МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Отчет по дисциплине: «Объектно-ориентированное программирование»

«Телефонный справочник»

Студент,
группы 5130201/40003 _____ Адиатуллин Т. Р.

Работу
принял _____ Глазунов В. В.

«____»_____ 2026 г.

Санкт-Петербург, 2026

Содержание

1. Введение

Современные информационные системы требуют эффективных инструментов для управления контактными данными. Телефонный справочник является базовым, но важным приложением, демонстрирующим принципы работы с данными, пользовательским интерфейсом и файловым хранилищем. В рамках данной лабораторной работы разработано консольное или десктопное приложение для управления телефонными контактами.

1.1. Используемые технологии

В ходе работы применяются следующие технологии и инструменты:

- **Язык программирования:** C++ (стандарт C++17)
- **Фреймворк разработки:** Qt Framework (версия 6.10)
- **Среда разработки:** Visual Studio Code
- **Система сборки:** CMake

1.2. Описание фреймворка Qt

Qt представляет собой кросс-платформенный фреймворк для разработки приложений с графическим интерфейсом пользователя. Фреймворк Qt обладает следующими ключевыми особенностями:

1.2.1. Механизм сигналов и слотов

Основой архитектуры Qt является механизм сигналов и слотов, который обеспечивает гибкую систему обмена сообщениями между объектами. Сигнал излучается объектом при наступлении определённого события, а слот представляет собой функцию, вызываемую в ответ на этот сигнал.

1.2.2. Виджеты и компоненты интерфейса

Qt предоставляет обширную библиотеку готовых виджетов для построения графического интерфейса:

- Контейнеры: QWidget, QMainWindow, QDialog
- Элементы ввода: QLineEdit, QTextEdit, QComboBox
- Кнопки: QPushButton, QRadioButton, QCheckBox
- Отображение данных: QTableWidget, QListWidget
- Диалоги: QMessageBox, QFileDialog, QInputDialog

1.2.3. Работа с данными

Qt включает мощные инструменты для работы с различными типами данных:

- Работа с файловой системой через QFile, QDir
- Контейнеры данных: QList, QVector, QMap, QString

1.3. Предметная область - телефонный справочник

Телефонный справочник - это приложение для хранения, организации и управления контактной информацией. Типичная запись справочника содержит:

- **Личные данные:** фамилия, имя, отчество (опционально)
- **Контактная информация:** один или несколько номеров телефона
- **Дополнительные данные:** адрес электронной почты, физический адрес

Современный телефонный справочник должен обеспечивать следующие возможности:

1. **Управление записями:**
   - Добавление новых контактов с валидацией введённых данных
   - Просмотр полного списка сохранённых контактов
   - Редактирование существующих записей
   - Удаление контактов
2. **Поиск и фильтрация:**
   - Быстрый поиск по различным критериям (имя, фамилия, телефон, Email, дата рождения)
3. **Сортировка данных:**
   - Упорядочивание по алфавиту
   - Сортировка по дате добавления записи
4. **Персистентность данных:**
   - Сохранение данных между сеансами работы приложения
   - Автоматическое или ручное сохранение изменений
   - Возможность экспорта и импорта данных

2. Постановка задачи

Цель работы - разработать функциональное приложение «Телефонный справочник» с графическим интерфейсом пользователя, реализующее полный набор операций по управлению контактными данными с соблюдением требований к валидации, хранению и отображению информации.

2.1. Формализованное задание

Разработать приложение «Телефонный справочник» со следующим функционалом:

2.1.1. Требования к функциональности

Управление контактами:

1. **Добавление нового контакта**
    - Ввод фамилии, имени, отчества контакта
    - Ввод номера телефона с валидацией формата
    - Ввод дополнительной информации (email, адрес)
    - Проверка уникальности номера телефона
    - Сохранение контакта в список
2. **Просмотр списка контактов**
    - Отображение всех контактов в виде таблицы или списка
    - Вывод информации: порядковый номер, фамилия, имя, телефон
    - Возможность сортировки по различным полям
3. **Поиск контакта**
    - Поиск по фамилии, имени, отчеству, email, дата рождения, номеру телефона
    - Отображение результатов поиска
    - Обработка случая отсутствия совпадений (пустота)
4. **Редактирование контакта**
    - Выбор существующего контакта для редактирования
    - Изменение любого поля записи
    - Повторная валидация изменённых данных

- Сохранение обновлённой информации

5. **Удаление контакта**

   - Выбор контакта для удаления
   - Подтверждение операции удаления
   - Удаление из списка и файла хранения

2.2. Технические требования

2.2.1. Формат хранения данных

Данные должны сохраняться в файл для обеспечения персистентности. Рекомендуемые форматы:

```
{
  "contacts": [
    {
        "address": "Мичурина 152",
        "birthDate": "2006-12-13",
        "email": "timrsamara@gmail.com",
        "firstName": "Тимур",
        "lastName": "Адиатуллин",
        "middleName": "Ринатович",
        "phoneNumbers": [
            "+79953448227"
        ]
    },
    {
        "address": "ул. Ленина 12",
        "birthDate": "2006-01-08",
        "email": "ivan.ivanov@example.com",
        "firstName": "Иван",
        "lastName": "Иванов",
        "middleName": "Иванович",
        "phoneNumbers": [
            "+79991234567",
            "+78005553535"
        ]
    }
  ]
}
```

Listing 1: Пример JSON формата

2.2.2. Регулярные выражения для валидации

Для обеспечения корректности введённых данных необходимо применять следующие регулярные выражения:

Фамилия Имя Отчество

```
^[A-Za-AzЯаяЁё--0-9][A-Za-AzЯаяЁё--0-9\- ]*[A-Za-AzЯаяЁё--0-9]$
```

Требования: начинается с заглавной буквы, содержит только буквы и дефис (для двойных фамилий).

Номер телефона

```
^\+\d+$
```

Поддерживаемые форматы:

- +7 (123) 456-78-90
- +7 123 456 78 90
- 81234567890
- +79991234567

Email

```
// Первая проверка
^[A-Za-z0-9._%+\-]+$
// Проверка домена
^[A-Za-z0-9][A-Za-z0-9\-]*(\.[A-Za-z0-9][A-Za-z0-9\-]*)*\.[A-Za-z]{2,}$
```

Требования: стандартный формат электронной почты.

2.3. Дополнительные требования

2.3.1. Функции для обязательной реализации

Чтение и запись файла:

- `bool loadFromFile(const QString& filename)` - загрузка данных из файла при запуске
- `bool saveToFile(const QString& filename)` - сохранение данных в файл
- Обработка ошибок при работе с файлами (файл не существует, нет прав доступа)

Обработка пользовательского ввода:

- Валидация данных перед сохранением
- Очистка и нормализация введённых данных (удаление лишних пробелов)
- Информирование пользователя о некорректных данных
- Предотвращение ввода дубликатов номеров телефонов

Сортировка данных:

- Сортировка по фамилии (в алфавитном порядке)
- Сортировка по имени
- Возможность обратной сортировки
- Сохранение порядка сортировки между сеансами (опционально)

Поиск и фильтрация:

- Регистронезависимый поиск
- Поиск по части строки (подстроке)
- Множественные результаты при совпадении
- Быстрый доступ к найденным записям

2.4. Ожидаемые результаты

По завершении работы должно быть получено:

- Полностью функциональное приложение с интерфейсом пользователя
- Исходный код, организованный в виде логически разделённых классов
- Файл данных с сохранёнными контактами

3. Реализация

3.1. Архитектура приложения

Приложение «Телефонный справочник» построено на основе объектно-ориентирован подхода с четким разделением ответственности между компонентами системы. Архитектура следует принципам модульности и слабой связанности, что обеспечивает удобство сопровождения и расширения функциональности.

Основные компоненты приложения:

- **Contact** - структура данных для хранения информации о контакте;
- **ContactValidator** - класс для валидации и нормализации пользовательского ввода;
- **ContactStorage** - класс для работы с файловым хранилищем (JSON);
- **MainWindow** - класс главного окна, реализующий графический интерфейс и логику взаимодействия с пользователем.

Структура директорий проекта:

```
lab8
  CMakeLists.txt
  src
    main.cpp
    contact.cpp
    contactstorage.cpp
    contactvalidator.cpp
    mainwindow.cpp
  headers/
    contact.h
    contactstorage.h
    contactvalidator.h
    mainwindow.h
```

Listing 2: Структура проекта

Взаимодействие компонентов представлено на диаграмме (см. рисунок 1).

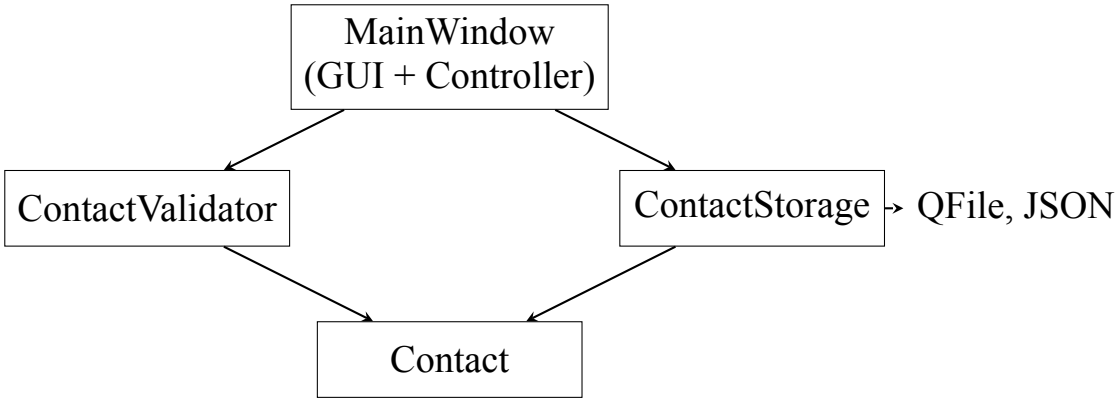

Рис. 1: Архитектура приложения

Класс MainWindow выступает центральным компонентом, координирующим работу остальных модулей. Он обрабатывает события пользовательского интерфейса, вызывает методы валидации перед операциями с данными и использует ContactStorage для сохранения и загрузки информации.

3.2. Описание классов

3.2.1. Структура Contact

Структура Contact представляет модель данных для хранения информации о контакте. Содержит все необходимые поля согласно требованиям задания.

```
#ifndef CONTACT_H
#define CONTACT_H

#include <QString>
#include <QDate>
#include <QStringList>
#include <QJsonObject>

class Contact
{
public:
    Contact();
    Contact(const QString& firstName, const QString& lastName,
            const QString& middleName, const QString& address,
            const QDate& birthDate, const QString& email,
            const QStringList& phoneNumbers);

    QString firstName() const { return m_firstName; }
    QString lastName() const { return m_lastName; }
    QString middleName() const { return m_middleName; }
    QString address() const { return m_address; }
    QDate birthDate() const { return m_birthDate; }
    QString email() const { return m_email; }
    QStringList phoneNumbers() const { return m_phoneNumbers; }

    void setFirstName(const QString& firstName);
    void setLastName(const QString& lastName);
    void setMiddleName(const QString& middleName);
    void setAddress(const QString& address);
    void setBirthDate(const QDate& birthDate);
    void setEmail(const QString& email);
    void setPhoneNumbers(const QStringList& phoneNumbers);

    QJsonObject toJson() const;
    static Contact fromJson(const QJsonObject& json);

    bool operator<(const Contact& other) const;
    bool operator==(const Contact& other) const;

```

```
40 private:
41     QString m_firstName;
42     QString m_lastName;
43     QString m_middleName;
44     QString m_address;
45     QDate m_birthDate;
46     QString m_email;
47     QStringList m_phoneNumbers;
48 };
49
50 #endif // CONTACT_H
```

Listing 3: Заголовочный файл contact.h

Класс реализует методы сериализации в формат JSON и десериализации из него, что обеспечивает удобное сохранение данных в файл. Операторы сравнения используются для сортировки контактов по фамилии, имени и отчеству.

3.2.2. Класс ContactValidator

Класс `ContactValidator` инкапсулирует логику валидации пользовательского ввода с использованием регулярных выражений. Обеспечивает проверку корректности данных перед их сохранением.

```
1  #ifndef CONTACTVALIDATOR_H
2  #define CONTACTVALIDATOR_H
3
4  #include <QString>
5  #include <QDate>
6
7  class ContactValidator
8  {
9  public:
10     static bool validateName(const QString& name,
11                              QString& errorMessage);
12     static bool validatePhone(const QString& phone,
13                               QString& errorMessage);
14     static bool validateBirthDate(const QDate& date,
15                                   QString& errorMessage);
16     static bool validateEmail(const QString& email,
17                               QString& errorMessage);
18
19     static QString normalizeName(const QString& name);
20     static QString normalizePhone(const QString& phone);
21     static QString normalizeEmail(const QString& email);
22
23 private:
24     static QString extractPhoneDigits(const QString& phone);
25 };
26
27 #endif // CONTACTVALIDATOR_H
```

Listing 4: Заголовочный файл contactvalidator.h

Регулярные выражения для валидации:

- **Имя:** `^[A-Za-zA-Яа-яЁё0-9][A-Za-zA-Яа-яЁё0-9\- ]*[A-Za-zA-Яа-я`
  - Разрешены буквы (латиница и кириллица), цифры, дефис, пробел;
  - Не может начинаться или заканчиваться дефисом;
  - Запрещены двойные дефисы и множественные пробелы.
- **Телефон:** `^\+\d+$`
  - Международный формат (начинается с +);
  - Длина от 10 до 15 цифр;
  - Автоматическое преобразование формата 8(XXX)XXX-XX-XX в +7XXXXXXXXXX.
- **Email:** `^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$`
  - Имя пользователя: латинские буквы, цифры, точки, подчеркивания;
  - Обязательное наличие символа @;
  - Домен должен содержать минимум одну точку.
- **Дата рождения:**
  - Должна быть меньше текущей даты;
  - Валидация високосных годов выполняется классом QDate.

Метод validateName():

Выполняет проверку корректности имени контакта.
Основные этапы работы метода:

- нормализация входной строки имени и проверка её на пустое значение;
- удаление начальных и конечных пробелов у исходного ввода;
- проверка того, что имя не начинается и не заканчивается дефисом;
- проверка допустимого набора символов (буквы латинского и кириллического алфавита, цифры, пробел и дефис);
- проверка отсутствия двойных дефисов и последовательностей из нескольких пробелов.

В случае обнаружения ошибки метод возвращает значение `false` и записывает описание причины ошибки в параметр `errorMessage`. При успешном прохождении всех проверок метод возвращает значение `true`.

Методы нормализации:

Нормализация обеспечивает приведение данных к единому формату перед сохранением:

- `normalizeName` - удаляет лишние пробелы, приводит первую букву каждого слова к заглавной, остальные к строчным;
- `normalizePhone` - удаляет пробелы, скобки, дефисы, преобразует российский формат 8(...) в +7(...);
- `normalizeEmail` - приводит к нижнему регистру, удаляет пробелы.

3.2.3. Класс ContactStorage

Класс `ContactStorage` отвечает за сохранение и загрузку контактов из файла в формате JSON. Использует классы `QFile`, `QJsonDocument`, `QJsonArray` и `QJsonObject` из фреймворка Qt.

```
#ifndef CONTACTSTORAGE_H
#define CONTACTSTORAGE_H

#include "contact.h"
#include <QList>
#include <QString>

class ContactStorage
{
public:
    ContactStorage(const QString& filename = "phonebook.json");

    bool load();
    bool save();

    QList<Contact>& contacts();
    const QList<Contact>& contacts() const;

    void addContact(const Contact& contact);
    void removeContact(int index);
    void updateContact(int index, const Contact& contact);

    QString filename() const;
    void setFilename(const QString& filename);

private:
    QList<Contact> m_contacts;
    QString m_filename;
};

```

```
31 #endif // CONTACTSTORAGE_H
```

Listing 5: Заголовочный файл contactstorage.h

Метод load():

Метод `load` выполняет загрузку списка контактов из файла в формате JSON. Основные этапы работы метода:

- проверка существования файла с данными контактов; при отсутствии файла список контактов очищается и метод завершается успешно;
- открытие файла в режиме чтения и загрузка его содержимого в память;
- разбор JSON-данных и обработка возможных ошибок парсинга;
- проверка того, что корневой элемент JSON-документа является массивом;
- очистка текущего списка контактов;
- последовательное преобразование элементов JSON-массива в объекты класса `Contact` и добавление их в коллекцию контактов.

В случае возникновения ошибок при открытии файла или разборе JSON метод возвращает значение `false`. При успешной загрузке данных метод возвращает значение `true`.

Метод save():

Метод `save` выполняет сохранение списка контактов в файл в формате JSON. Основные этапы работы метода:

- открытие файла для записи по указанному имени;
- формирование JSON-массива на основе текущего списка контактов;
- преобразование объектов класса `Contact` в формат JSON и добавление их в массив;
- создание JSON-документа и запись его в файл;
- закрытие файла после завершения записи данных.

В случае невозможности открыть файл для записи метод возвращает значение `false`. При успешном сохранении данных метод возвращает значение `true`.

