
    }
    return "";
}

bool ContactRepository::searchContactNumberOrg(string search)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getPhoneNumber() ))
        {
            return true;
        }
    }
     return false;
}

 bool ContactRepository::searchContactNumberPer(string search)
{
    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if ((search == vectorPerContact[i].getPhoneNumber() ))
        {
            return true;
        }
    }
   return false;
}


PersonalContact ContactRepository::PerContactToTable(int i)
{
    return vectorPerContact[i];
}

OrganisationContact ContactRepository::OrgContactToTable(int i)
{
    return vectorOrgContact[i];
}


PersonalContact ContactRepository::searchPerContact(string search, unsigned
long long k)
```

```
{
    for (; k < vectorPerContact.size(); k++)
    {
        if ((search == vectorPerContact[k].getAddress()) || (search ==
vectorPerContact[k].getCity()) || (search == vectorPerContact[k].getCountry())
||
                (search == vectorPerContact[k].getPhoneNumber() || (search ==
vectorPerContact[k].getFirstname()) ||
                 (search == vectorPerContact[k].getLastname()) || (search ==
vectorPerContact[k].getFullName()) ))
        {
            return vectorPerContact[k];
        }
    }
}

unsigned long long ContactRepository::searchPerContactPosition(string search,
unsigned long long k)
{
    for (; k < vectorPerContact.size(); k++)
    {
        if ((search == vectorPerContact[k].getAddress()) || (search ==
vectorPerContact[k].getCity()) || (search == vectorPerContact[k].getCountry())
||
                (search == vectorPerContact[k].getPhoneNumber() || (search ==
vectorPerContact[k].getFirstname()) ||
                 (search == vectorPerContact[k].getLastname()) || (search ==
vectorPerContact[k].getFullName()) ))
        {
            return k;
        }
    }
}

bool ContactRepository::searchPerContactBool(string search, unsigned long long
i)
{
    for (; i < vectorPerContact.size(); i++)
    {
        if ((search == vectorPerContact[i].getAddress()) || (search ==
vectorPerContact[i].getCity()) || (search == vectorPerContact[i].getCountry())
||
                (search == vectorPerContact[i].getPhoneNumber() || (search ==
vectorPerContact[i].getFirstname()) ||
                 (search == vectorPerContact[i].getLastname()) || (search ==
vectorPerContact[i].getFullName()) ))
        {
            return true;
        }
    }
    return false;
}

OrganisationContact ContactRepository::searchOrgContact(string search, unsigned
long long i)
{
    for (; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getAddress()) || (search ==
vectorOrgContact[i].getCity()) || (search == vectorOrgContact[i].getCountry())
||
                (search == vectorOrgContact[i].getPhoneNumber() || (search ==
vectorOrgContact[i].getNameOrg()) ))
```

```
        {
            return vectorOrgContact[i];
        }
    }
}

unsigned long long ContactRepository::searchOrgContactPosition(string search,
unsigned long long i)
{
    for (; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getAddress()) || (search ==
vectorOrgContact[i].getCity()) || (search == vectorOrgContact[i].getCountry())
||
                (search == vectorOrgContact[i].getPhoneNumber() || (search ==
vectorOrgContact[i].getNameOrg()) ))
        {
            return i;
        }
    }
}

bool ContactRepository::searchOrgContactBool(string search, unsigned long long
i)
{
    for (; i < vectorOrgContact.size(); i++)
    {
        if ((search == vectorOrgContact[i].getAddress()) || (search ==
vectorOrgContact[i].getCity()) || (search == vectorOrgContact[i].getCountry())
||
                (search == vectorOrgContact[i].getPhoneNumber() || (search ==
vectorOrgContact[i].getNameOrg()) ))
        {
            return true;
        }
    }
    return false;
}

void ContactRepository::Delete(string id, int categoty)
{
    unsigned long long delNum;
    if(categoty == 0){
        for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
        {
            if (id == vectorOrgContact[i].getId())
            {
                delNum = i;
                vectorOrgContact.erase(vectorOrgContact.begin()+delNum);
                return;
            }
        }
    }
    if(categoty == 1){
        for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
        {
            if (id == vectorPerContact[i].getId())
            {
                delNum = i;
                vectorPerContact.erase(vectorPerContact.begin()+delNum);
                return;
            }
        }
```

```
    }
}

bool ContactRepository::idCheck(string id)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if (id == vectorOrgContact[i].getId())
        {
            return true;
        }
    }
    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if (id == vectorPerContact[i].getId())
        {
            return true;
        }
    }

    return false;