3.2.4. Класс MainWindow

Класс `MainWindow` наследуется от `QMainWindow` и реализует графический интерфейс приложения, а также всю бизнес-логику взаимодействия с пользователем.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTableWidget>
#include <QPushButton>
#include <QLineEdit>
#include <QDateEdit>
#include <QTextEdit>
#include <QComboBox>
#include <QStack>
#include "contactstorage.h"
#include "contact.h"

struct ContactAction {
    enum Type { Add, Edit, Delete };
    Type type;
    Contact contact;
    int index;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void addContact();
    void editContact();
    void deleteContact();
    void searchContacts();
    void clearSearch();
    void onTableSelectionChanged();
    void loadContacts();
    void saveContacts();
    void undoLastAction();

private:
    void setupUI();
    void setupTable();
    void setupForm();
    void setupSearch();
    void populateTable();
    void clearForm();
```

```
48      bool validateForm(QString& errorMessage);
49      bool checkForDuplicates(const Contact& contact,
50                              int excludeIndex = -1);
51
52      // UI компоненты
53      QTableWidget* m_table;
54      QLineEdit* m_firstNameEdit;
55      QLineEdit* m_lastNameEdit;
56      QLineEdit* m_middleNameEdit;
57      QTextEdit* m_addressEdit;
58      QDateEdit* m_birthDateEdit;
59      QLineEdit* m_emailEdit;
60      QTextEdit* m_phoneNumbersEdit;
61      QPushButton* m_addButton;
62      QPushButton* m_editButton;
63      QPushButton* m_deleteButton;
64      QPushButton* m_undoButton;
65      QLineEdit* m_searchEdit;
66      QComboBox* m_searchFieldCombo;
67      QDateEdit* m_searchDateEdit;
68      QComboBox* m_dateSearchTypeCombo;
69
70      // Данные
71      ContactStorage* m_storage;
72      int m_editingIndex;
73      bool m_isEditing;
74      QStack<ContactAction> m_undoStack;
75  };
76
77  #endif // MAINWINDOW_H
```

Listing 6: Фрагмент заголовочного файла mainwindow.h

Ключевые компоненты графического интерфейса:

- **QTableWidget** - таблица для отображения контактов с возможностью сортировки по столбцам;
- **QLineEdit, QTextEdit, QDateEdit** - поля ввода данных контакта;
- **QComboBox** - выпадающие списки для выбора поля поиска и типа сравнения даты;
- **QPushButton** - кнопки управления (Добавить, Редактировать, Удалить, Отмена, Поиск);
- **QGroupBox** - группировка элементов формы и поиска.

Внешний вид главного окна представлен на рисунке 2.

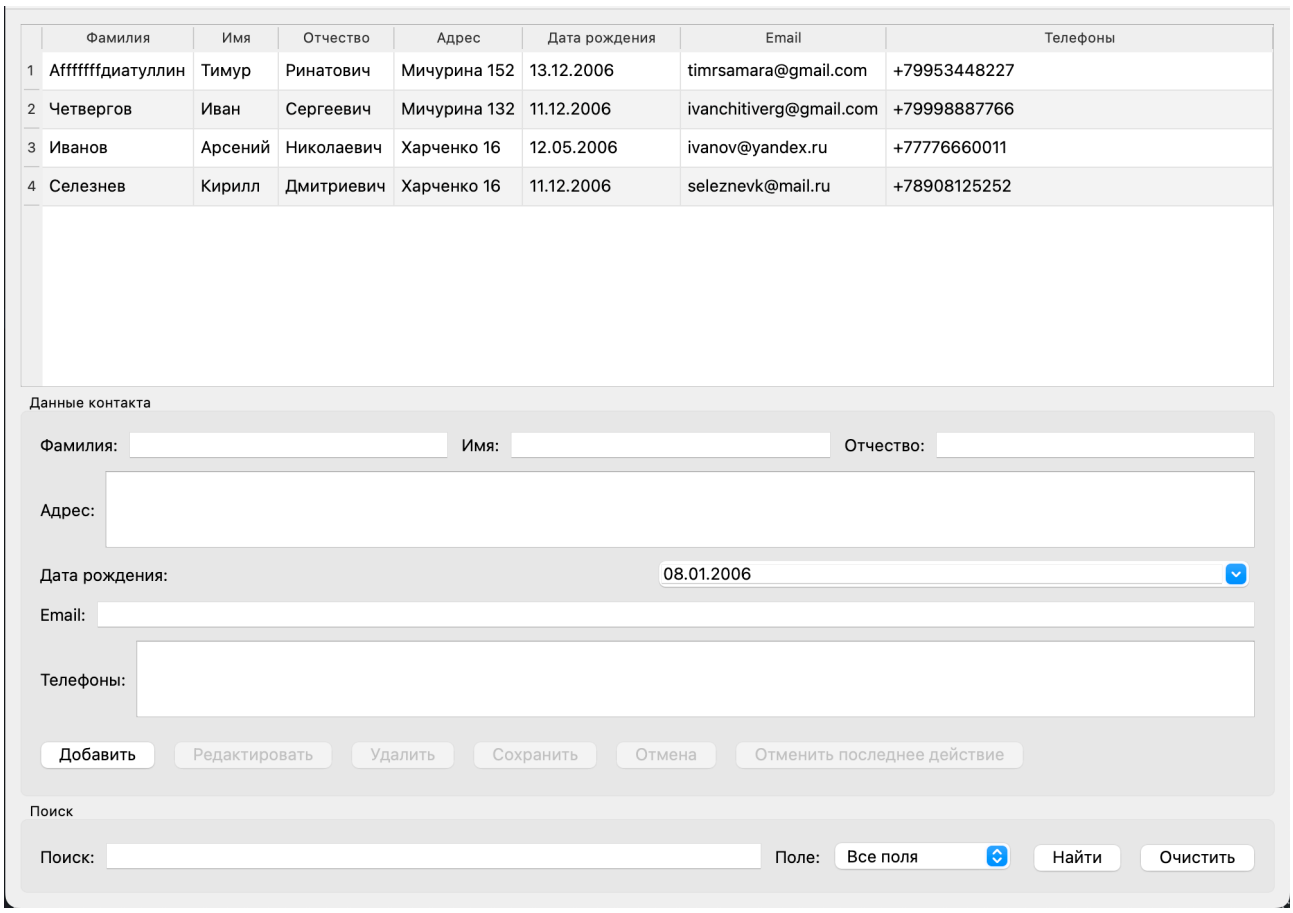

Рис. 2: Главное окно приложения

Механизм сигналов и слотов:

Qt использует архитектуру сигналов и слотов для обработки событий. Основные соединения в приложении:

```
// Кнопки управления
connect(m_addButton, &QPushButton::clicked,
        this, &MainWindow::addContact);
connect(m_editButton, &QPushButton::clicked,
        this, &MainWindow::editContact);
connect(m_deleteButton, &QPushButton::clicked,
        this, &MainWindow::deleteContact);
connect(m_undoButton, &QPushButton::clicked,
        this, &MainWindow::undoLastAction);

// Поиск
connect(m_searchButton, &QPushButton::clicked,
        this, &MainWindow::searchContacts);
connect(m_clearSearchButton, &QPushButton::clicked,
        this, &MainWindow::clearSearch);

// Таблица
connect(m_table, &QTableWidget::itemSelectionChanged,
        this, &MainWindow::onTableSelectionChanged);

// Автосохранение
```

```
22 connect(m_storage, &ContactStorage::dataChanged,
23         this, &MainWindow::saveContacts);
```

Listing 7: Соединения сигналов и слотов

3.3. Ключевые алгоритмы

3.3.1. Алгоритм добавления контакта

Процесс добавления нового или обновления существующего контакта включает несколько этапов проверки и обработки данных. Блок-схема представлена на рис. 3.

Основные этапы работы метода `addContact`:

- проверка корректности заполнения формы методом `validateForm`; при обнаружении ошибок пользователю выводится сообщение, выполнение метода прекращается;
- проверка предупреждений о форматировании данных (например, нестандартное оформление ФИО или номеров телефонов) с возможностью подтверждения продолжения пользователем; при отказе выполнение метода прекращается;
- создание объекта `Contact` на основе данных формы;
- проверка на наличие дубликатов по ключевым полям (ФИО и Email); при обнаружении дубликата пользователю выводится сообщение об ошибке, выполнение метода прекращается;
- если выполняется редактирование существующего контакта:
  - сохранение старого состояния контакта для возможности отмены изменений;
  - обновление контакта в хранилище;
  - вывод пользователю сообщения об успешном обновлении.
- если добавляется новый контакт:
  - добавление контакта в хранилище;
  - фиксация действия для истории изменений;
  - вывод пользователю сообщения об успешном добавлении.
- завершение обработки:
  - очистка формы ввода;
  - обновление таблицы контактов;

– сохранение данных в файл;

– сброс флагов редактирования и управление состоянием кнопок интерфейса.

Метод addContact():

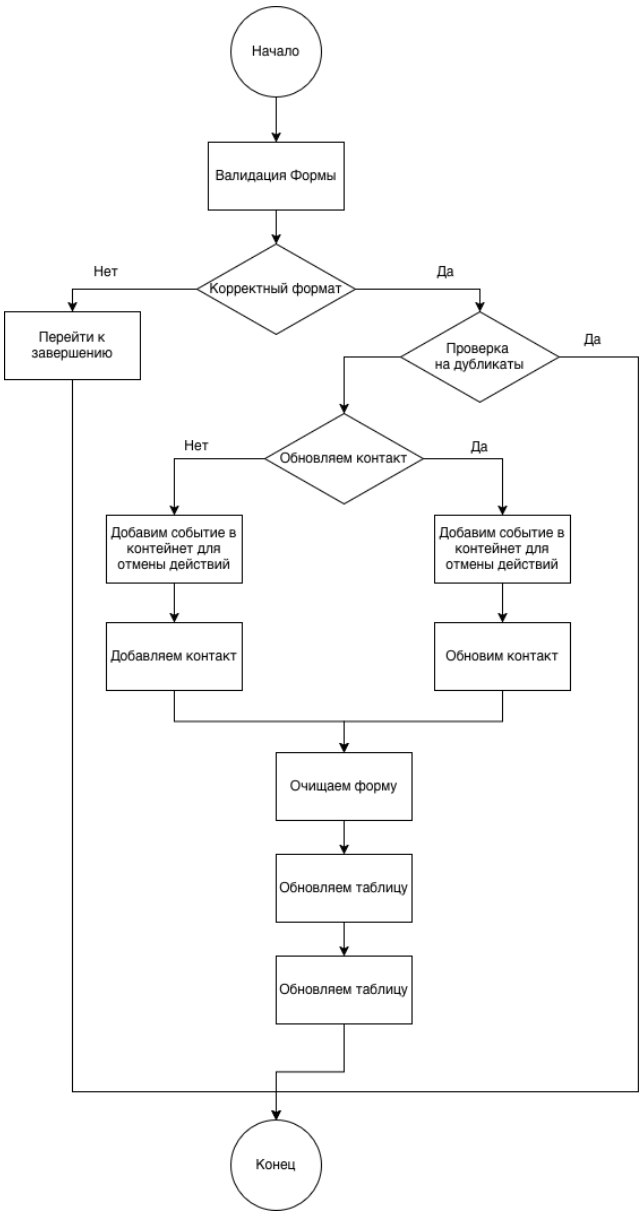

Рис. 3: Блок-схема метода addContact()

Метода editContact():

Процесс редактирования существующего контакта включает несколько этапов проверки и подготовки данных. Блок-схема представлена на рис. 4.

Основные этапы работы метода `editContact`:

- получение выбранной строки из таблицы контактов;

- проверка, что контакт выбран; при отсутствии выбора выводится сообщение об ошибке, выполнение метода прекращается;
- извлечение данных выбранного контакта (Фамилия, Имя, Email) из таблицы;
- поиск контакта в хранилище по ключевым полям (Фамилия, Имя, Email); если контакт не найден, выводится сообщение об ошибке и метод завершает работу;
- установка индекса редактируемого контакта и флага редактирования;
- заполнение формы данными выбранного контакта для последующего редактирования;
- изменение состояния элементов интерфейса: блокировка кнопок добавления, редактирования и удаления, активация кнопок сохранения и отмены.

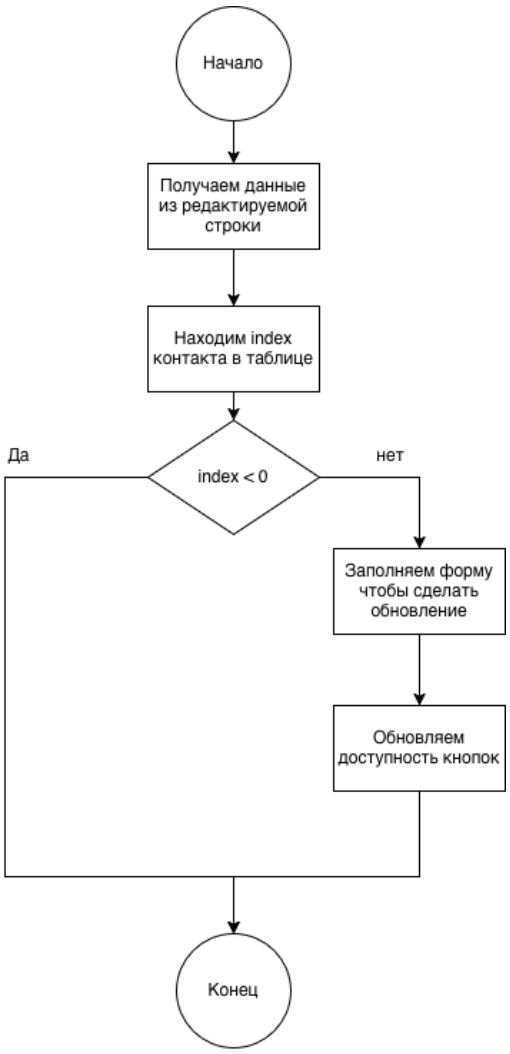

Рис. 4: Блок-схема метода editContact()

3.3.2. Алгоритм поиска контактов

Приложение поддерживает два типа поиска:

- **Поиск по текстовым полям** (имя, фамилия, отчество, email, телефон, адрес) - поиск подстроки без учета регистра;
- **Поиск по дате рождения** - точное совпадение, диапазон ”до даты”, ”после даты”.

Метод searchContacts():

Процесс поиска контактов в таблице включает фильтрацию и отображение результатов. Блок-схема представлена на рис. 5.

Основные этапы работы метода `searchContacts`:

- определение выбранного поля поиска;
- если выбран поиск по дате рождения:
  - получение даты и типа поиска (точная дата, год или месяц и год);
  - фильтрация контактов по дате;
  - обновление таблицы с найденными контактами.
- если выбран текстовый поиск:
  - получение текста поиска;
  - при пустом поисковом запросе таблица заполняется всеми контактами;
  - фильтрация контактов по выбранному полю или по всем полям;
  - обновление таблицы с найденными контактами.

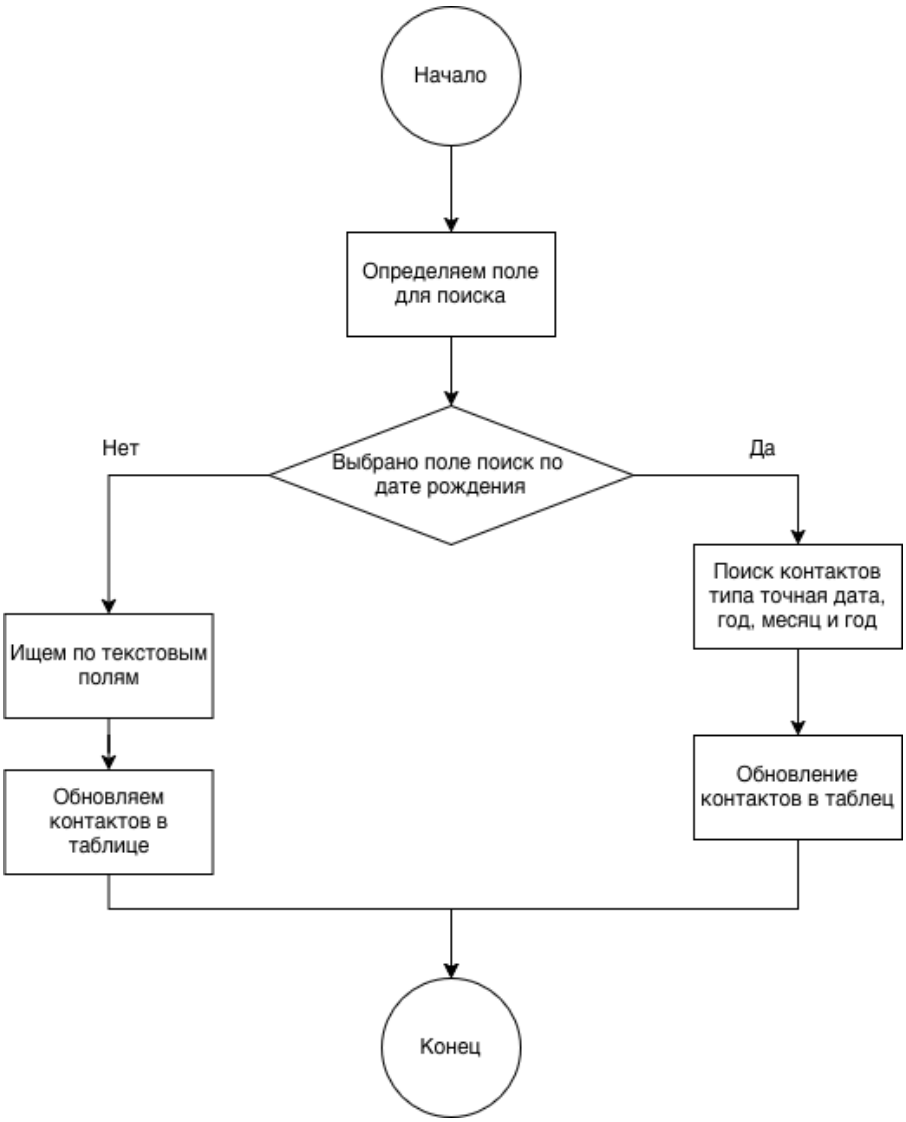

Рис. 5: Блок-схема метода searchContacts()

3.3.3. Механизм отмены действий

Для реализации функции отмены последнего действия используется структура `ContactAction` и стек `QStack<ContactAction>`.

Структура ContactAction:

```
struct ContactAction {
    enum Type { Add, Edit, Delete };
    Type type;          // Тип действия
    Contact contact;    // Сохраненный контакт
    int index;          // Индекс для( Edit и Delete)
};
```

Listing 8: Структура для хранения действия

Метод undoLastAction:

Рис. 6: Блок-схема метода undoLastAction()

При каждом изменении данных (добавление, редактирование, удаление) информация о действии сохраняется в стек методом `pushAction`. Это позволяет откатить последнюю операцию одним нажатием кнопки ”Отменить”.

3.3.4. Проверка на дубликаты

Перед добавлением или редактированием контакта выполняется проверка на наличие идентичных записей. Контакты считаются дубликатами, если совпадают все поля, кроме списка телефонов.

Псевдокод метода checkForDuplicates:

Функция checkForDuplicates(contact, excludeIndex):
для i от 0 до m_storage.contactCount() - 1:
если i == excludeIndex:
продолжить следующую итерацию
existing = m_storage.contacts()[i]
если existing.firstName() == contact.firstName() и
existing.lastName() == contact.lastName() и
existing.middleName() == contact.middleName() и
existing.birthDate() == contact.birthDate() и
existing.email() == contact.email() и
existing.address() == contact.address():
вернуть true
вернуть false

Если обнаружен дубликат, пользователю выводится предупреждение с вопросом о продолжении добавления.

3.4. Сортировка данных

Сортировка контактов в таблице выполняется автоматически при клике на заголовок столбца благодаря встроенному механизму `QTableWidget`. Метод `setSortingEnabled(true)` активирует эту функциональность.

При каждой загрузке данных из файла список контактов предварительно сортируется по фамилии, имени и отчеству с помощью оператора `operator<`, реализованного в классе `Contact`:

```
bool Contact::operator<(const Contact& other) const
{
    if (m_lastName != other.m_lastName) {
        return m_lastName < other.m_lastName;
    }
    if (m_firstName != other.m_firstName) {
        return m_firstName < other.m_firstName;
    }
    return m_middleName < other.m_middleName;
```