}

PersonalContact ContactRepository::getPerObj(string id)
{
    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if ((id == vectorPerContact[i].getId() ))
        {
            return vectorPerContact[i];
        }
    }
}

OrganisationContact ContactRepository::getOrgObj(string id)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if ((id == vectorOrgContact[i].getId() ))
        {
            return vectorOrgContact[i];
        }
    }
}

void ContactRepository::updPerObj(string id, PersonalContact upd )
{
    for (unsigned long long i = 0; i < vectorPerContact.size(); i++)
    {
        if ((id == vectorPerContact[i].getId() ))
        {
            vectorPerContact[i].setFirstname(upd.getFirstname());
            vectorPerContact[i].setLastname(upd.getLastname());
            vectorPerContact[i].setCountry(upd.getCountry());
            vectorPerContact[i].setPhoneNumber(upd.getPhoneNumber());
            vectorPerContact[i].setCity(upd.getCity());
            vectorPerContact[i].setAddress(upd.getAddress());
            vectorPerContact[i].setComment(upd.getComment());
        }
    }
}
```

```
void ContactRepository::updOrgObj(string id, OrganisationContact upd)
{
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++)
    {
        if ((id == vectorOrgContact[i].getId() ))
        {
            vectorOrgContact[i].setNameOrg(upd.getNameOrg());
            vectorOrgContact[i].setPhoneNumber(upd.getPhoneNumber());
            vectorOrgContact[i].setCountry(upd.getCountry());
            vectorOrgContact[i].setCity(upd.getCity());
            vectorOrgContact[i].setAddress(upd.getAddress());
            vectorOrgContact[i].setComment(upd.getComment());
        }
    }
}

void ContactRepository::writeFilePerContact() {
    ofstream fout("PersonalContactDB.txt");
    for (unsigned long long i = 0; i < vectorPerContact.size(); i++) {
        fout << vectorPerContact[i].getId() << "/"
             << vectorPerContact[i].getFirstname() << "/"
             << vectorPerContact[i].getLastname() << "/"
             << vectorPerContact[i].getPhoneNumber() << "/"
             << vectorPerContact[i].getCountry() << "/"
             << vectorPerContact[i].getCity() << "/"
             << vectorPerContact[i].getAddress() << "/";
        if(i == (vectorPerContact.size()-1)) fout << this-
>vectorPerContact[i].getComment() << "/.";
        else fout << this->vectorPerContact[i].getComment() << "/";
    }
    fout.close();
}

void ContactRepository::writeFileOrgContact() {
    ofstream fout("OrganisationContactDB.txt");
    for (unsigned long long i = 0; i < vectorOrgContact.size(); i++) {
        fout << vectorOrgContact[i].getId() << "/"
             << vectorOrgContact[i].getNameOrg() << "/"
             << vectorOrgContact[i].getPhoneNumber() << "/"
             << vectorOrgContact[i].getCountry() << "/"
             << vectorOrgContact[i].getCity() << "/"
             << vectorOrgContact[i].getAddress() << "/";
        if(i == (vectorOrgContact.size()-1)) fout << this-
>vectorOrgContact[i].getComment() << "/.";
        else fout << this->vectorOrgContact[i].getComment() << "/";
    }
    fout.close();
}

void ContactRepository::readFileOrgContact() {
    vectorOrgContact.clear();

    OrganisationContact *a = new OrganisationContact();
    ifstream ins("OrganisationContactDB.txt");
    char stop;
        while (1) {
            ins.get(stop);
            if (stop == '.')
                break;
            else
                ins.seekg(-1, ins.cur);

            getline(ins, a->_id, '/');
```

```
            a->setFileId(a->_id);

            getline(ins, a->_nameOrg, '/');
            a->setNameOrg(a->_nameOrg);

            getline(ins, a->_phoneNumber, '/');
            a->setPhoneNumber(a->_phoneNumber);

            getline(ins, a->_country, '/');
            a->setCountry(a->_country);

            getline(ins, a->_city, '/');
            a->setCity(a->_city);

            getline(ins, a->_address, '/');
            a->setAddress(a->_address);

            getline(ins, a->_comment, '/');
            a->setComment(a->_comment);

            vectorOrgContact.push_back(*a);
        }
    ins.close();

}

void ContactRepository::readFilePerContact() {
    vectorPerContact.clear();