```
10 }
```

Listing 9: Оператор сравнения для сортировки

Данный подход обеспечивает единообразие отображения данных при каждом запуске приложения.

4. Тестирование приложения

В данном разделе представлены основные сценарии использования приложения с описанием действий пользователя и ожидаемого поведения системы.

4.1. Начальное состояние приложения

При первом запуске приложения отображается главное окно с пустой таблицей контактов и формой ввода данных (см. рисунок 7). Если файл `phonebook.json` не существует, он будет автоматически создан при сохранении первого контакта.

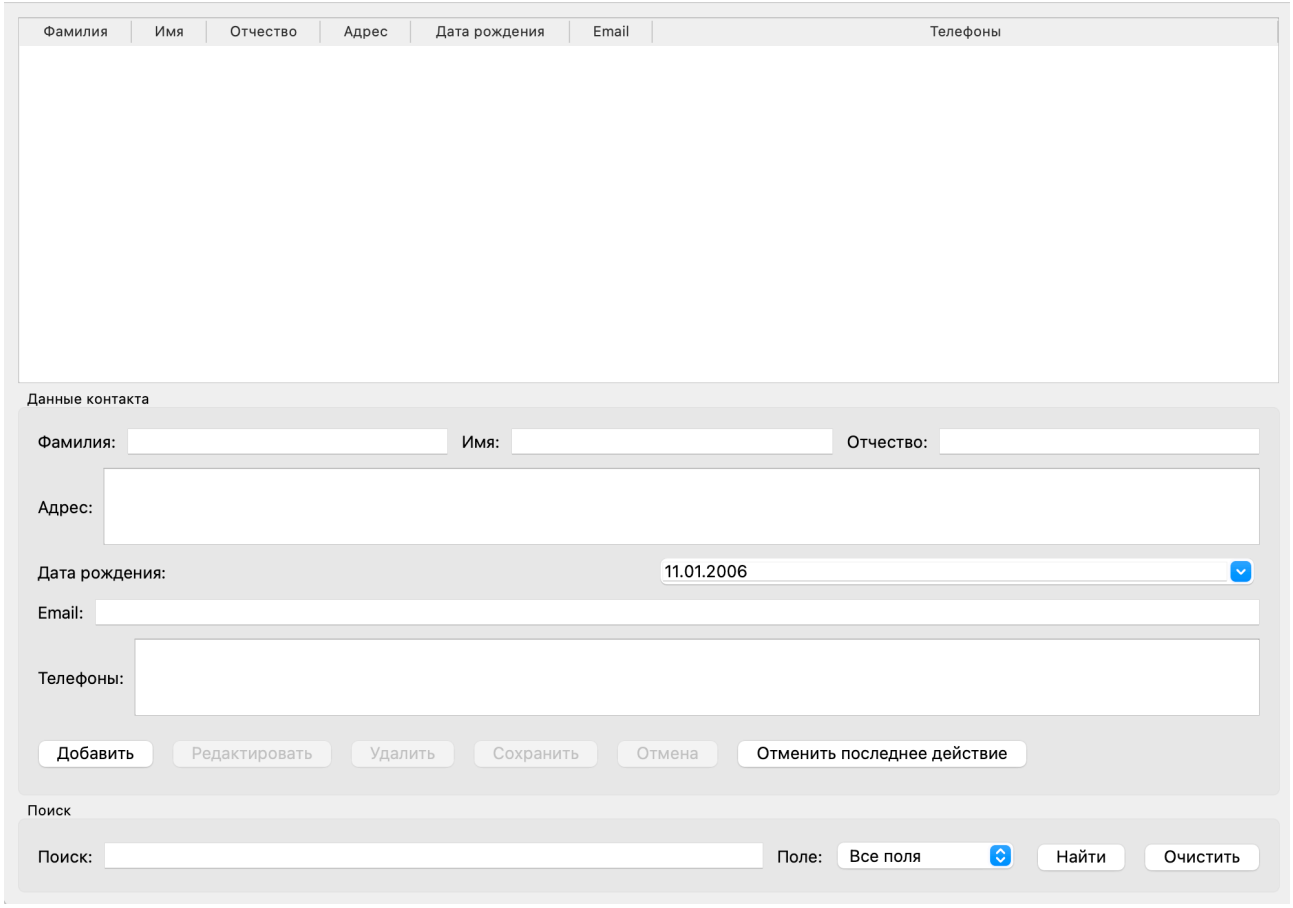

Рис. 7: Начальное состояние приложения при первом запуске

Элементы интерфейса:

- **Форма ввода данных** - содержит поля для всех атрибутов контакта;
- **Кнопки управления** - ”Добавить”, ”Редактировать”, ”Удалить”, ”Отменить”;
- **Панель поиска** - поле ввода, выбор поля для поиска, фильтры по дате;
- **Таблица контактов** - отображает список всех записей с возможностью сортировки;
- **Кнопки файловых операций** - ”Сохранить” и ”Загрузить”.

4.2. Сценарий 1: Добавление нового контакта

4.2.1. Шаг 1: Заполнение формы

Пользователь заполняет все поля формы корректными данными (см. рисунок 8):

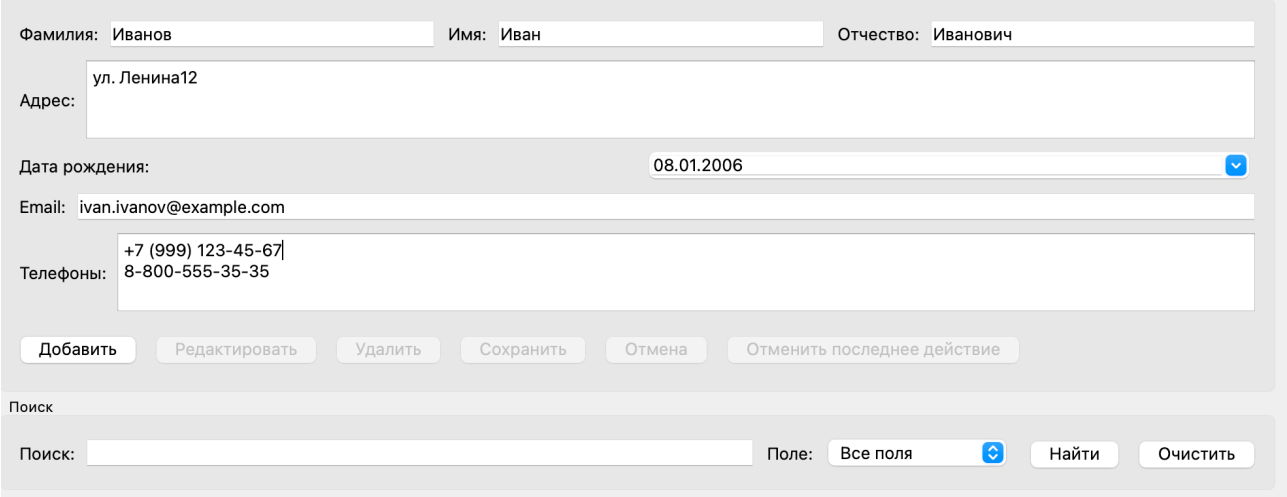

Рис. 8: Заполненная форма для добавления контакта

4.2.2. Шаг 2: Нормализация данных

При вводе данных класс `ContactValidator` автоматически нормализует некоторые поля:

- **Имя ”иван”** преобразуется в **”Иван”** (первая буква заглавная);
- **Телефон ”8-800-555-35-35”** преобразуется в **”+78005553535”** (международный формат);
- **Email ”Ivan.Ivanov@Example.COM”** преобразуется в **”ivan.ivanov@example.com”** (нижний регистр).

Если происходит нормализация, пользователю выводится предупреждение с указанием изменений (см. рисунок 9).

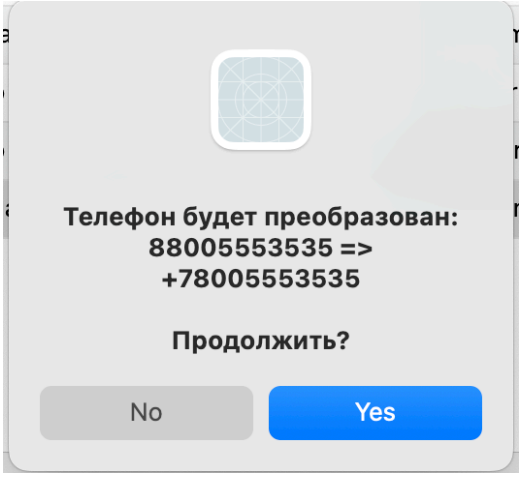

Рис. 9: Предупреждение о нормализации данных

4.2.3. Шаг 3: Успешное добавление

После нажатия кнопки ”Добавить” происходит:

1. Валидация всех полей формы;
2. Проверка на дубликаты;
3. Добавление контакта в `ContactStorage`;
4. Автоматическое сохранение в файл `phonebook.json`;
5. Отображение контакта в таблице;
6. Сохранение действия в стек отмены;
7. Очистка полей формы;
8. Показ информационного сообщения об успешном добавлении (см. рисунок 10).

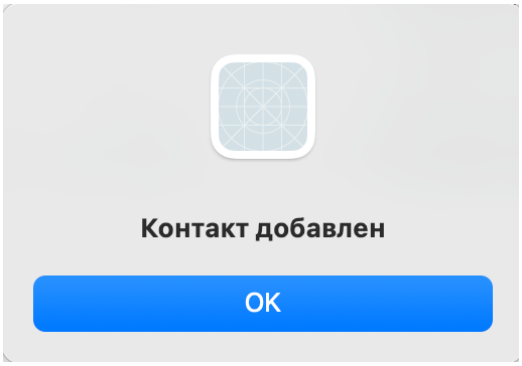

Рис. 10: Контакт успешно добавлен в таблицу

4.3. Сценарий 2: Валидация некорректных данных

4.3.1. Пример 1: Некорректное имя

Пользователь вводит имя, начинающееся с дефиса: ”-Иван”. Система выводит ошибку валидации (см. рисунок 11).

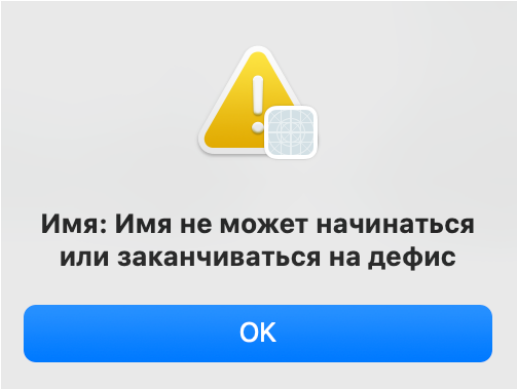

Рис. 11: Ошибка валидации имени

Сообщение об ошибке: ”Имя не может начинаться или заканчиваться на дефис”

4.3.2. Пример 2: Некорректный телефон

Пользователь вводит телефон без знака ”+”: ”79993332211”.
Класс `ContactValidator` автоматически добавляет ”+” в начало, преобразуя номер в ”+79993332211”.

Если длина телефона меньше 10 цифр, выводится ошибка: ”Телефон должен содержать минимум 10 цифр” (см. рисунок 12).

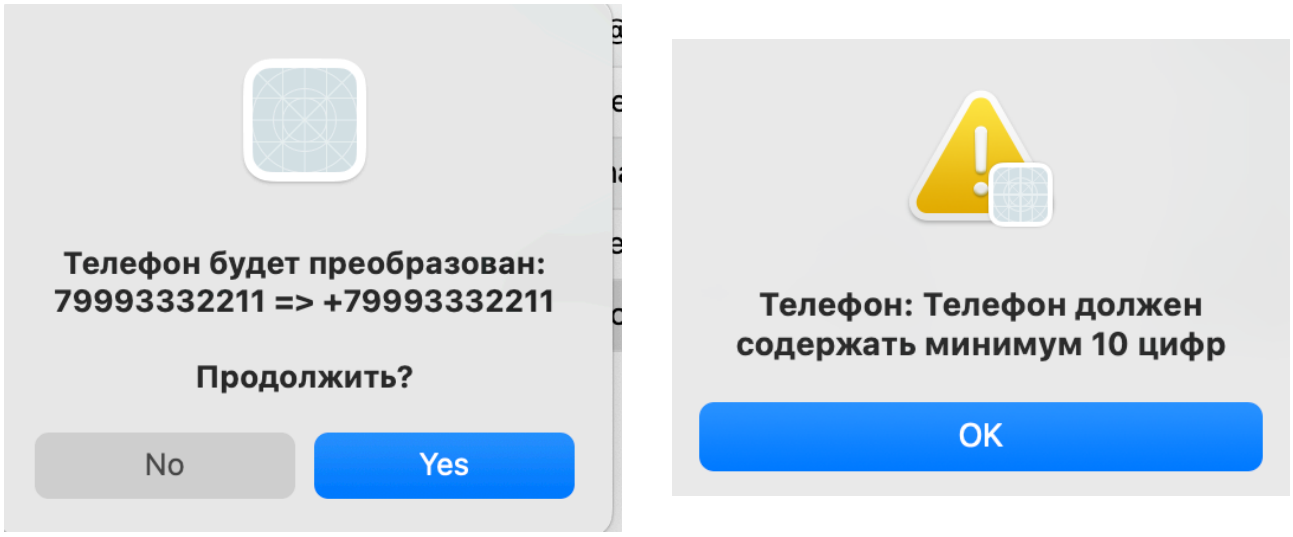

Рис. 12: Некорректный ввод

4.3.3. Пример 3: Некорректный email

Пользователь вводит email без символа ”@”: ”ivanovexample.com”(см. рис 13). Система выводит ошибку:

Сообщение об ошибке: ”Email должен содержать символ @”

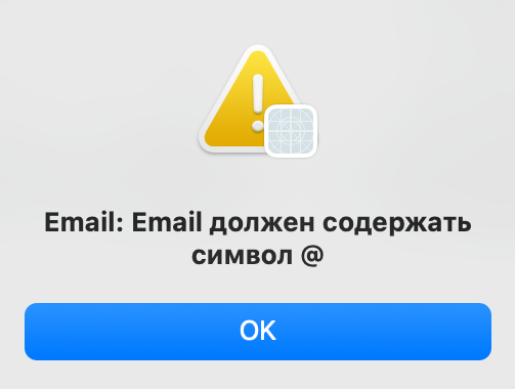

Рис. 13: Некорректный email

4.4. Сценарий 3: Редактирование существующего контакта

4.4.1. Шаг 1: Выбор контакта

Пользователь кликает на строку в таблице. При этом:

- Активируются кнопки ”Редактировать” и ”Удалить”;
- Строка в таблице подсвечивается.

4.4.2. Шаг 2: Переход в режим редактирования

После нажатия кнопки ”Редактировать” (см. рисунок 14):

- Приложение переходит в режим редактирования (`m_isEditing = true`);
- Кнопка ”Добавить” меняет текст на ”Сохранить”;
- Кнопки ”Редактировать” и ”Удалить” становятся неактивными;
- Появляется кнопка ”Отмена”;
- Таблица блокируется (нельзя выбрать другую строку);
- Все поля формы доступны для изменения.

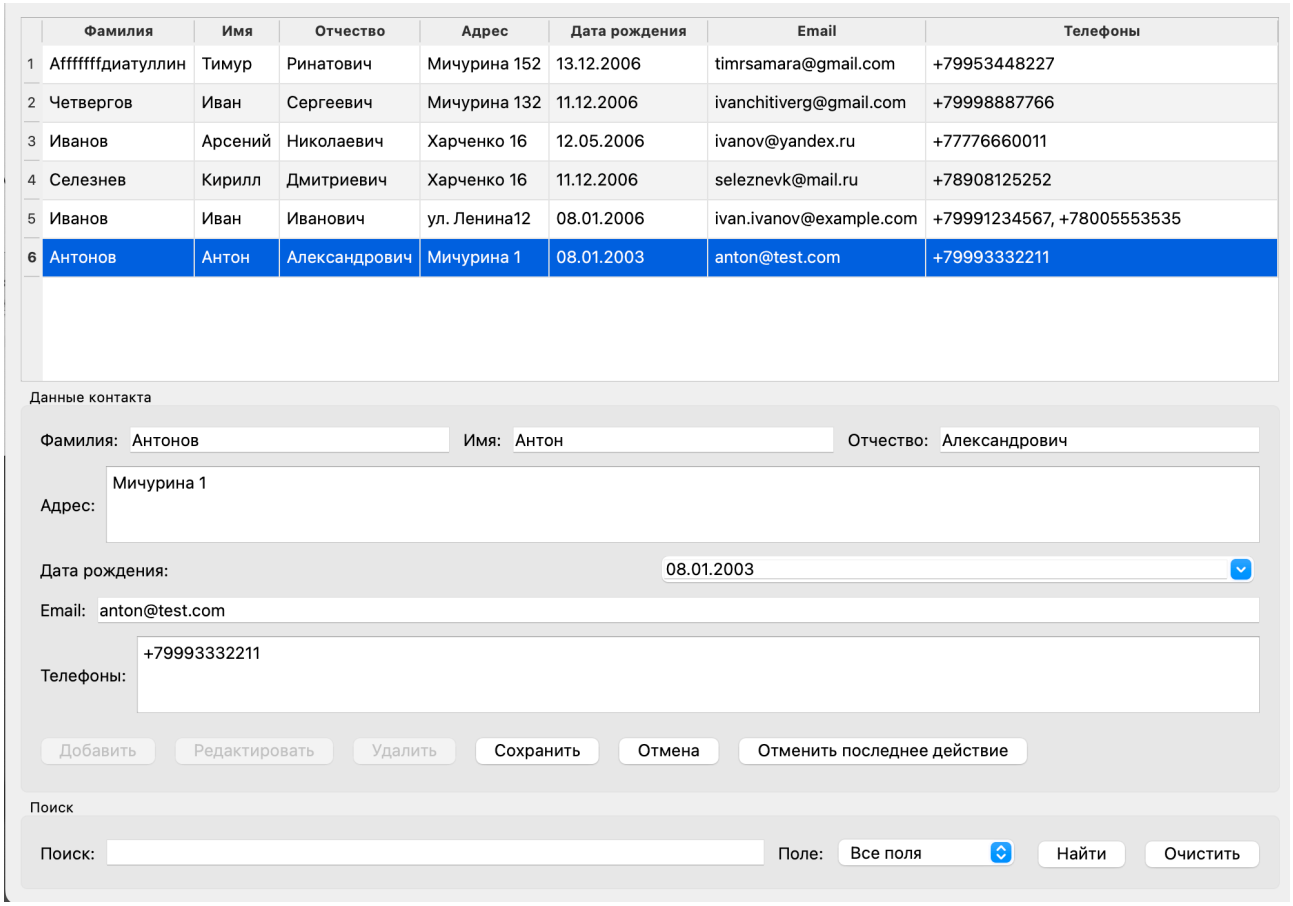

Рис. 14: Режим редактирования контакта

4.4.3. Шаг 3: Изменение данных и сохранение

Пользователь изменяет необходимые поля (например, номер телефона) и нажимает ”Сохранить”. Происходит:

1. Валидация изменённых данных;
2. Проверка на дубликаты (исключая текущий редактируемый контакт);
3. Обновление контакта в `ContactStorage`;
4. Сохранение старой версии контакта в стек отмены;
5. Автоматическое сохранение в файл;
6. Возврат в режим просмотра;
7. Показ сообщения об успешном редактировании(см. рисунок 15).

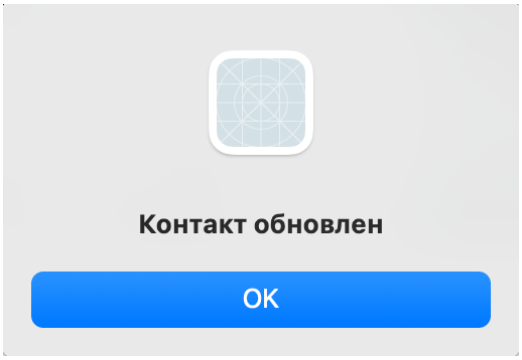

Рис. 15: Контакт успешно отредактирован

4.4.4. Шаг 4: Отмена редактирования

При нажатии кнопки ”Отмена”:

- Все изменения откатываются;
- Приложение возвращается в режим просмотра;
- Поля формы очищаются;
- Таблица снова становится активной.

4.5. Сценарий 4: Удаление контакта

4.5.1. Шаг 1: Выбор контакта для удаления

Пользователь выбирает строку в таблице и нажимает кнопку ”Удалить”.

4.5.2. Шаг 2: Подтверждение удаления

Система выводит диалоговое окно с запросом подтверждения (см. рисунок 16).

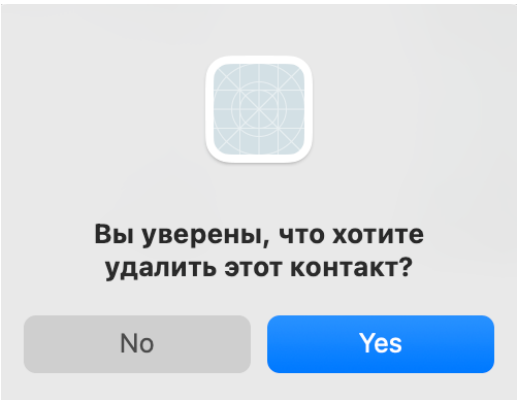

Рис. 16: Запрос подтверждения удаления контакта

4.5.3. Шаг 3: Удаление и сохранение

Если пользователь подтверждает удаление:

1. Контакт сохраняется в стек отмены;
2. Контакт удаляется из `ContactStorage`;
3. Автоматическое сохранение в файл;
4. Обновление таблицы;
5. Показ сообщения ”Контакт удалён” (см. рисунок 17).

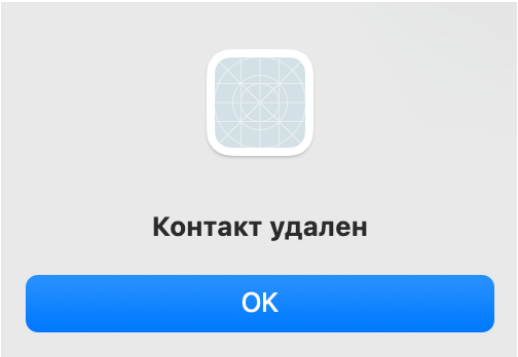

Рис. 17: Контакт удалён из таблицы

Если пользователь отменяет удаление, никаких изменений не происходит.

4.6. Сценарий 5: Поиск контактов

4.6.1. Поиск по текстовым полям

Пользователь вводит текст в поле поиска и выбирает поле для поиска из выпадающего списка (Имя, Фамилия, Отчество, Email, Телефон, Адрес).

Пример: Поиск по фамилии ”Иванов”

Система выполняет поиск подстроки без учета регистра. Все контакты, содержащие ”иванов”, ”Иванов”, ”ИВАНОВ” в поле фамилии, отображаются в таблице (см. рисунок 18).

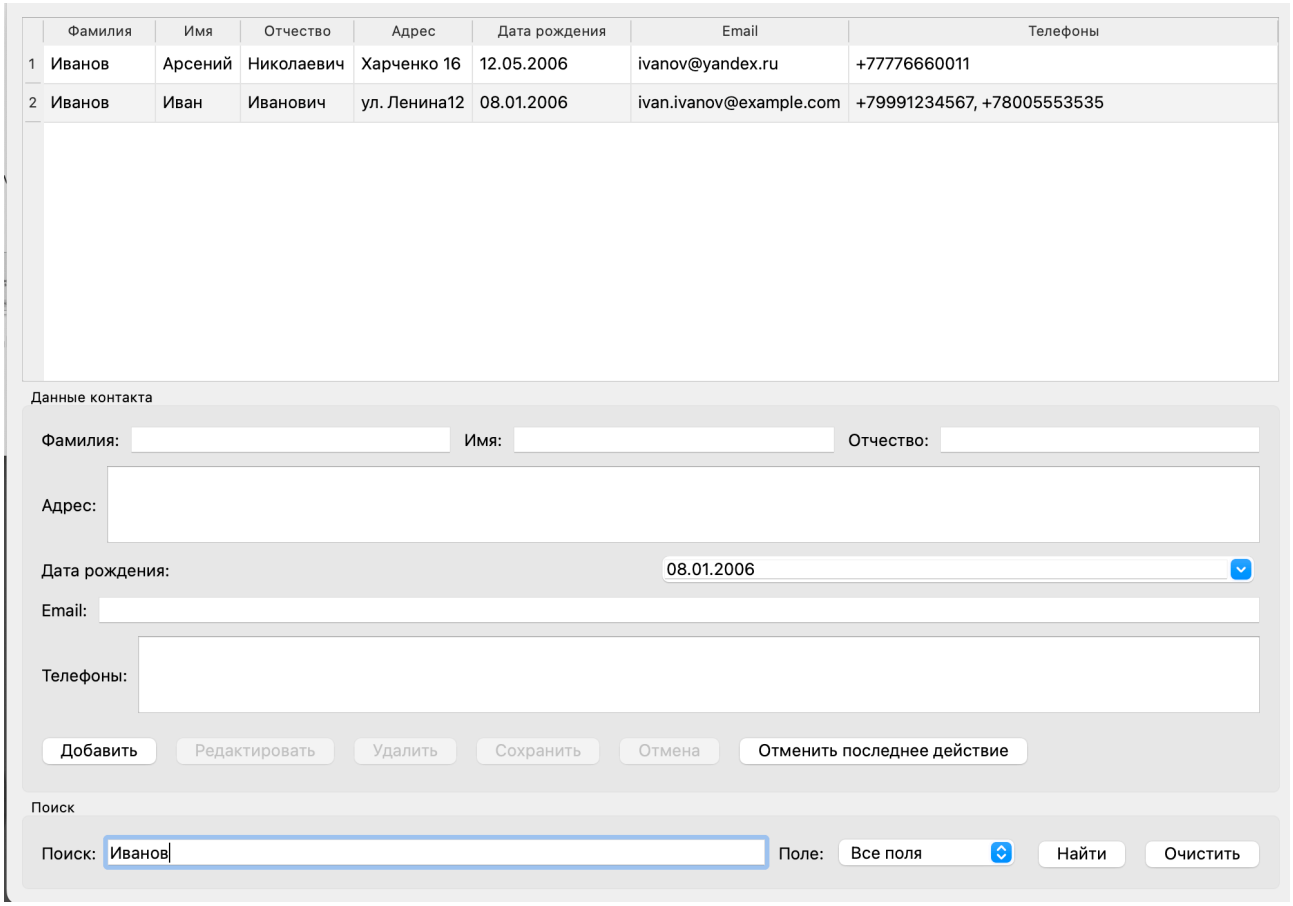

Рис. 18: Результат поиска по фамилии

4.6.2. Поиск по дате рождения

При выборе поля ”Дата рождения” (см. рисунок 19):

- Поле текстового ввода скрывается;
- Отображается `QDateEdit` для выбора даты;
- Отображается `QComboBox` с типами сравнения: ”Точная дата”, ”Год”, ”Месяц и год”.

Рис. 19: Панель поиска по дате рождения

Пример 1: Точное совпадение

Пользователь выбирает дату ”08.01.2006” и тип ”Точное совпадение”. В таблице отображаются только контакты, родившиеся именно в этот день (см. рисунок 20).

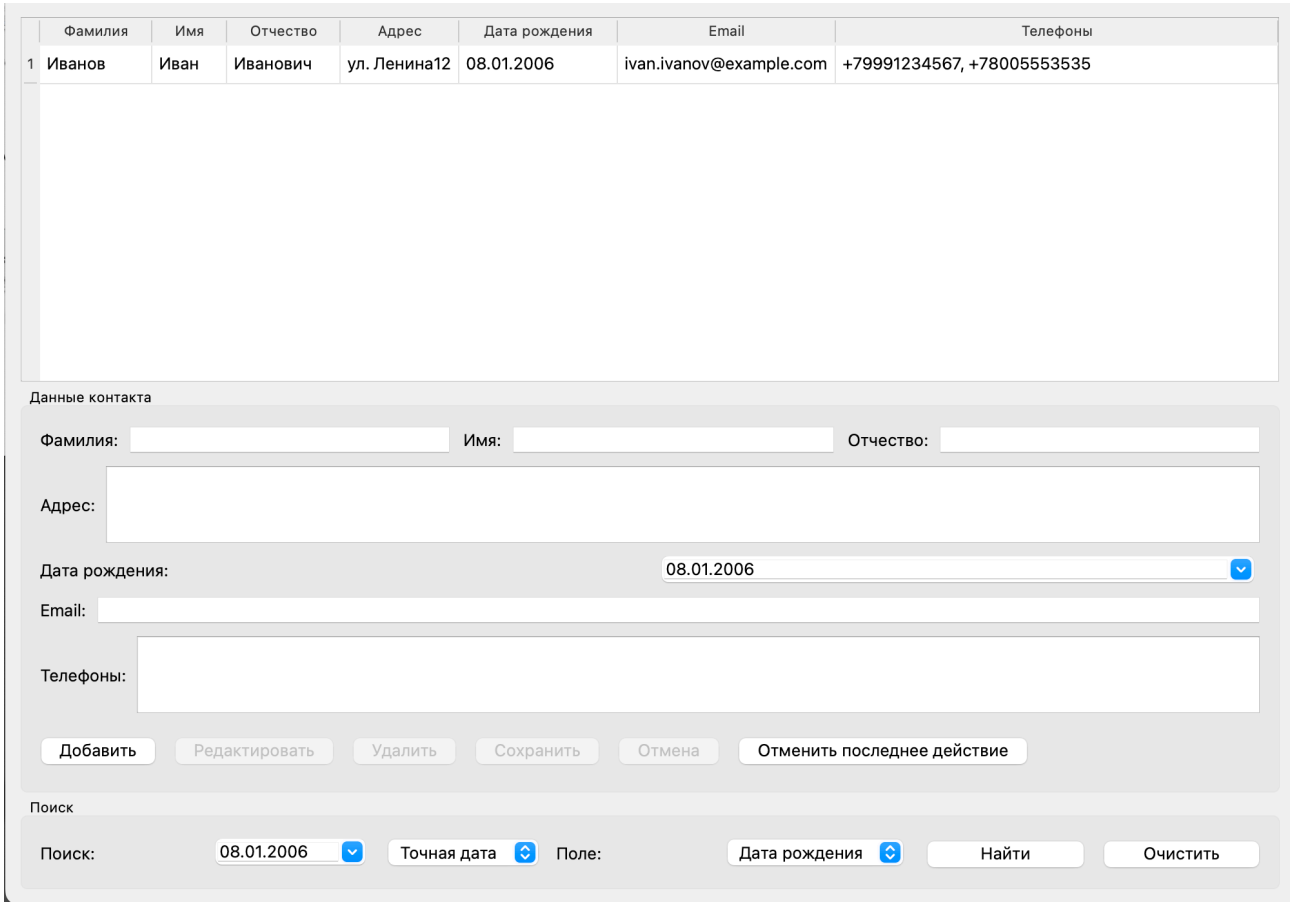

Рис. 20: Результат поиска по дате (08.01.2006)

Пример 2: Год

Пользователь выбирает дату ”08.01.2006” и тип ”Год. В таблице отображаются все контакты, родившиеся в 2006 году (см. рисунок 21).

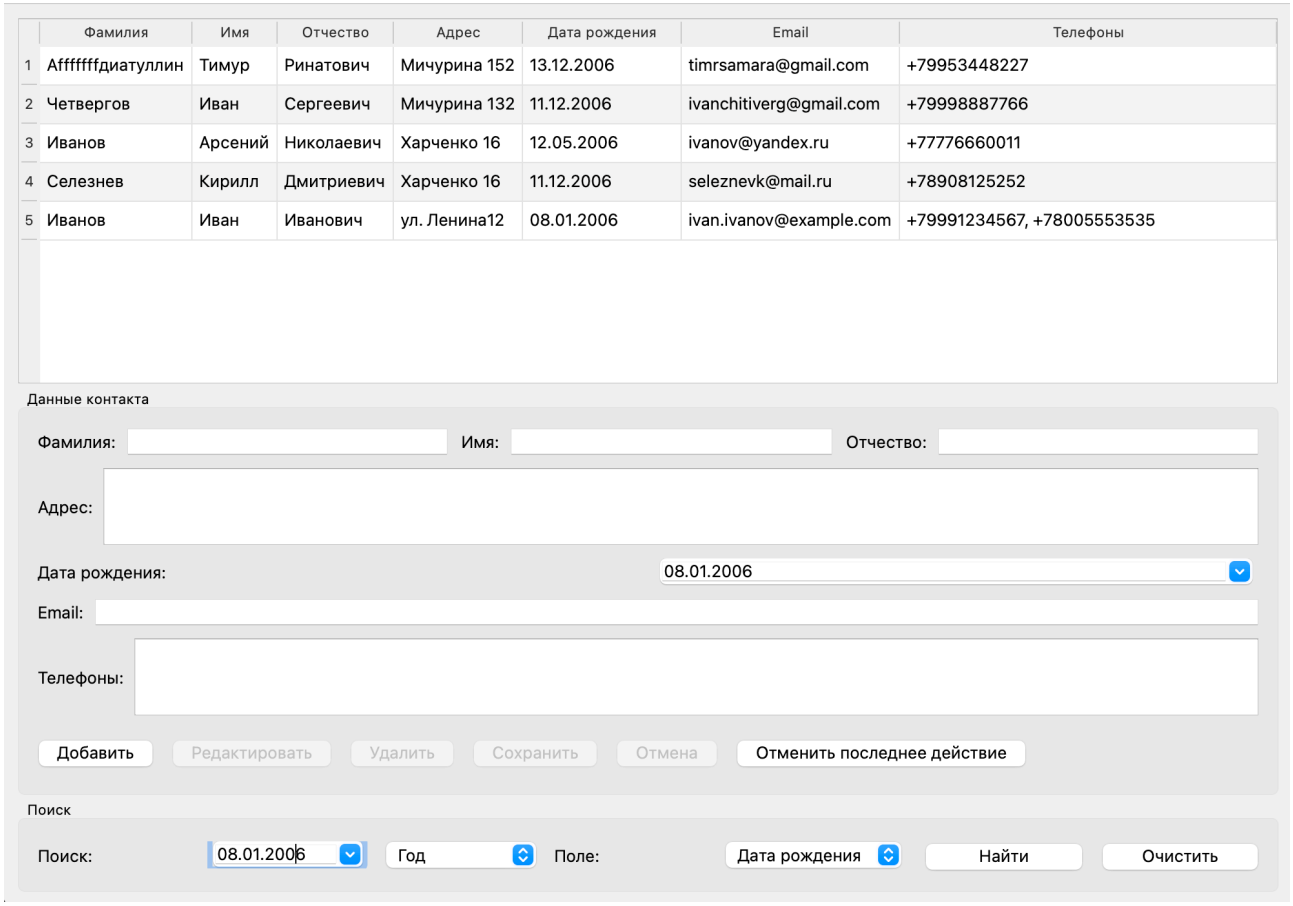

Рис. 21: Результат поиска по году рождения (2006)

Пример 3: Месяц и год

Пользователь выбирает дату ”08.12.2006” и тип ”Месяц и год”. В таблице отображаются все контакты, родившиеся в декабре 2006 года (см. рисунок 22).

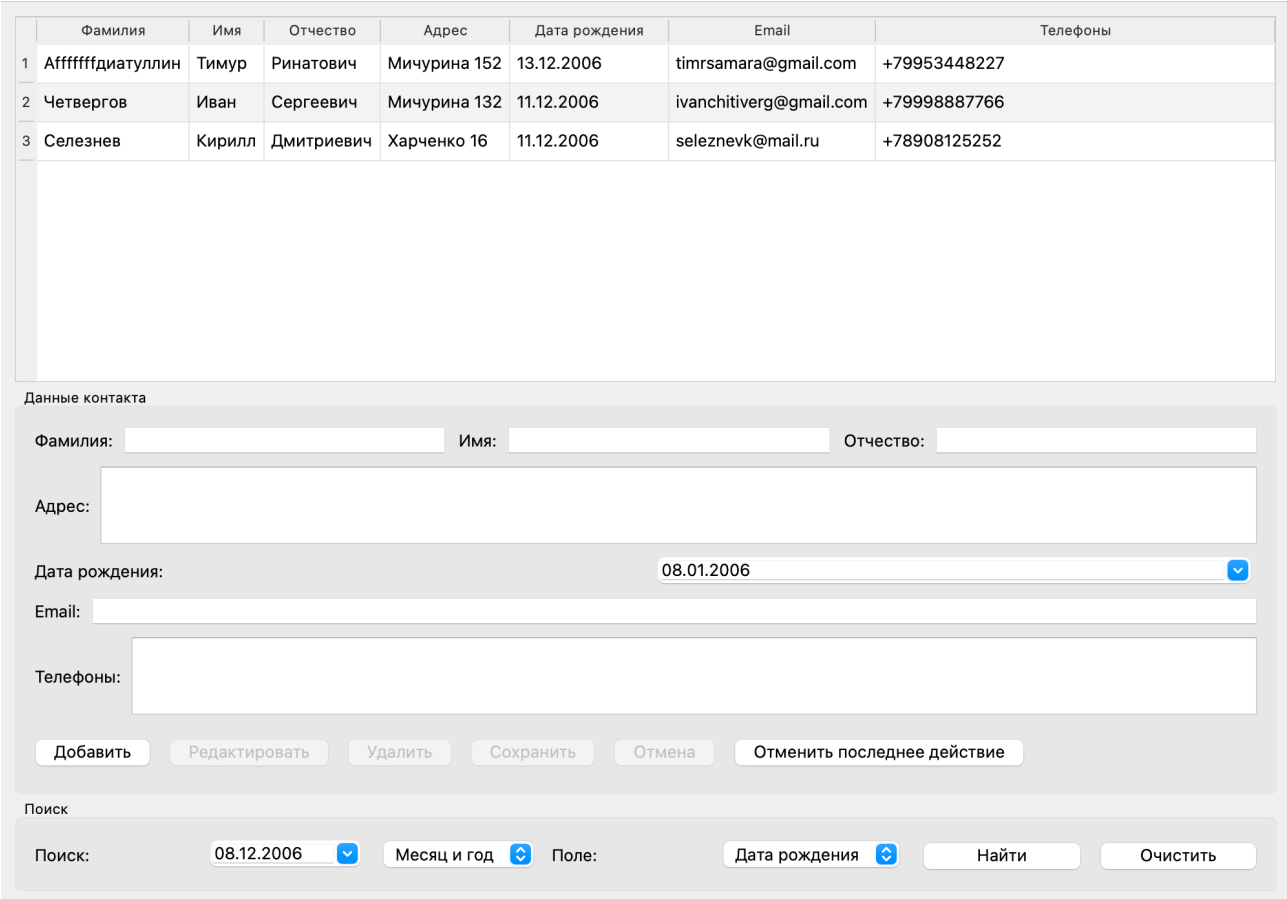

Рис. 22: Результат поиска по месяцу и году (декабрь 2005)

4.6.3. Сброс поиска

При нажатии кнопки ”Сбросить поиск”:

- Поле поиска очищается;
- Таблица отображает все контакты;
- Восстанавливается исходная сортировка.

Если результаты поиска пусты, выводится сообщение ”Ничего не найдено”.

4.7. Сценарий 6: Отмена последнего действия

4.7.1. Отмена добавления

Пользователь добавил контакт, но сразу нажал кнопку ”Отменить”. Происходит (см. рисунок 23 и 24):

1. Извлечение последнего действия из стека (`m_undoStack.pop()`);
2. Определение типа действия (Add);

3. Удаление контакта из `ContactStorage`;
4. Автоматическое сохранение в файл;
5. Обновление таблицы;
6. Показ сообщения ”Добавление отменено”.

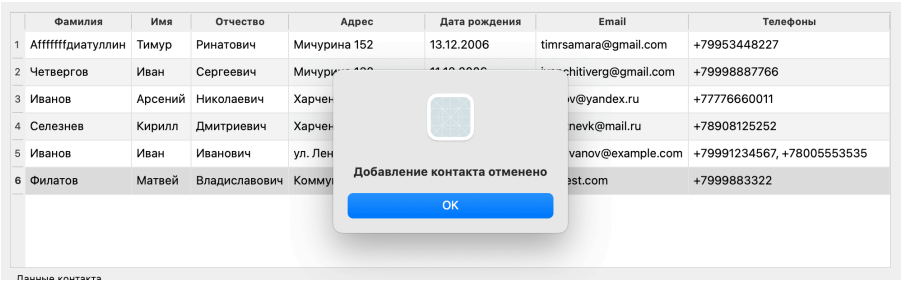

Рис. 23: Сообщение об отмене последнего действия

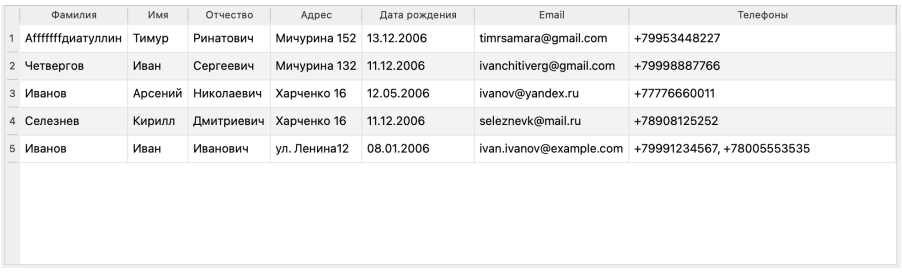

Рис. 24: Таблица после отмены дейсвия

4.7.2. Отмена редактирования

Пользователь отредактировал контакт и нажал ”Отменить”. Система восстанавливает старую версию контакта из стека и сохраняет изменения.

4.7.3. Отмена удаления

Пользователь удалил контакт и нажал ”Отменить”. Система восстанавливает удалённый контакт на его прежнюю позицию в списке.

4.7.4. Попытка отмены при пустом стеке

Если стек отмены пуст, при нажатии кнопки ”Отменить” выводится сообщение: ”Нечего отменять”.

4.8. Сценарий 7: Проверка на дубликаты

4.8.1. Обнаружение дубликата

Пользователь пытается добавить контакт с данными, идентичными существующему (кроме телефонов). Система обнаруживает дубликат и выводит предупреждение (см. рисунок 25).

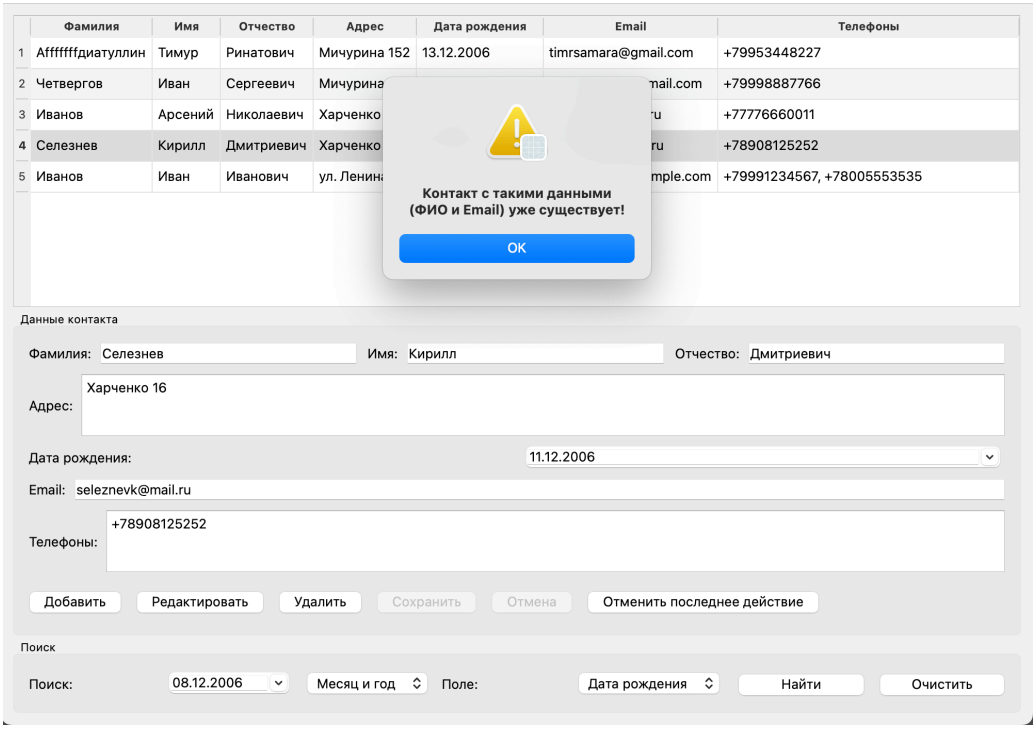

Рис. 25: Предупреждение о дубликате контакта

Проверка на дубликаты учитывает следующие поля:

- Имя, Фамилия, Отчество
- Дата рождения
- Email
- Адрес

Телефоны не учитываются при проверке, поскольку у одного человека может быть несколько номеров.

4.9. Сценарий 8: Сохранение и загрузка данных

4.9.1. Автоматическое сохранение

После каждого изменения данных (добавление, редактирование, удаление) приложение автоматически вызывает метод `ContactStorage::save()`, который записывает все контакты в файл `phonebook.json`.

Формат файла (JSON):