    PersonalContact *b = new PersonalContact();
    ifstream ins("PersonalContactDB.txt");
    char stop;
        while (1) {
            ins.get(stop);
            if (stop == '.')
                break;
            else
                ins.seekg(-1, ins.cur);

            getline(ins, b->_id, '/');
            b->setFileId(b->_id);

            getline(ins, b->_firstname, '/');
            b->setFirstname(b->_firstname);

            getline(ins, b->_lastname, '/');
            b->setLastname(b->_lastname);

            getline(ins, b->_phoneNumber, '/');
            b->setPhoneNumber(b->_phoneNumber);

            getline(ins, b->_country, '/');
            b->setCountry(b->_country);

            getline(ins, b->_city, '/');
            b->setCity(b->_city);

            getline(ins, b->_address, '/');
            b->setAddress(b->_address);

            getline(ins, b->_comment, '/');
            b->setComment(b->_comment);
```

```
            vectorPerContact.push_back(*b);
        }
    ins.close();

}

string ContactRepository::checkNumber(string number){
    string num = number;
    for(unsigned long long i=0; i < num.length();i++){
        if(!((num[i] >= '0' && num[i] <= '9') || num[0] == '+' || num[i] == '
'))
            return "";
    }
    if(num.length() > 20)
        return "";

    return num;
}
```

organisationcontact.h

```
#ifndef ORGANISATIONCONTACT_H
#define ORGANISATIONCONTACT_H
#include <string>
#include <cstring>
#include <contact.h>

using namespace std;

class OrganisationContact: virtual public Contact {

public:
    OrganisationContact();

    OrganisationContact(string nameOrg);

    void setNameOrg(string nameOrg);

    string getNameOrg();

    ~OrganisationContact();

    friend class ContactRepository;
private:
    string _nameOrg;
};

#endif // ORGANISATIONCONTACT_H
```

organisationcontact.cpp

```
#include "organisationcontact.h"
#include "contact.h"
#include <iostream>
#include <clocale>
#include <cstring>
#include <string>
#include <contact.h>
```

```
using namespace std;


OrganisationContact::OrganisationContact() {
    _nameOrg = "";
}

OrganisationContact::OrganisationContact(string nameOrg) :_nameOrg(nameOrg) {};

void OrganisationContact::setNameOrg(string nameOrg) {
    this->_nameOrg = nameOrg;
}
string OrganisationContact::getNameOrg() {
    return _nameOrg;
}

OrganisationContact::~OrganisationContact() {};
```

personalcontact.h

```
#ifndef PERSONALCONTACT_H
#define PERSONALCONTACT_H
#include <string>
#include <cstring>
#include <contact.h>

using namespace std;

class PersonalContact : virtual public Contact {

public:
    PersonalContact();
    PersonalContact(string firstname, string lastname);

    void setFirstname(string firstname);
    string getFirstname();

    void setLastname(string lastname);
    string getLastname();

    string getFullName();

    ~PersonalContact();

    friend class ContactRepository;
private:
    string _firstname;
    string _lastname;
};

#endif // PERSONALCONTACT_H
```

personalcontact.cpp

```
#include "personalcontact.h"
#include "contact.h"
#include <iostream>
```

```
#include <clocale>
#include <cstring>
#include <contact.h>

using namespace std;

PersonalContact::PersonalContact() {
    _firstname = "";
    _lastname = "";
}

PersonalContact::PersonalContact(string firstname, string lastname) :
_firstname(firstname), _lastname(lastname) {};

void PersonalContact::setFirstname(string firstname) {
    this->_firstname = firstname;
}
string PersonalContact::getFirstname() {
    return _firstname;
}

void PersonalContact::setLastname(string lastname) {
    this->_lastname = lastname;
}
string PersonalContact::getLastname() {
    return _lastname;
}

string PersonalContact::getFullName() {
    return this->_firstname + ' ' + this->_lastname;
}

PersonalContact::~PersonalContact() {};
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <contact.h>
#include <QStandardItemModel>
#include <contactrepository.h>
#include <organisationcontact.h>
#include <personalcontact.h>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

using namespace std;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
```

```
    void on_pushButton_clicked();

    void on_action_4_triggered();

    void on_exitMainpushButton_clicked();

    void on_CreateContactCitizpushButton_2_clicked();

    void on_ClearCitizpushButton_3_clicked();

    void on_createOrgpushButton_3_clicked();

    void on_clearOrgpushButton_2_clicked();

    void countContacts();

    void contactClear();

    void on_delpushButton_clicked();

    OrganisationContact BuildOrgContact(OrganisationContact *oContact);

    PersonalContact BuildPerContact(PersonalContact *pContact);

    void setColummDataPer(int lastRowIndex, PersonalContact *pContact);

    void setColummDataOrg(int lastRowIndex, OrganisationContact *oContact);

    void on_searchpushButton_clicked();

    void ClearTable(int c);

    void on_setTablePerradioButton_pressed();

    void on_setTableOrgradioButton_2_pressed();

    void on_updatePerpushButton_clicked();

    void on_updateOrgpushButton_clicked();

    void on_UpdateContactpushButton_clicked();

    bool is_emptyFile(std::ifstream& pFile);

    void on_CitizradioButton_pressed();

    void on_OrgradioButton_clicked();

    void on_closeTabWpushButton_2_clicked();

    void on_closeTabWPerpushButton_2_clicked();

private:
    Ui::MainWindow *ui;
    QStandardItemModel *model;
    QStandardItemModel *csvModel;
    ContactRepository *repository;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QMessageBox>
#include <QDebug>
#include <QtWidgets/QApplication>
#include "QStandardItemModel"
#include "QStandardItem"
#include <QTableView>
#include <QAbstractItemModel>
#include <QAbstractItemView>
#include <iomanip>
#include <fstream>

#include <contact.h>
#include <organisationcontact.h>
#include <personalcontact.h>
#include "contactrepository.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    repository = new ContactRepository;

    ui->tabWidget->setVisible(false);

    model = new QStandardItemModel( 0, 6 , this);

    ui->tableView->setModel(model);

    countContacts();

    model -> setHeaderData(0, Qt::Horizontal, "Имя");
    model -> setHeaderData(1, Qt::Horizontal, "Номер");
    model -> setHeaderData(2, Qt::Horizontal, "Страна");
    model -> setHeaderData(3, Qt::Horizontal, "Город");
    model -> setHeaderData(4, Qt::Horizontal, "Адрес");
    model -> setHeaderData(5, Qt::Horizontal, "Комментарий");
    ui->CitizradioButton->setChecked(1);
    ui->setTablePerradioButton->setChecked(1);

    ui->updateOrgpushButton->setVisible(false);
    ui->updatePerpushButton->setVisible(false);

    ifstream fileOrgContactDB("OrganisationContactDB.txt");
    ifstream filePerContactDB("PersonalContactDB.txt");

    if (!is_emptyFile(fileOrgContactDB))
    {
        repository->readFileOrgContact();
    }

    if (!is_emptyFile(filePerContactDB))
    {
        repository->readFilePerContact();
    }

    repository->writeFileOrgContact();
    repository->writeFilePerContact();

    on_setTablePerradioButton_pressed();
```

```
    countContacts();
}

MainWindow::~MainWindow()
{
    delete ui;
}

bool MainWindow::is_emptyFile(std::ifstream& pFile)
{
    return pFile.peek() == std::ifstream::traits_type::eof();
}

void MainWindow::on_pushButton_clicked()
{
    ui->tabWidget->setVisible(true);
    ui->createOrgpushButton_3->setVisible(true);
    ui->CreateContactCitizpushButton_2->setVisible(true);
    ui->updateOrgpushButton->setVisible(false);
    ui->updatePerpushButton->setVisible(false);
    contactClear();
}

void MainWindow::on_action_4_triggered()
{
    repository->writeFileOrgContact();
    repository->writeFilePerContact();
    QApplication::quit();
}

void MainWindow::on_exitMainpushButton_clicked()
{
    QMessageBox::StandardButton exit = QMessageBox::question(this,"Выход","Вы
точно хотите выйти?", QMessageBox::Yes | QMessageBox::No);
    if(exit == QMessageBox::Yes)
        QApplication::exit();
}

void MainWindow::on_CreateContactCitizpushButton_2_clicked()
{
    string mainLineName = ui->nameCitizlineEdit_3->text().toStdString();
    string mainLineLastName = ui->lastnameCitizlineEdit_4-
>text().toStdString();
    string mainLineNumber = ui->numberCitizrlineEdit_5->text().toStdString();
    mainLineNumber = repository->checkNumber(mainLineNumber);

     if(repository->searchContactNumberOrg(mainLineNumber)== false &&
repository->searchContactNumberPer(mainLineNumber)== false){
        if (mainLineName != "" && mainLineLastName != "" && mainLineNumber !=
"") {
            PersonalContact *pContact = new PersonalContact();
            *pContact = BuildPerContact(pContact);
            repository->CreatePerContact(*pContact);
            repository->writeFilePerContact();