```
[
    {
        "firstName": "Иван",
        "lastName": "Иванов",
        "middleName": "Иванович",
        "address": "г. Москва, ул. Ленина, д. 1, кв. 10",
        "birthDate": "1990-03-15",
        "email": "ivan.ivanov@example.com",
        "phoneNumbers": ["+79991234567", "+78005553535"]
    },
    {
        "firstName": "Петр",
        "lastName": "Петров",
        "middleName": "Петрович",
        "address": "г. СанктПетербург-, Невский пр., д. 50",
        "birthDate": "1985-07-20",
        "email": "petr.petrov@example.com",
        "phoneNumbers": ["+79161234567"]
    }
]
```

Listing 10: Пример phonebook.json

4.9.2. Загрузка при запуске

При запуске приложения автоматически вызывается метод `ContactStorage::loa` который загружает данные из файла. Если файл не существует или содержит некорректные данные, приложение начинает с пустой базы.

4.9.3. Ручное сохранение и загрузка

Кнопки ”Сохранить файл” и ”Загрузить файл” позволяют пользователю вручную управлять файлом данных. При нажатии кнопки ”Загрузить файл” все несохранённые изменения будут потеряны (после подтверждения).

4.10. Итоги тестирования

Все сценарии использования приложения были успешно протестированы. Приложение корректно обрабатывает:

- Валидацию пользовательского ввода с помощью регулярных выражений;
- Нормализацию данных перед сохранением;
- Операции CRUD (создание, чтение, обновление, удаление);
- Поиск по текстовым полям и по дате рождения;

- Сортировку данных по любому столбцу;
- Отмену последнего действия;
- Проверку на дубликаты;
- Автоматическое сохранение в файл формата JSON.

Графический интерфейс отзывчив и интуитивно понятен. Все операции сопровождаются информационными сообщениями, что повышает удобство использования приложения.

Заключение

В ходе выполнения лабораторной работы было разработано приложение «Телефонный справочник» с графическим пользовательским интерфейсом, реализованное с использованием фреймворка Qt.

Реализованный функционал

В рамках работы был полностью реализован требуемый функционал:

- **Добавление контактов** - реализован ввод данных о контактах с валидацией всех полей и автоматической нормализацией введённых данных;
- **Редактирование контактов** - реализована возможность изменения всех полей существующих контактов с сохранением валидации;
- **Удаление контактов** - реализовано удаление контактов с подтверждением действия;
- **Поиск контактов** - реализован поиск по всем текстовым полям, а также специализированный поиск по дате рождения с поддержкой трёх режимов (точная дата, год, месяц и год);
- **Сортировка данных** - реализована сортировка по всем полям таблицы с возможностью изменения направления сортировки;
- **Отмена действий** - реализован механизм отмены последнего действия (добавление, редактирование, удаление) с использованием стека операций;
- **Проверка на дубликаты** - реализована защита от добавления контактов с идентичными данными;
- **Сохранение и загрузка данных** - реализовано автоматическое сохранение данных в файл формата JSON после каждой операции и загрузка при запуске приложения.

Валидация данных

Была реализована комплексная система валидации вводимых данных с использованием регулярных выражений:

- **Имя, фамилия, отчество** - проверка на соответствие требованиям (буквы, цифры, дефисы, пробелы), запрет начала и окончания на дефис, автоматическая нормализация регистра символов;

- **Телефонные номера** - проверка формата международного номера, автоматическое преобразование российских номеров, начинающихся с «8», в формат «+7», удаление лишних символов форматирования;
- **Email** - проверка корректности адреса электронной почты с валидацией имени пользователя и доменного имени, автоматическое удаление пробелов и приведение к нижнему регистру;
- **Дата рождения** - проверка корректности даты, контроль, что дата находится в прошлом и не превышает разумный диапазон (150 лет).

Система валидации предоставляет пользователю информативные сообщения об ошибках, указывающие на конкретное поле и причину отклонения введённых данных.

Изученные технологии

В процессе разработки были изучены и освоены следующие технологии и компоненты фреймворка Qt:

- **Система сигналов и слотов Qt** - механизм взаимодействия объектов, обеспечивающий слабую связанность компонентов интерфейса;
- **Классы контейнеров Qt** - использовались классы `QList` для хранения списка контактов, `QStringList` для списка телефонных номеров, `QStack` для реализации отмены действий;
- **Виджеты Qt** - освоена работа с классами `QMainWindow`, `QTableWidget`, `QLineEdit`, `QTextEdit`, `QDateEdit`, `QComboBox`, `QPushButton`, `QGroupBox`, `QMessageBox`;
- **Работа с JSON** - использованы классы `QJsonDocument`, `QJsonObject`, `QJsonArray` для сериализации и десериализации данных;
- **Работа с файлами** - использован класс `QFile` для чтения и записи данных в файловую систему;
- **Регулярные выражения** - использован класс `QRegularExpression` для валидации вводимых пользователем данных;
- **Работа с датами** - освоена работа с классом `QDate` и виджетом `QDateEdit` с всплывающим календарём;
- **Система компоновки** - использованы классы `QVBoxLayout` и `QHBoxLayout` для автоматической компоновки элементов интерфейса.

Архитектурные решения

При разработке приложения были применены принципы объектно-ориентированного программирования и разделения ответственности:

- **Разделение модели и представления** - класс `Contact` инкапсулирует данные контакта, класс `MainWindow` отвечает за отображение и взаимодействие с пользователем;
- **Инкапсуляция логики валидации** - класс `ContactValidator` содержит всю логику проверки и нормализации данных, что обеспечивает повторное использование и упрощает тестирование;
- **Изоляция работы с хранилищем** - класс `ContactStorage` инкапсулирует операции чтения и записи файлов, что позволяет легко изменить формат хранения данных без модификации остальных компонентов.

Такая архитектура обеспечивает хорошую расширяемость приложения и упрощает его сопровождение.

Полученные навыки

В результате выполнения лабораторной работы были получены следующие знания и навыки:

- Проектирование графических пользовательских интерфейсов с использованием фреймворка Qt;
- Работа с механизмом сигналов и слотов для обработки событий;
- Применение регулярных выражений для валидации и нормализации данных;
- Работа с форматом JSON для хранения структурированных данных;
- Организация кода в соответствии с принципами ООП;
- Использование контейнеров и алгоритмов стандартной библиотеки Qt;
- Создание интуитивно понятных пользовательских интерфейсов.

Разработанное приложение полностью соответствует поставленным требованиям и может быть использовано в качестве основы для более сложных систем управления контактной информацией.

Список литературы

[1] Qt Documentation. Официальная документация Qt.
URL: https://doc.qt.io/ (дата обращения: 05.01.2026)

[2] Signals & Slots | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/signalsandslots.html (дата обращения: 05.01.2026)

[3] JSON Support in Qt | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/json.html (дата обращения: 05.01.2026)

[4] QRegularExpression Class | Qt Core 6.x.
URL: https://doc.qt.io/qt-6/qregularexpression.html (дата обращения: 05.01.2026)

[5] Qt Widgets 6.x.
URL: https://doc.qt.io/qt-6/qtwidgets-index.html (дата обращения: 05.01.2026)

[6] Regular expressions library (since C++11) - cppreference.com.
URL: https://en.cppreference.com/w/cpp/regex (дата обращения: 05.01.2026)

Полный исходный код

contact.h