            ui->tabWidget->setVisible(false);

            contactClear();

            countContacts();

            on_setTablePerradioButton_pressed();
```

```
        }else
            QMessageBox::warning(this,"Нельзя создать контакт","Возможные
причины: 1.Контакт с таким номером уже существует 2. Не все обязательные поля
заполнены 3. Номер введен некорректно  ");
    }else
        QMessageBox::warning(this,"Нельзя создать контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
}

void MainWindow::on_ClearCitizpushButton_3_clicked()
{
    contactClear();
}

void MainWindow::on_createOrgpushButton_3_clicked()
{
    string mainLineName = ui->nameOrglineEdit_9->text().toStdString();
    string mainLineNumber = ui->numberOrglineEdit_10->text().toStdString();
    mainLineNumber = repository->checkNumber(mainLineNumber);
    repository->searchContactNumberOrg(mainLineNumber);
    if(repository->searchContactNumberOrg(mainLineNumber)== false &&
repository->searchContactNumberPer(mainLineNumber)== false){
    if (mainLineName != "" && mainLineNumber != ""){
        OrganisationContact *oContact = new OrganisationContact();
        if (mainLineName != "" && mainLineNumber != ""){
            *oContact = BuildOrgContact(oContact);
            repository->CreateOrgContact(*oContact);
            repository->writeFileOrgContact();

            ui->tabWidget->setVisible(false);

            contactClear();

            countContacts();

            on_setTableOrgradioButton_2_pressed();
        }
    }else{
        QMessageBox::warning(this,"Нельзя создать контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
    }
    }else
         QMessageBox::warning(this,"Нельзя создать контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
}
void MainWindow::contactClear(){

    ui->nameOrglineEdit_9->clear();
    ui->numberOrglineEdit_10->clear();
    ui->countryOrglineEdit_11->clear();
    ui->cityOrglineEdit_12->clear();
    ui->adresOrglineEdit_3->clear();
    ui->commentOrglineEdit_4->clear();
    ui->nameCitizlineEdit_3->clear();

    ui->lastnameCitizlineEdit_4->clear();
    ui->numberCitizrlineEdit_5->clear();
    ui->contryCitizlineEdit_6->clear();
    ui->cityCitizlineEdit_7->clear();
    ui->adresCitizlineEdit_3->clear();
```

```
    ui->commentCitizlineEdit_4->clear();
}

void MainWindow::on_clearOrgpushButton_2_clicked()
{
    contactClear();
}

void MainWindow::countContacts() // ок
{
    int a = repository->getVecSizeContact();
    QString s = QString::number(a);
    ui->countPersons->setText(s);
}

void MainWindow::on_delpushButton_clicked()
{

    if(ui->setTableOrgradioButton_2->isChecked())
    {
        if((ui->tableView->currentIndex().isValid())){
            int a = ui->tableView->currentIndex().row();
            string s = ui->tableView->model()->data(ui->tableView->model()-
>index(a,1)).toString().toStdString();
            int z = 0;
            ClearTable(1);
            repository->Delete(repository->searchContact(s),z);
            on_setTableOrgradioButton_2_pressed();
            repository->writeFileOrgContact();
            a=0;
        }
        else {
            QMessageBox::information(this,"Удаление контакта","Чтобы удалить
контакт, нужно выбрать его в таблице");
        }
    }

    if(ui->setTablePerradioButton->isChecked())
    {
        if((ui->tableView->currentIndex().isValid())){
            int b = ui->tableView->currentIndex().row();
            string s = ui->tableView->model()->data(ui->tableView->model()-
>index(b,1)).toString().toStdString();
            int z = 1;
            ClearTable(1);
            repository->Delete(repository->searchContact(s),z);
            on_setTablePerradioButton_pressed();
            repository->writeFilePerContact();
            b=0;
        }else {
            QMessageBox::information(this,"Удаление контакта","Чтобы удалить
контакт, нужно выбрать его в таблице");
        }
    }
    countContacts();
}

OrganisationContact MainWindow::BuildOrgContact(OrganisationContact *oContact)
{

    oContact->setNameOrg(ui->nameOrglineEdit_9->text().toStdString());
    oContact->setPhoneNumber(ui->numberOrglineEdit_10->text().toStdString());
    oContact->setCountry(ui->countryOrglineEdit_11->text().toStdString());
```