```
#pragma once

#include <QString>
#include <QDate>
#include <QStringList>
#include <QJsonObject>

class Contact
{
public:
    Contact();
    Contact(const QString& firstName, const QString& lastName,
   const QString& middleName,
            const QString& address, const QDate& birthDate, const
   QString& email,
            const QStringList& phoneNumbers);

    QString firstName() const { return m_firstName; }
    QString lastName() const { return m_lastName; }
    QString middleName() const { return m_middleName; }
    QString address() const { return m_address; }
    QDate birthDate() const { return m_birthDate; }
    QString email() const { return m_email; }
    QStringList phoneNumbers() const { return m_phoneNumbers; }

    void setFirstName(const QString& firstName) { m_firstName =
   firstName; }
    void setLastName(const QString& lastName) { m_lastName =
   lastName; }
    void setMiddleName(const QString& middleName) { m_middleName =
   middleName; }
    void setAddress(const QString& address) { m_address = address;
   }
    void setBirthDate(const QDate& birthDate) { m_birthDate =
   birthDate; }
    void setEmail(const QString& email) { m_email = email; }
    void setPhoneNumbers(const QStringList& phoneNumbers) {
   m_phoneNumbers = phoneNumbers; }

    QJsonObject toJson() const;
    static Contact fromJson(const QJsonObject& json);

    bool operator<(const Contact& other) const;
    bool operator==(const Contact& other) const;

private:
    QString m_firstName;
    QString m_lastName;
```

```
    QString m_middleName;
    QString m_address;
    QDate m_birthDate;
    QString m_email;
    QStringList m_phoneNumbers;
};
```

contactstorage.h

```
#ifndef CONTACTSTORAGE_H
#define CONTACTSTORAGE_H

#include "contact.h"
#include <QList>
#include <QString>

class ContactStorage
{
public:
    ContactStorage(const QString& filename = "phonebook.json");

    bool load();
    bool save();

    QList<Contact>& contacts() { return m_contacts; }
    const QList<Contact>& contacts() const { return m_contacts; }

    void addContact(const Contact& contact);
    void removeContact(int index);
    void updateContact(int index, const Contact& contact);

    QString filename() const { return m_filename; }
    void setFilename(const QString& filename) { m_filename =
   filename; }

private:
    QList<Contact> m_contacts;
    QString m_filename;
};

#endif // CONTACTSTORAGE_H
```

contactvalidator.h

```
#ifndef CONTACTVALIDATOR_H
#define CONTACTVALIDATOR_H

#include <QString>
#include <QDate>
```

```
#include <QStringList>

class ContactValidator
{
public:
    static bool validateName(const QString& name, QString&
   errorMessage);
    static bool validatePhone(const QString& phone, QString&
   errorMessage);
    static bool validateBirthDate(const QDate& date, QString&
   errorMessage);
    static bool validateEmail(const QString& email, QString&
   errorMessage);

    // Normalize functions
    static QString normalizeName(const QString& name);
    static QString normalizePhone(const QString& phone);
    static QString normalizeEmail(const QString& email);

private:
    static QString extractPhoneDigits(const QString& phone);
};

#endif // CONTACTVALIDATOR_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTableWidget>
#include <QPushButton>
#include <QLineEdit>
#include <QDateEdit>
#include <QTextEdit>
#include <QComboBox>
#include <QLabel>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QGroupBox>
#include <QMessageBox>
#include <QHeaderView>
#include <QStack>
#include "contactstorage.h"
#include "contact.h"

// Структура для хранения действия
struct ContactAction {
    enum Type { Add, Edit, Delete };
    Type type;
```

```
25      Contact contact;
26      int index; // для Edit и Delete
27  };
28
29  class MainWindow : public QMainWindow
30  {
31      Q_OBJECT
32
33  public:
34      MainWindow(QWidget *parent = nullptr);
35      ~MainWindow();
36
37  private slots:
38      void addContact();
39      void editContact();
40      void deleteContact();
41      void searchContacts();
42      void clearSearch();
43      void onTableSelectionChanged();
44      void onTableSortChanged(int column);
45      void loadContacts();
46      void saveContacts();
47      void undoLastAction();
48
49  private:
50      void setupUI();
51      void setupTable();
52      void setupForm();
53      void setupSearch();
54      void populateTable();
55      void clearForm();
56      void fillForm(const Contact& contact);
57      Contact getContactFromForm() const;
58      bool validateForm(QString& errorMessage);
59      void showError(const QString& message);
60      void showInfo(const QString& message);
61      int getSelectedRow() const;
62      bool checkForDuplicates(const Contact& contact, int
    excludeIndex = -1);
63      void pushAction(ContactAction::Type type, const Contact&
    contact, int index = -1);
64      QString getFormattingWarnings(const QString& firstName, const
    QString& lastName,
65                                    const QString& middleName,
    const QString& phone);
66
67      // UI Components
68      QWidget* m_centralWidget;
69      QVBoxLayout* m_mainLayout;
70
71      // Table
72      QTableWidget* m_table;
73
```

```
    // Form
    QGroupBox* m_formGroup;
    QLineEdit* m_firstNameEdit;
    QLineEdit* m_lastNameEdit;
    QLineEdit* m_middleNameEdit;
    QTextEdit* m_addressEdit;
    QDateEdit* m_birthDateEdit;
    QLineEdit* m_emailEdit;
    QTextEdit* m_phoneNumbersEdit;

    // Buttons
    QPushButton* m_addButton;
    QPushButton* m_editButton;
    QPushButton* m_deleteButton;
    QPushButton* m_saveButton;
    QPushButton* m_cancelButton;
    QPushButton* m_undoButton;

    // Search
    QGroupBox* m_searchGroup;
    QLineEdit* m_searchEdit;
    QComboBox* m_searchFieldCombo;
    QPushButton* m_searchButton;
    QPushButton* m_clearSearchButton;
    QDateEdit* m_searchDateEdit;
  QComboBox* m_dateSearchTypeCombo;

    // Data
    ContactStorage* m_storage;
    int m_editingIndex;
    bool m_isEditing;
    QStack<ContactAction> m_undoStack;
};

#endif // MAINWINDOW_H
```

contact.cpp

```
#include "contact.h"
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonValue>

Contact::Contact()
    : m_birthDate(QDate::currentDate())
{
}

Contact::Contact(const QString& firstName, const QString&
   lastName, const QString& middleName,
```

```
                  const QString& address, const QDate& birthDate,
    const QString& email,
                  const QStringList& phoneNumbers)
    : m_firstName(firstName)
    , m_lastName(lastName)
    , m_middleName(middleName)
    , m_address(address)
    , m_birthDate(birthDate)
    , m_email(email)
    , m_phoneNumbers(phoneNumbers)
{
}

QJsonObject Contact::toJson() const
{
    QJsonObject json;
    json["firstName"] = m_firstName;
    json["lastName"] = m_lastName;
    json["middleName"] = m_middleName;
    json["address"] = m_address;
    json["birthDate"] = m_birthDate.toString(Qt::ISODate);
    json["email"] = m_email;

    QJsonArray phoneArray;
    for (const QString& phone : m_phoneNumbers) {
        phoneArray.append(phone);
    }
    json["phoneNumbers"] = phoneArray;

    return json;
}

Contact Contact::fromJson(const QJsonObject& json)
{
    Contact contact;
    contact.m_firstName = json["firstName"].toString();
    contact.m_lastName = json["lastName"].toString();
    contact.m_middleName = json["middleName"].toString();
    contact.m_address = json["address"].toString();
    contact.m_birthDate =
    QDate::fromString(json["birthDate"].toString(), Qt::ISODate);
    contact.m_email = json["email"].toString();

    QJsonArray phoneArray = json["phoneNumbers"].toArray();
    for (const QJsonValue& value : phoneArray) {
        contact.m_phoneNumbers.append(value.toString());
    }

    return contact;
}

bool Contact::operator<(const Contact& other) const
{
```

```
63      if (m_lastName != other.m_lastName) {
64          return m_lastName < other.m_lastName;
65      }
66      if (m_firstName != other.m_firstName) {
67          return m_firstName < other.m_firstName;
68      }
69      return m_middleName < other.m_middleName;
70  }
71
72  bool Contact::operator==(const Contact& other) const
73  {
74      return m_firstName == other.m_firstName &&
75             m_lastName == other.m_lastName &&
76             m_middleName == other.m_middleName &&
77             m_address == other.m_address &&
78             m_birthDate == other.m_birthDate &&
79             m_email == other.m_email &&
80             m_phoneNumbers == other.m_phoneNumbers;
81  }
```

contactstorage.cpp

```
1   #include "contact.h"
2   #include <QJsonObject>
3   #include <QJsonArray>
4   #include <QJsonValue>
5
6   Contact::Contact()
7       : m_birthDate(QDate::currentDate())
8   {
9   }
10
11  Contact::Contact(const QString& firstName, const QString&
        lastName, const QString& middleName,
12                   const QString& address, const QDate& birthDate,
        const QString& email,
13                   const QStringList& phoneNumbers)
14      : m_firstName(firstName)
15      , m_lastName(lastName)
16      , m_middleName(middleName)
17      , m_address(address)
18      , m_birthDate(birthDate)
19      , m_email(email)
20      , m_phoneNumbers(phoneNumbers)
21  {
22  }
23
24  QJsonObject Contact::toJson() const
25  {
26      QJsonObject json;
27      json["firstName"] = m_firstName;
```

```
    json["lastName"] = m_lastName;
    json["middleName"] = m_middleName;
    json["address"] = m_address;
    json["birthDate"] = m_birthDate.toString(Qt::ISODate);
    json["email"] = m_email;

    QJsonArray phoneArray;
    for (const QString& phone : m_phoneNumbers) {
        phoneArray.append(phone);
    }
    json["phoneNumbers"] = phoneArray;

    return json;
}

Contact Contact::fromJson(const QJsonObject& json)
{
    Contact contact;
    contact.m_firstName = json["firstName"].toString();
    contact.m_lastName = json["lastName"].toString();
    contact.m_middleName = json["middleName"].toString();
    contact.m_address = json["address"].toString();
    contact.m_birthDate =
   QDate::fromString(json["birthDate"].toString(), Qt::ISODate);
    contact.m_email = json["email"].toString();

    QJsonArray phoneArray = json["phoneNumbers"].toArray();
    for (const QJsonValue& value : phoneArray) {
        contact.m_phoneNumbers.append(value.toString());
    }

    return contact;
}

bool Contact::operator<(const Contact& other) const
{
    if (m_lastName != other.m_lastName) {
        return m_lastName < other.m_lastName;
    }
    if (m_firstName != other.m_firstName) {
        return m_firstName < other.m_firstName;
    }
    return m_middleName < other.m_middleName;
}

bool Contact::operator==(const Contact& other) const
{
    return m_firstName == other.m_firstName &&
           m_lastName == other.m_lastName &&
           m_middleName == other.m_middleName &&
           m_address == other.m_address &&
           m_birthDate == other.m_birthDate &&
           m_email == other.m_email &&
```

```
            m_phoneNumbers == other.m_phoneNumbers;
}
```

contactvalidator.cpp

```
#include "contactvalidator.h"
#include <QRegularExpression>
#include <QDate>

// проверка имени
bool ContactValidator::validateName(const QString& name, QString&
    errorMessage)
{
    // нормализуем
    QString normalized = normalizeName(name);

    if (normalized.isEmpty()) {
        errorMessage = "Имя не может быть пустым";
        return false;
    }

    // работаем с исходным вводом обрезаем( пробелы)
    QString trimmed = name.trimmed();

    // не должно начинаться или заканчиваться на дефис
    if (trimmed.startsWith('-') || trimmed.endsWith('-')) {
        errorMessage = "Имя не может начинаться или заканчиваться
    на дефис";
        return false;
    }

    // разрешенные символы: буквы, цифры, дефис, пробел
    QRegularExpression
    regex(R"(^[A-Za-AzЯаяЁё--0-9][A-Za-AzЯаяЁё--0-9\-
    ]*[A-Za-AzЯаяЁё--0-9]$)");
    if (!regex.match(trimmed).hasMatch()) {
        errorMessage = "Имя может содержать только буквы, цифры,
    дефис и пробел";
        return false;
    }

    // запрет двойных дефисов и множественных пробелов
    if (trimmed.contains("--") ||
    trimmed.contains(QRegularExpression(R"(\s{2,})"))) {
        errorMessage = "Имя не должно содержать двойные дефисы или
    множественные пробелы";
        return false;
    }

    return true;
}
```

```
40
41 // проверка телефона
42 bool ContactValidator::validatePhone(const QString& phone,
       QString& errorMessage)
43 {
44     QString normalized = normalizePhone(phone);
45
46     if (normalized.isEmpty()) {
47         errorMessage = "Телефон не может быть пустым";
48         return false;
49     }
50
51     // оставляем только цифры для проверки длины
52     QString digits = extractPhoneDigits(normalized);
53
54     if (digits.length() < 10) {
55         errorMessage = "Телефон должен содержать минимум 10 цифр";
56         return false;
57     }
58     if (digits.length() > 15) {
59         errorMessage = "Телефон должен содержать максимум 15 цифр";
60         return false;
61     }
62
63     // формат: должен быть + и цифры
64     QRegularExpression formatRegex(R"(^\+\d+$)");
65     if (!formatRegex.match(normalized).hasMatch()) {
66         errorMessage = "Телефон должен быть в международном
       формате (+...)";
67         return false;
68     }
69
70     return true;
71 }
72
73 // проверка даты рождения
74 bool ContactValidator::validateBirthDate(const QDate& date,
       QString& errorMessage)
75 {
76     if (!date.isValid()) {
77         errorMessage = "Неверная дата";
78         return false;
79     }
80
81     QDate currentDate = QDate::currentDate();
82
83     // дата должна быть в прошлом
84     if (date >= currentDate) {
85         errorMessage = "Дата рождения должна быть меньше текущей
       даты";
86         return false;
87     }
88
```

```
    // не старше 150 лет
    QDate minDate = currentDate.addYears(-150);
    if (date < minDate) {
        errorMessage = "Дата рождения слишком давняя";
        return false;
    }

    // тут можно было бы проверять днивисокосные/ года, но qt уже валидирует дату
    return true;
}

// проверка email
bool ContactValidator::validateEmail(const QString& email, QString& errorMessage)
{
    QString normalized = normalizeEmail(email);

    if (normalized.isEmpty()) {
        errorMessage = "Email не может быть пустым";
        return false;
    }

    // должно содержать @ ровно один
    if (!normalized.contains('@')) {
        errorMessage = "Email должен содержать символ @";
        return false;
    }

    QStringList parts = normalized.split('@');
    if (parts.size() != 2) {
        errorMessage = "Email должен содержать ровно один символ @";
        return false;
    }

    QString username = parts[0];
    QString domain = parts[1];

    if (username.isEmpty()) {
        errorMessage = "Имя пользователя не может быть пустым";
        return false;
    }
    if (domain.isEmpty()) {
        errorMessage = "Домен не может быть пустым";
        return false;
    }

    // имя пользователя: латиница, цифры, точки, подчеркивания
    if (username.startsWith('.') || username.endsWith('.')) {
        errorMessage = "Имя пользователя не может начинаться или заканчиваться на точку";
        return false;
```

```
138     }
139     QRegularExpression usernameRegex(R"(^[A-Za-z0-9._%+\-]+$)");
140     if (!usernameRegex.match(username).hasMatch()) {
141         errorMessage = "Имя пользователя должно состоять из
    латинских букв, цифр, точек и подчеркиваний";
142         return false;
143     }
144 
145     // домен должен содержать точку
146     if (!domain.contains('.')) {
147         errorMessage = "Домен должен содержать минимум одну точку
    например(, example.com)";
148         return false;
149     }
150 
151     // проверка формата домена
152     QRegularExpression
    domainRegex(R"(^[A-Za-z0-9][A-Za-z0-9\-]*(\.[A-Za-z0-9][A-Za-z0-9\-]*)*\
153     if (!domainRegex.match(domain).hasMatch()) {
154         errorMessage = "Неверный формат домена";
155         return false;
156     }
157 
158     return true;
159 }
160 
161 // нормализация имени
162 QString ContactValidator::normalizeName(const QString& name)
163 {
164     QString normalized = name.trimmed();
165 
166     if (normalized.isEmpty()) {
167         return normalized;
168     }
169 
170     // убираем лишние пробелы
171     normalized.replace(QRegularExpression(R"(\s{2,})"), " ");
172     // убираем повторные дефисы
173     normalized.replace(QRegularExpression(R"(-{2,})"), "-");
174 
175     // первая буква заглавная
176     normalized[0] = normalized[0].toUpper();
177 
178     // делаем буквы после пробеладефиса/ заглавными, остальные
    строчными
179     for (int i = 1; i < normalized.length(); ++i) {
180         QChar current = normalized[i];
181         QChar previous = normalized[i-1];
182         if ((previous == ' ' || previous == '-') &&
    current.isLetter()) {
183             normalized[i] = current.toUpper();
184         } else if (current.isLetter() && previous != ' ' &&
    previous != '-') {
```

```
185             normalized[i] = current.toLower();
186         }
187     }
188 
189     return normalized;
190 }
191 
192 // нормализация телефона
193 QString ContactValidator::normalizePhone(const QString& phone)
194 {
195     QString normalized = phone.trimmed();
196 
197     // удаляем пробелы, скобки, дефисы
198     normalized.remove(QRegularExpression(R"([\s\(\)\-])"));
199 
200     // русский 8->+7
201     if (normalized.startsWith("8")) {
202         QString digits = extractPhoneDigits(normalized);
203         if (digits.length() == 11) {
204             normalized = "+7" + digits.mid(1);
205         }
206     }
207     // если начинается на 7 без +, добавляем +
208     else if (normalized.startsWith("7") &&
    !normalized.startsWith("+")) {
209         normalized = "+" + normalized;
210     }
211     // если нет +, добавляем
212     else if (!normalized.startsWith("+")) {
213         normalized = "+" + normalized;
214     }
215 
216     return normalized;
217 }
218 
219 // нормализация email
220 QString ContactValidator::normalizeEmail(const QString& email)
221 {
222     QString normalized = email.trimmed().toLower();
223 
224     // убираем пробелы вокруг @ и все пробелы
225     normalized.replace(QRegularExpression(R"(\s*@\s*)"), "@");
226     normalized.remove(' ');
227 
228     return normalized;
229 }
230 
231 // извлекаем только цифры
232 QString ContactValidator::extractPhoneDigits(const QString& phone)
233 {
234     QString digits;
235     for (const QChar& ch : phone) {
236         if (ch.isDigit()) {
```

```
            digits.append(ch);
        }
    }
    return digits;
}
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow window;
    window.show();

    return app.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "contactvalidator.h"
#include <QApplication>

// запускается при создании программы
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), m_editingIndex(-1), m_isEditing(false)
{
    m_storage = new ContactStorage("phonebook.json");
    m_storage->load();
    setupUI();       // делаем интерфейс
    populateTable(); // заполняем таблицу
    connect(qApp, &QApplication::aboutToQuit, this,
   &MainWindow::saveContacts); // сохраняем перед выходом
}

// деструктор
MainWindow::~MainWindow(){
    saveContacts(); // сохраняем
    delete m_storage; // чистим память
}

// собираем ui
void MainWindow::setupUI(){
    m_centralWidget = new QWidget(this);
    setCentralWidget(m_centralWidget);
```

```
    m_mainLayout = new QVBoxLayout(m_centralWidget);

    setupTable(); // таблица
    setupForm();  // форма ввода
    setupSearch(); // поиск

    setWindowTitle("Телефонный справочник");
    resize(1000, 700);
}

// создаем таблицу
void MainWindow::setupTable(){
    m_table = new QTableWidget(this);
    m_table->setColumnCount(7);
    m_table->setHorizontalHeaderLabels({"Фамилия", "Имя",
   "Отчество", "Адрес", "Дата рождения", "Email", "Телефоны"});
    m_table->setSelectionBehavior(QAbstractItemView::SelectRows);
    m_table->setSelectionMode(QAbstractItemView::SingleSelection);
    m_table->setSortingEnabled(true); // включаем сортировку
    m_table->setEditTriggers(QAbstractItemView::NoEditTriggers);
    m_table->horizontalHeader()->setStretchLastSection(true);
    m_table->setAlternatingRowColors(true);

    // сигналы выбора и сортировки
    connect(m_table, &QTableWidget::itemSelectionChanged, this,
   &MainWindow::onTableSelectionChanged);
    connect(m_table->horizontalHeader(),
   &QHeaderView::sortIndicatorChanged, this,
   &MainWindow::onTableSortChanged);
    m_mainLayout->addWidget(m_table);
}

// создаем форму ввода
void MainWindow::setupForm(){
    m_formGroup = new QGroupBox("Данные контакта", this);
    QVBoxLayout* formLayout = new QVBoxLayout(m_formGroup);

    // фио
    QHBoxLayout* nameLayout = new QHBoxLayout();
    m_lastNameEdit = new QLineEdit(this);
    m_firstNameEdit = new QLineEdit(this);
    m_middleNameEdit = new QLineEdit(this);
    nameLayout->addWidget(new QLabel("Фамилия:", this));
    nameLayout->addWidget(m_lastNameEdit);
    nameLayout->addWidget(new QLabel("Имя:", this));
    nameLayout->addWidget(m_firstNameEdit);
    nameLayout->addWidget(new QLabel("Отчество:", this));
    nameLayout->addWidget(m_middleNameEdit);
    formLayout->addLayout(nameLayout);

    // адрес
    QHBoxLayout* addressLayout = new QHBoxLayout();
    m_addressEdit = new QTextEdit(this);
```

```
    m_addressEdit->setMaximumHeight(60);
    addressLayout->addWidget(new QLabel("Адрес:", this));
    addressLayout->addWidget(m_addressEdit);
    formLayout->addLayout(addressLayout);

    // дата рождения
    QHBoxLayout* dateLayout = new QHBoxLayout();
    m_birthDateEdit = new QDateEdit(this);
    m_birthDateEdit->setCalendarPopup(true);
    m_birthDateEdit->setDate(QDate::currentDate().addYears(-20));

m_birthDateEdit->setMaximumDate(QDate::currentDate().addDays(-1));
    dateLayout->addWidget(new QLabel("Дата рождения:", this));
    dateLayout->addWidget(m_birthDateEdit);
    formLayout->addLayout(dateLayout);

    // email
    QHBoxLayout* emailLayout = new QHBoxLayout();
    m_emailEdit = new QLineEdit(this);
    emailLayout->addWidget(new QLabel("Email:", this));
    emailLayout->addWidget(m_emailEdit);
    formLayout->addLayout(emailLayout);

    // телефоны
    QHBoxLayout* phoneLayout = new QHBoxLayout();
    m_phoneNumbersEdit = new QTextEdit(this);
    m_phoneNumbersEdit->setMaximumHeight(60);
    phoneLayout->addWidget(new QLabel("Телефоны:", this));
    phoneLayout->addWidget(m_phoneNumbersEdit);
    formLayout->addLayout(phoneLayout);

    // кнопки
    QHBoxLayout* buttonLayout = new QHBoxLayout();
    m_addButton = new QPushButton("Добавить", this);
    m_editButton = new QPushButton("Редактировать", this);
    m_deleteButton = new QPushButton("Удалить", this);
    m_saveButton = new QPushButton("Сохранить", this);
    m_cancelButton = new QPushButton("Отмена", this);
    m_undoButton = new QPushButton("Отменить последнее действие",
this);

    // изначальные состояния кнопок
    m_editButton->setEnabled(false);
    m_deleteButton->setEnabled(false);
    m_saveButton->setEnabled(false);
    m_cancelButton->setEnabled(false);
    m_undoButton->setEnabled(false);

    buttonLayout->addWidget(m_addButton);
    buttonLayout->addWidget(m_editButton);
    buttonLayout->addWidget(m_deleteButton);
    buttonLayout->addWidget(m_saveButton);
    buttonLayout->addWidget(m_cancelButton);
```

```
126     buttonLayout->addWidget(m_undoButton);
127     buttonLayout->addStretch();
128 
129     formLayout->addLayout(buttonLayout);
130     m_mainLayout->addWidget(m_formGroup);
131 
132     // подключаем кнопки
133     connect(m_addButton, &QPushButton::clicked, this,
    &MainWindow::addContact);
134     connect(m_editButton, &QPushButton::clicked, this,
    &MainWindow::editContact);
135     connect(m_deleteButton, &QPushButton::clicked, this,
    &MainWindow::deleteContact);
136     connect(m_saveButton, &QPushButton::clicked, this,
    &MainWindow::addContact);
137     connect(m_undoButton, &QPushButton::clicked, this,
    &MainWindow::undoLastAction);
138     connect(m_cancelButton, &QPushButton::clicked, this, [this]() {
139         clearForm();
140         m_isEditing = false;
141         m_editingIndex = -1;
142         m_saveButton->setEnabled(false);
143         m_cancelButton->setEnabled(false);
144         m_addButton->setEnabled(true);
145     });
146 }
147 
148 // создаем поиск
149 void MainWindow::setupSearch(){
150     m_searchGroup = new QGroupBox("Поиск", this);
151     QHBoxLayout* searchLayout = new QHBoxLayout(m_searchGroup);
152 
153     m_searchEdit = new QLineEdit(this);
154     m_searchFieldCombo = new QComboBox(this);
155     m_searchFieldCombo->addItems({"Все поля", "Фамилия", "Имя",
    "Отчество", "Адрес", "Email", "Телефон", "Дата рождения"});
156     m_searchButton = new QPushButton("Найти", this);
157     m_clearSearchButton = new QPushButton("Очистить", this);
158 
159     // Поле для поиска по дате
160   m_searchDateEdit = new QDateEdit(this);
161   m_searchDateEdit->setCalendarPopup(true);
162   m_searchDateEdit->setDate(QDate::currentDate());
163   m_searchDateEdit->setVisible(false);
164 
165   // Тип поиска по дате
166   m_dateSearchTypeCombo = new QComboBox(this);
167   m_dateSearchTypeCombo->addItems({"Точная дата", "Год", "Месяц и
    год"});
168   m_dateSearchTypeCombo->setVisible(false);
169 
170   searchLayout->addWidget(new QLabel("Поиск:", this));
171   searchLayout->addWidget(m_searchEdit);
```

```
172   searchLayout->addWidget(m_searchDateEdit);
173   searchLayout->addWidget(m_dateSearchTypeCombo);
174   searchLayout->addWidget(new QLabel("Поле:", this));
175   searchLayout->addWidget(m_searchFieldCombo);
176   searchLayout->addWidget(m_searchButton);
177   searchLayout->addWidget(m_clearSearchButton);
178   m_mainLayout->addWidget(m_searchGroup);
179
180   // Переключение между текстовым полем и датой
181   connect(m_searchFieldCombo,
       QOverload<int>::of(&QComboBox::currentIndexChanged), this,
       [this](int index) {
182     bool isDateSearch = (index == 7); // Дата" рождения"
183     m_searchEdit->setVisible(!isDateSearch);
184     m_searchDateEdit->setVisible(isDateSearch);
185     m_dateSearchTypeCombo->setVisible(isDateSearch);
186   });
187
188     // подключаем поиск
189     connect(m_searchButton, &QPushButton::clicked, this,
       &MainWindow::searchContacts);
190     connect(m_clearSearchButton, &QPushButton::clicked, this,
       &MainWindow::clearSearch);
191     connect(m_searchEdit, &QLineEdit::returnPressed, this,
       &MainWindow::searchContacts);
192 }
193
194 // проверка на дубликаты
195 bool MainWindow::checkForDuplicates(const Contact& contact, int
       excludeIndex) {
196     const QList<Contact>& contacts = m_storage->contacts();
197     for (int i = 0; i < contacts.size(); ++i) {
198         if (i == excludeIndex) continue; // пропускаем сам контакт
       при редактировании
199
200         const Contact& existing = contacts[i];
201         if (existing.firstName() == contact.firstName() &&
202             existing.lastName() == contact.lastName() &&
203             existing.middleName() == contact.middleName() &&
204       existing.phoneNumbers() == contact.phoneNumbers() &&
205             existing.email() == contact.email()) {
206             return true; // дубликат найден
207         }
208     }
209     return false;
210 }
211
212 // проверка предупреждений о форматировании
213 QString MainWindow::getFormattingWarnings(const QString&
       firstName, const QString& lastName,
214                                           const QString&
       middleName, const QString& phone) {
215     QStringList warnings;
```