```
    oContact->setCity(ui->cityOrglineEdit_12->text().toStdString());
    oContact->setAddress(ui->adresOrglineEdit_3->text().toStdString());
    oContact->setComment(ui->commentOrglineEdit_4->text().toStdString());
    return *oContact;
}

PersonalContact MainWindow::BuildPerContact(PersonalContact *pContact)
{
    pContact->setFirstname(ui->nameCitizlineEdit_3->text().toStdString());
    pContact->setLastname(ui->lastnameCitizlineEdit_4->text().toStdString());
    pContact->setPhoneNumber(ui->numberCitizrlineEdit_5->text().toStdString());
    pContact->setCountry(ui->contryCitizlineEdit_6->text().toStdString());
    pContact->setCity(ui->cityCitizlineEdit_7->text().toStdString());
    pContact->setAddress(ui->adresCitizlineEdit_3->text().toStdString());
    pContact->setComment(ui->commentCitizlineEdit_4->text().toStdString());
    return *pContact;
}

void MainWindow::setColummDataPer(int lastRowIndex, PersonalContact *pContact){
    int col = 0;
    QModelIndex rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact-
>getFullName()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact-
>getPhoneNumber()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact-
>getCountry()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact->getCity()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact-
>getAddress()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(pContact-
>getComment()));
}

void MainWindow::setColummDataOrg(int lastRowIndex, OrganisationContact
*oContact)
{
    int col = 0;
    QModelIndex rowColIndex = model->index(lastRowIndex,col);
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact-
>getNameOrg()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact-
>getPhoneNumber()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact-
>getCountry()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact->getCity()));
```

```
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact-
>getAddress()));
    col++;
    rowColIndex = model->index(lastRowIndex,col);
    model->setData(rowColIndex, QString::fromStdString(oContact-
>getComment()));
}

void MainWindow::on_searchpushButton_clicked()
{
    string search = ui->lineEdit_search->text().toStdString();
    ui->lineEdit_search->clear();
    if(search != ""){
        if(ui->CitizradioButton->isChecked()){
            PersonalContact *pContact = new PersonalContact();

            if(repository->searchPerContactBool(search,0) == true){
                ClearTable(0);
                int j = 0;
                for (int i = 0; i < repository->getVecSizePerContact(); i++)
                {
                    if(repository->searchPerContactBool(search,i) == true){

                        i=repository->searchPerContactPosition(search, i);

                        *pContact = repository->searchPerContact(search, i);
                        model->insertRows(j,1);
                        setColummDataPer(j , pContact);
                        j++;
                    }
                }
            }else
                QMessageBox::information(this,"Поиск","Контакт не найдет");
            return;
        }
        if(ui->OrgradioButton->isChecked()){
            OrganisationContact *pContact = new OrganisationContact();

            if(repository->searchOrgContactBool(search,0) == true){
                ClearTable(0);
                int j = 0;
                for (int i = 0; i < repository->getVecSizeOrgContact(); i++)
                {
                    if(repository->searchOrgContactBool(search,i) == true){

                        i=repository->searchOrgContactPosition(search, i);

                        *pContact = repository->searchOrgContact(search, i);
                        model->insertRows(j,1);
                        setColummDataOrg(j , pContact);
                        j++;
                    }
                }
            }else
                QMessageBox::information(this,"Поиск","Контакт не найдет");
            return;
        }
    }else{
        if(ui->setTablePerradioButton->isChecked()){
            on_setTablePerradioButton_pressed();
        }
```

```
        if(ui->setTableOrgradioButton_2->isChecked()){
            on_setTableOrgradioButton_2_pressed();
        }
    }
}
void MainWindow::ClearTable(int c)
{
    if(repository->getVecSizeOrgContact() >= repository-
>getVecSizePerContact()) {
        for(int i = 0; i < repository->getVecSizeOrgContact()+ c; i++)
            model->removeRows(ui->tableView->rowAt(i) , 1);
    }else if(repository->getVecSizeOrgContact() < repository-
>getVecSizePerContact()) {
        for(int i = 0; i < repository->getVecSizePerContact() + c; i++)
            model->removeRows(ui->tableView->rowAt(i) , 1);
    }
}

void MainWindow::on_setTablePerradioButton_pressed()
{
    ui->CitizradioButton->setChecked(1);
    ui->OrgradioButton->setChecked(0);
    ClearTable(0);
    for(int k = 0; k  < repository->getVecSizePerContact() ; k++ ){
        PersonalContact *pContact = new PersonalContact();
        *pContact = repository->PerContactToTable(k);
        model->insertRows(k,1);
        setColummDataPer(k , pContact);
    }
}

void MainWindow::on_setTableOrgradioButton_2_pressed()
{
    ui->CitizradioButton->setChecked(0);
    ui->OrgradioButton->setChecked(1);
    ClearTable(0);
    for(int k = 0; k  < repository->getVecSizeOrgContact() ; k++ ){
        OrganisationContact *oContact = new OrganisationContact();
        *oContact = repository->OrgContactToTable(k);
        model->insertRows(k,1);
        setColummDataOrg(k, oContact);
    }
}

void MainWindow::on_updatePerpushButton_clicked()
{
    string mainLineName = ui->nameCitizlineEdit_3->text().toStdString();
    string mainLineLastName = ui->lastnameCitizlineEdit_4-
>text().toStdString();
    string mainLineNumber = ui->numberCitizrlineEdit_5->text().toStdString();

    mainLineNumber = repository->checkNumber(mainLineNumber);

    int a = ui->tableView->currentIndex().row();
    string s = ui->tableView->model()->data(ui->tableView->model()-
>index(a,1)).toString().toStdString();

    if((repository->searchContactNumberOrg(mainLineNumber)== false &&
repository->searchContactNumberPer(mainLineNumber)== false) || (s ==
mainLineNumber)){
         mainLineNumber = repository->checkNumber(mainLineNumber);
    if (mainLineName != "" && mainLineNumber != "" && mainLineNumber != "") {
        if((ui->tableView->currentIndex().isValid())){
```

```
            if(ui->setTablePerradioButton->isChecked()){
                int a = ui->tableView->currentIndex().row();
                string s = ui->tableView->model()->data(ui->tableView->model()-
>index(a,1)).toString().toStdString();
                PersonalContact *pContact = new PersonalContact();
                *pContact = repository->getPerObj(repository-
>searchContact(s));
                string id = pContact->getId();
                *pContact = BuildPerContact(pContact);
                repository->updPerObj(id, *pContact);
                ClearTable(0);
                on_setTablePerradioButton_pressed();
                repository->writeFilePerContact();
                contactClear();
            }else
                QMessageBox::warning(this,"Нельзя обновить
контакт","Неправильно выбрана категория контакта");
        }
        on_clearOrgpushButton_2_clicked();
        ui->tabWidget->setVisible(false);
    }else
        QMessageBox::warning(this,"Нельзя обновить контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
}else
   QMessageBox::warning(this,"Нельзя обновить контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
}

void MainWindow::on_updateOrgpushButton_clicked()
{
    string mainLineName = ui->nameOrglineEdit_9->text().toStdString();
    string mainLineNumber = ui->numberOrglineEdit_10->text().toStdString();
    int a = ui->tableView->currentIndex().row();
    string s = ui->tableView->model()->data(ui->tableView->model()-
>index(a,1)).toString().toStdString();

  if((repository->searchContactNumberOrg(mainLineNumber)== false && repository-
>searchContactNumberPer(mainLineNumber)== false) || (s == mainLineNumber)){
     mainLineNumber= repository->checkNumber(mainLineNumber);
    if (mainLineName != ""  && mainLineNumber != "") {
        if((ui->tableView->currentIndex().isValid())){
            if(ui->setTableOrgradioButton_2->isChecked()){
                int a = ui->tableView->currentIndex().row();
                string s = ui->tableView->model()->data(ui->tableView->model()-
>index(a,1)).toString().toStdString();

                OrganisationContact *oContact = new OrganisationContact();
                *oContact = repository->getOrgObj(repository-
>searchContact(s));
                string id = oContact->getId();
                *oContact = BuildOrgContact(oContact);
                repository->updOrgObj(id, *oContact);
                ClearTable(0);
                on_setTableOrgradioButton_2_pressed();
                repository->writeFileOrgContact();
                contactClear();

            }else {
                QMessageBox::information(this,"Обновление
контакта","Неправильно выбрана категория контакта");
            }
```