```
216
217     // Проверка имён на множественные заглавные буквы
218     auto checkMultipleUppercase = [](const QString& name, const
    QString& fieldName) -> QString {
219         if (name.isEmpty()) return "";
220         int uppercaseCount = 0;
221         for (const QChar& ch : name) {
222             if (ch.isUpper()) uppercaseCount++;
223         }
224         if (uppercaseCount > 1) {
225             return fieldName + ": обнаружено несколько заглавных
    букв, произведем изменение на первая( заглавная, остальные
    маленькие)";
226         }
227         return "";
228     };
229
230     QString firstWarning = checkMultipleUppercase(firstName,
    "Имя");
231     QString lastWarning = checkMultipleUppercase(lastName,
    "Фамилия");
232     QString middleWarning = checkMultipleUppercase(middleName,
    "Отчество");
233
234     if (!firstWarning.isEmpty()) warnings.append(firstWarning);
235     if (!lastWarning.isEmpty()) warnings.append(lastWarning);
236     if (!middleWarning.isEmpty()) warnings.append(middleWarning);
237
238     // Проверка телефона на неправильный формат
239     QString phoneText = m_phoneNumbersEdit->toPlainText();
240     QStringList phoneLines = phoneText.split('\n',
    Qt::SkipEmptyParts);
241     for (const QString& phoneLine : phoneLines) {
242         QString trimmed = phoneLine.trimmed();
243         QString normalized =
    ContactValidator::normalizePhone(trimmed);
244         if (trimmed != normalized && !trimmed.isEmpty()) {
245             warnings.append("Телефон будет преобразован: " +
    trimmed + " => " + normalized);
246         }
247     }
248
249     return warnings.join("\n\n");
250 }
251
252 // сохранение действия для отмены
253 void MainWindow::pushAction(ContactAction::Type type, const
    Contact& contact, int index) {
254     ContactAction action;
255     action.type = type;
256     action.contact = contact;
257     action.index = index;
258     m_undoStack.push(action);
```

```
259     m_undoButton->setEnabled(true);
260 }
261 
262 // отмена последнего действия
263 void MainWindow::undoLastAction() {
264   ContactAction action = m_undoStack.pop();
265 
266     switch (action.type) { // проверяем события
267     case ContactAction::Add:
268         if (!m_storage->contacts().isEmpty()) {
269             m_storage->removeContact(m_storage->contacts().size()
    - 1);
270             showInfo("Добавление контакта отменено");
271         }
272         break;
273 
274     case ContactAction::Edit:
275         if (action.index >= 0 && action.index <
    m_storage->contacts().size()) {
276             m_storage->updateContact(action.index, action.contact);
277             showInfo("Редактирование контакта отменено");
278         }
279         break;
280 
281     case ContactAction::Delete:
282         if (action.index >= 0) {
283       m_storage->contacts().insert(action.index, action.contact);
284             showInfo("Удаление контакта отменено");
285         }
286         break;
287     }
288     populateTable();
289     saveContacts();
290 
291     if (m_undoStack.isEmpty()) {
292         m_undoButton->setEnabled(false);
293     }
294 }
295 
296 // заполняем таблицу контактами
297 void MainWindow::populateTable(){
298     m_table->setSortingEnabled(false);
299     m_table->setRowCount(0);
300 
301     const QList<Contact>& contacts = m_storage->contacts();
302     for (int i = 0; i < contacts.size(); ++i) {
303         const Contact& contact = contacts[i];
304         int row = m_table->rowCount();
305         m_table->insertRow(row);
306 
307         m_table->setItem(row, 0, new
    QTableWidgetItem(contact.lastName()));
```

```
308         m_table->setItem(row, 1, new
    QTableWidgetItem(contact.firstName()));
309         m_table->setItem(row, 2, new
    QTableWidgetItem(contact.middleName()));
310         m_table->setItem(row, 3, new
    QTableWidgetItem(contact.address()));
311         m_table->setItem(row, 4, new
    QTableWidgetItem(contact.birthDate().toString("dd.MM.yyyy")));
312         m_table->setItem(row, 5, new
    QTableWidgetItem(contact.email()));
313         m_table->setItem(row, 6, new
    QTableWidgetItem(contact.phoneNumbers().join(", ")));
314     }
315
316     m_table->setSortingEnabled(true);
317     m_table->resizeColumnsToContents();
318 }
319
320 // очищаем форму
321 void MainWindow::clearForm(){
322     m_firstNameEdit->clear();
323     m_lastNameEdit->clear();
324     m_middleNameEdit->clear();
325     m_addressEdit->clear();
326     m_birthDateEdit->setDate(QDate::currentDate().addYears(-20));
327     m_emailEdit->clear();
328     m_phoneNumbersEdit->clear();
329 }
330
331 // заполняем форму данными контакта
332 void MainWindow::fillForm(const Contact& contact){
333     m_firstNameEdit->setText(contact.firstName());
334     m_lastNameEdit->setText(contact.lastName());
335     m_middleNameEdit->setText(contact.middleName());
336     m_addressEdit->setPlainText(contact.address());
337     m_birthDateEdit->setDate(contact.birthDate());
338     m_emailEdit->setText(contact.email());
339
    m_phoneNumbersEdit->setPlainText(contact.phoneNumbers().join("\n"));
340 }
341
342 // собираем контакт из формы
343 Contact MainWindow::getContactFromForm() const{
344     QString firstName =
    ContactValidator::normalizeName(m_firstNameEdit->text());
345     QString lastName =
    ContactValidator::normalizeName(m_lastNameEdit->text());
346     QString middleName =
    ContactValidator::normalizeName(m_middleNameEdit->text());
347     QString address = m_addressEdit->toPlainText().trimmed();
348     QDate birthDate = m_birthDateEdit->date();
349     QString email =
    ContactValidator::normalizeEmail(m_emailEdit->text());
```

```
350
351     QStringList phoneNumbers;
352     QString phoneText = m_phoneNumbersEdit->toPlainText();
353     QStringList phoneLines = phoneText.split('\n',
    Qt::SkipEmptyParts);
354     for (const QString& phone : phoneLines) {
355         QString normalized =
    ContactValidator::normalizePhone(phone.trimmed());
356         if (!normalized.isEmpty()) phoneNumbers.append(normalized);
357     }
358
359     return Contact(firstName, lastName, middleName, address,
    birthDate, email, phoneNumbers);
360 }
361
362 // проверяем форму
363 bool MainWindow::validateForm(QString& errorMessage){
364     QString firstName =
    ContactValidator::normalizeName(m_firstNameEdit->text());
365     QString lastName =
    ContactValidator::normalizeName(m_lastNameEdit->text());
366     QString middleName =
    ContactValidator::normalizeName(m_middleNameEdit->text());
367     QString email =
    ContactValidator::normalizeEmail(m_emailEdit->text());
368     QDate birthDate = m_birthDateEdit->date();
369     QString msg;
370
371     if (!firstName.isEmpty() &&
    !ContactValidator::validateName(firstName, msg)) {
372         errorMessage = "Имя: " + msg;
373         return false;
374     }
375     if (!lastName.isEmpty() &&
    !ContactValidator::validateName(lastName, msg)) {
376         errorMessage = "Фамилия: " + msg;
377         return false;
378     }
379     if (!middleName.isEmpty() &&
    !ContactValidator::validateName(middleName, msg)) {
380         errorMessage = "Отчество: " + msg;
381         return false;
382     }
383     if (!ContactValidator::validateBirthDate(birthDate, msg)) {
384         errorMessage = "Дата рождения: " + msg;
385         return false;
386     }
387     if (!ContactValidator::validateEmail(email, msg)) {
388         errorMessage = "Email: " + msg;
389         return false;
390     }
391
392     QString phoneText = m_phoneNumbersEdit->toPlainText();
```

```
393     QStringList phoneLines = phoneText.split('\n',
    Qt::SkipEmptyParts);
394     if (phoneLines.isEmpty()) {
395         errorMessage = "Необходимо указать хотя бы один телефон";
396         return false;
397     }
398 
399     for (const QString& phone : phoneLines) {
400         QString normalizedPhone =
    ContactValidator::normalizePhone(phone.trimmed());
401         if (!ContactValidator::validatePhone(normalizedPhone,
    msg)) {
402             errorMessage = "Телефон: " + msg;
403             return false;
404         }
405     }
406     return true;
407 }
408 
409 // окно с ошибкой
410 void MainWindow::showError(const QString& message){
411     QMessageBox::critical(this, "Ошибка", message);
412 }
413 
414 // окно с инфой
415 void MainWindow::showInfo(const QString& message){
416     QMessageBox::information(this, "Информация", message);
417 }
418 
419 // получить выбранную строку
420 int MainWindow::getSelectedRow() const{
421     QList<QTableWidgetItem*> selected = m_table->selectedItems();
422     return selected.isEmpty() ? -1 : selected.first()->row();
423 }
424 
425 // добавляем или обновляем контакт
426 void MainWindow::addContact(){
427     QString errorMessage;
428     if (!validateForm(errorMessage)) {
429         showError(errorMessage);
430         return;
431     }
432 
433     // Проверяем предупреждения о форматировании
434     QString warnings =
    getFormattingWarnings(m_firstNameEdit->text(),
    m_lastNameEdit->text(), m_middleNameEdit->text(),
    m_phoneNumbersEdit->toPlainText());
435 
436     if (!warnings.isEmpty()) {
437         QMessageBox::StandardButton reply = QMessageBox::question(
438             this, "Предупреждение о форматировании",
439             warnings + "\n\Продолжитьn?",
```

```
            QMessageBox::Yes | QMessageBox::No
        );

        if (reply != QMessageBox::Yes) {
            return;
        }
    }

    Contact contact = getContactFromForm();

    // Проверка на дубликаты
    if (checkForDuplicates(contact, m_isEditing ? m_editingIndex :
-1)) {
        showError("Контакт с такими данными ФИО( и Email) уже
существует!");
        return;
    }

    if (m_isEditing && m_editingIndex >= 0) {
        // Сохраняем старый контакт для отмены
        Contact oldContact = m_storage->contacts()[m_editingIndex];
        pushAction(ContactAction::Edit, oldContact,
m_editingIndex);

        m_storage->updateContact(m_editingIndex, contact);
        showInfo("Контакт обновлен");
    } else {
        m_storage->addContact(contact);
        pushAction(ContactAction::Add, contact,
m_storage->contacts().size() - 1);
        showInfo("Контакт добавлен");
    }

    clearForm();
    populateTable();
    saveContacts();

    m_isEditing = false;
    m_editingIndex = -1;
    m_saveButton->setEnabled(false);
    m_cancelButton->setEnabled(false);
    m_addButton->setEnabled(true);
}

// переходим в режим редактирования
void чMainWindow::editContact(){
    int row = getSelectedRow();
    if (row < 0) {
        showError("Выберите контакт для редактирования");
        return;
    }

    QString lastName = m_table->item(row, 0)->text();
```

```
    QString firstName = m_table->item(row, 1)->text();
    QString email = m_table->item(row, 5)->text();

    int index = -1;
    const QList<Contact>& contacts = m_storage->contacts();
    for (int i = 0; i < contacts.size(); ++i) {
        if (contacts[i].lastName() == lastName &&
            contacts[i].firstName() == firstName &&
            contacts[i].email() == email) {
            index = i;
            break;
        }
    }

    if (index < 0) {
        showError("Контакт не найден");
        return;
    }

    m_editingIndex = index;
    m_isEditing = true;
    fillForm(contacts[index]);

    m_addButton->setEnabled(false);
    m_editButton->setEnabled(false);
    m_deleteButton->setEnabled(false);
    m_saveButton->setEnabled(true);
    m_cancelButton->setEnabled(true);
}

// удаляем контакт
void MainWindow::deleteContact(){
    int row = getSelectedRow();
    if (row < 0) {
        showError("Выберите контакт для удаления");
        return;
    }

    QMessageBox::StandardButton reply = QMessageBox::question(
        this, "Подтверждение", "Вы уверены, что хотите удалить
   этот контакт?",
        QMessageBox::Yes | QMessageBox::No
    );

    if (reply == QMessageBox::Yes) {
        QString lastName = m_table->item(row, 0)->text();
        QString firstName = m_table->item(row, 1)->text();
        QString email = m_table->item(row, 5)->text();

        int index = -1;
        const QList<Contact>& contacts = m_storage->contacts();
        for (int i = 0; i < contacts.size(); ++i) {
            if (contacts[i].lastName() == lastName &&
```

```
                contacts[i].firstName() == firstName &&
                contacts[i].email() == email) {
                index = i;
                break;
            }
        }

        if (index >= 0) {
            // Сохраняем контакт для отмены
            Contact deletedContact = contacts[index];
            pushAction(ContactAction::Delete, deletedContact,
   index);

            m_storage->removeContact(index);
            populateTable();
            saveContacts();
            showInfo("Контакт удален");
        }
    }
}

// поиск контактов
void MainWindow::searchContacts(){
    int fieldIndex = m_searchFieldCombo->currentIndex();

    // Поиск по дате рождения
  if (fieldIndex == 7) {
    QDate searchDate = m_searchDateEdit->date();
    int dateSearchType = m_dateSearchTypeCombo->currentIndex();

    m_table->setSortingEnabled(false);
    m_table->setRowCount(0);

    const QList<Contact>& contacts = m_storage->contacts();
    for (int i = 0; i < contacts.size(); ++i) {
      const Contact& contact = contacts[i];
      bool matches = false;

      switch (dateSearchType) {
      case 0: // Точная дата
        matches = (contact.birthDate() == searchDate);
        break;
      case 1: // Только год
        matches = (contact.birthDate().year() ==
   searchDate.year());
        break;
      case 2: // Месяц и год
        matches = (contact.birthDate().year() == searchDate.year()
   &&
            contact.birthDate().month() == searchDate.month());
        break;
      }

```

```
591         if (matches) {
592             int row = m_table->rowCount();
593             m_table->insertRow(row);
594             m_table->setItem(row, 0, new
    QTableWidgetItem(contact.lastName()));
595             m_table->setItem(row, 1, new
    QTableWidgetItem(contact.firstName()));
596             m_table->setItem(row, 2, new
    QTableWidgetItem(contact.middleName()));
597             m_table->setItem(row, 3, new
    QTableWidgetItem(contact.address()));
598             m_table->setItem(row, 4, new
    QTableWidgetItem(contact.birthDate().toString("dd.MM.yyyy")));
599             m_table->setItem(row, 5, new
    QTableWidgetItem(contact.email()));
600             m_table->setItem(row, 6, new
    QTableWidgetItem(contact.phoneNumbers().join(", ")));
601         }
602     }
603
604     m_table->setSortingEnabled(true);
605     m_table->resizeColumnsToContents();
606     return;
607 }
608
609     // Обычный текстовый поиск
610     QString searchText = m_searchEdit->text().trimmed();
611     if (searchText.isEmpty()) {
612         populateTable();
613         return;
614     }
615
616     QString searchLower = searchText.toLower();
617
618     m_table->setSortingEnabled(false);
619     m_table->setRowCount(0);
620
621     const QList<Contact>& contacts = m_storage->contacts();
622     for (int i = 0; i < contacts.size(); ++i) {
623         const Contact& contact = contacts[i];
624         bool matches = false;
625
626         switch (fieldIndex) {
627         case 0: // все поля
628             matches =
    contact.lastName().toLower().contains(searchLower) ||
629
    contact.firstName().toLower().contains(searchLower) ||
630
    contact.middleName().toLower().contains(searchLower) ||
631
    contact.address().toLower().contains(searchLower) ||
```

```
632
    contact.email().toLower().contains(searchLower) ||
633                         contact.phoneNumbers().join("
    ").toLower().contains(searchLower);
634             break;
635         case 1: matches =
    contact.lastName().toLower().contains(searchLower); break;
636         case 2: matches =
    contact.firstName().toLower().contains(searchLower); break;
637         case 3: matches =
    contact.middleName().toLower().contains(searchLower); break;
638         case 4: matches =
    contact.address().toLower().contains(searchLower); break;
639         case 5: matches =
    contact.email().toLower().contains(searchLower); break;
640         case 6: matches = contact.phoneNumbers().join("
    ").toLower().contains(searchLower); break;
641         }
642
643         if (matches) {
644             int row = m_table->rowCount();
645             m_table->insertRow(row);
646             m_table->setItem(row, 0, new
    QTableWidgetItem(contact.lastName()));
647             m_table->setItem(row, 1, new
    QTableWidgetItem(contact.firstName()));
648             m_table->setItem(row, 2, new
    QTableWidgetItem(contact.middleName()));
649             m_table->setItem(row, 3, new
    QTableWidgetItem(contact.address()));
650             m_table->setItem(row, 4, new
    QTableWidgetItem(contact.birthDate().toString("dd.MM.yyyy")));
651             m_table->setItem(row, 5, new
    QTableWidgetItem(contact.email()));
652             m_table->setItem(row, 6, new
    QTableWidgetItem(contact.phoneNumbers().join(", ")));
653         }
654     }
655
656     m_table->setSortingEnabled(true);
657     m_table->resizeColumnsToContents();
658 }
659
660 // сброс поиска
661 void MainWindow::clearSearch(){
662     m_searchEdit->clear();
663     populateTable();
664 }
665
666 // реакция на выбор в таблице
667 void MainWindow::onTableSelectionChanged(){
668     bool hasSelection = !m_table->selectedItems().isEmpty();
669     m_editButton->setEnabled(hasSelection && !m_isEditing);
```

```
670     m_deleteButton->setEnabled(hasSelection && !m_isEditing);
671 }
672 
673 void MainWindow::onTableSortChanged(int column){
674     Q_UNUSED(column);
675 }
676 
677 // загрузка из файла
678 void MainWindow::loadContacts(){
679     m_storage->load();
680     populateTable();
681 }
682 
683 // сохранение в файл
684 void MainWindow::saveContacts(){
685     m_storage->save();
686 }
```