```
        }
        on_clearOrgpushButton_2_clicked();
        ui->tabWidget->setVisible(false);
    }
    else{
        QMessageBox::warning(this,"Нельзя обновить контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
  }
}else
      QMessageBox::warning(this,"Нельзя обновить контакт","Возможные причины:
1.Контакт с таким номером уже существует 2. Не все обязательные поля заполнены
3. Номер введен некорректно  ");
}
void MainWindow::on_UpdateContactpushButton_clicked()
{
    on_clearOrgpushButton_2_clicked();

    if(repository->getVecSizeContact()!= 0) {

        ui->tabWidget->setVisible(true);

        ui->updateOrgpushButton->setVisible(false);
        ui->updatePerpushButton->setVisible(false);

        if(ui->setTableOrgradioButton_2->isChecked()){
            ui->updateOrgpushButton->setVisible(true);
            ui->updatePerpushButton->setVisible(false);
        }
        if(ui->setTablePerradioButton->isChecked()){
            ui->updateOrgpushButton->setVisible(false);
            ui->updatePerpushButton->setVisible(true);
        }

        ui->createOrgpushButton_3->setVisible(false);
        ui->CreateContactCitizpushButton_2->setVisible(false);
    if(ui->setTableOrgradioButton_2->isChecked()){
        if((ui->tableView->currentIndex().isValid())){
            if(repository->getVecSizeOrgContact()!= 0){
                if(ui->setTableOrgradioButton_2->isChecked()){
                    int a = ui->tableView->currentIndex().row();
                    string s = ui->tableView->model()->data(ui->tableView-
>model()->index(a,1)).toString().toStdString();
                    OrganisationContact *oContact = new OrganisationContact();
                    *oContact = repository->getOrgObj(repository-
>searchContact(s));

                    ui->nameOrglineEdit_9-
>setText(QString::fromStdString(oContact->getNameOrg()));
                    ui->numberOrglineEdit_10-
>setText(QString::fromStdString(oContact->getPhoneNumber()));
                    ui->countryOrglineEdit_11-
>setText(QString::fromStdString(oContact->getCountry()));
                    ui->cityOrglineEdit_12-
>setText(QString::fromStdString(oContact->getCity()));
                    ui->adresOrglineEdit_3-
>setText(QString::fromStdString(oContact->getAddress()));
                    ui->commentOrglineEdit_4-
>setText(QString::fromStdString(oContact->getComment()));

                }
            }
        }
```

```
    }
    if(ui->setTablePerradioButton->isChecked()){
        if((ui->tableView->currentIndex().isValid())){
            if(repository->getVecSizePerContact()!= 0){
                if(ui->setTablePerradioButton->isChecked()){
                    int a = ui->tableView->currentIndex().row();
                    string s = ui->tableView->model()->data(ui->tableView-
>model()->index(a,1)).toString().toStdString();
                    PersonalContact *pContact = new PersonalContact();
                    *pContact = repository->getPerObj(repository-
>searchContact(s));

                    ui->nameCitizlineEdit_3-
>setText(QString::fromStdString(pContact->getFirstname()));
                    ui->lastnameCitizlineEdit_4-
>setText(QString::fromStdString(pContact->getLastname()));
                    ui->numberCitizrlineEdit_5-
>setText(QString::fromStdString(pContact->getPhoneNumber()));
                    ui->contryCitizlineEdit_6-
>setText(QString::fromStdString(pContact->getCountry()));
                    ui->cityCitizlineEdit_7-
>setText(QString::fromStdString(pContact->getCity()));
                    ui->adresCitizlineEdit_3-
>setText(QString::fromStdString(pContact->getAddress()));
                    ui->commentCitizlineEdit_4-
>setText(QString::fromStdString(pContact->getComment()));
                }
            }
        }
    }
    }else
        QMessageBox::critical(this,"Ошибка", "Чтобы обновить контакт, нужно его
сначала создать ");
}

void MainWindow::on_CitizradioButton_pressed()
{
    ui->setTableOrgradioButton_2->setChecked(0);
    ui->setTablePerradioButton->setChecked(1);
    on_setTablePerradioButton_pressed();
}

void MainWindow::on_OrgradioButton_clicked()
{
    ui->setTableOrgradioButton_2->setChecked(1);
    ui->setTablePerradioButton->setChecked(0);
    on_setTableOrgradioButton_2_pressed();
}

void MainWindow::on_closeTabWpushButton_2_clicked()
{
    ui->tabWidget->setVisible(false);
    contactClear();
}

void MainWindow::on_closeTabWPerpushButton_2_clicked()
{
    ui->tabWidget->setVisible(false);
    contactClear();
}
```

ПРИЛОЖЕНИЕ Г
(*обязательное*)

Ведомость документов