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 _____ Селезнев К. Д.

Руководитель,
Преподаватель _____ Глазунов В.В.

«____»_____ 2026 г.

Санкт-Петербург, 2026

Содержание

1 Введение

В работе рассматривается разработка приложения «Телефонный справочник» с использованием фреймворка Qt. Приложение относится к классу систем управления данными и демонстрирует принципы объектно-ориентированного программирования, работу с пользовательским вводом и хранение информации в файле.

1.1 Фреймворк Qt

Qt представляет собой кроссплатформенную библиотеку для создания графических интерфейсов и работы с данными. В работе используются модули Qt Widgets для построения интерфейса, механизм сигналов и слотов для организации взаимодействия между объектами, а также классы QFile и QRegularExpression для работы с файлами и проверки корректности ввода. Дополнительно применяются средства для работы с датами и контейнерами, что позволяет реализовать хранение и обработку контактной информации.

1.2 Предметная область

Приложение «Телефонный справочник» предназначено для управления контактной информацией о пользователях. Каждый контакт включает имя, фамилию, отчество, адрес проживания, дату рождения, электронную почту и телефонные номера. Система должна обеспечивать добавление, редактирование и удаление записей, а также их поиск по выбранному столбцу и сортировку также по выбранному столбцу. Для хранения данных используется текстовый файл, а для представления информации пользователю — табличный интерфейс.

2 Постановка задачи

Целью работы является разработка программного приложения «Телефонный справочник», обеспечивающего хранение, обработку и управление контактной информацией пользователей с использованием локальной базы данных в виде текстового файла.

Для достижения поставленной цели необходимо решить следующие задачи:

1. **Определить структуру данных для хранения информации о контакте.**
   Требуется разработать класс, включающий следующие поля: фамилия, имя, отчество, адрес проживания, дата рождения, адрес электронной почты и список телефонных номеров. Структура должна обеспечивать строгую валидацию каждого поля, преобразование объекта в строку для сохранения в файл и восстановление объекта из строки при загрузке.

2. **Разработать формат хранения данных.**
   Необходимо определить единый текстовый файл, где каждая строка соответствует одному контакту, выбрать фиксированный разделитель полей, задать правила нормализации данных (удаление лишних пробелов), а также запретить использование разделителя внутри значений.

3. **Реализовать механизм загрузки данных из файла.**
   Программа должна открывать файл, считывать строки до конца файла, корректно обрабатывать пустые строки и ошибки формата, создавать объекты на основе строк и формировать внутренний список контактов.

4. **Реализовать механизм сохранения данных в файл.**
   Требуется сохранять контакты после операции изменения, записывать данные в текстовом виде, гарантировать корректность формата.

5. **Реализовать функциональность добавления контактов.**
   Необходимо создать диалоговое окно ввода данных, выполнить валидацию всех полей, добавить новый контакт в список, обновить таблицу.

6. **Реализовать функциональность редактирования контактов.**
   Программа должна корректно определять реальный индекс контакта даже после сортировки, открывать диалог с предзаполненными данными, выполнять повторную валидацию.

7. **Реализовать функциональность удаления контактов.**
   Требуется определить выбранную строку, запросить подтверждение удаления, удалить соответствующий объект из списка, обновить таблицу.

8. **Разработать пользовательский интерфейс для работы со справочником.**
   Интерфейс должен обеспечивать отображение контактов в табличном виде, выбор строк, сортировку по каждому столбцу, поиск по выбранному полю и динамическое скрытие / отображение строк при фильтрации.

   - **Реализовать сортировку данных.**
     Пользователь должен иметь возможность упорядочить контакты по любому полю. Сортировка должна выполняться через клик по заголовку столбца в таблице. При этом после сортировки должен корректно запоминаться индекс уже созданного контакта, чтобы впоследствии при редактировании и удалении выбирался нужный.
   - **Обеспечить корректную работу поиска и фильтрации.**
     Программа должна выполнять поиск по выбранному столбцу, учитывать особенности поля телефонов (несколько значений), обновлять отображение таблицы в реальном времени.

9. **Реализовать валидацию всех полей контакта.**
   Необходимо формализовать требования к каждому типу данных и обеспечить проверку корректности пользовательского ввода до сохранения информации.

   Требования включают:

   - **Валидация имени, фамилии и отчества.**
     Длина строки после удаления пробелов по краям должна быть больше нуля. Первый символ обязан быть заглавной буквой кириллицы или латиницы, последующие - строчными. Допустимые символы: буквы, пробелы, дефисы, цифры. Строка не может начинаться или заканчиваться дефисом. Пробелы в начале и конце должны удаляться автоматически.
   - **Валидация телефонного номера.**
     Допускается международный формат с символом «+» или национальный формат. Разрешённые символы: цифры, символ «+», пробелы. Примеры корректных форматов: `+79287766000`, `89287766000`. При сохранении номер должен нормализоваться до последовательности цифр и символа «+». Должна быть реализована проверка минимальной длины нормализованного номера.
   - **Валидация даты рождения.**
     Дата должна быть валидной и строго меньше текущей даты. Формат ввода должен обеспечивать корректность выбора даты пользователем.

- **Валидация адреса электронной почты.**
  Формат должен соответствовать схеме `username@domain.zone`. Имя пользователя — одна или более латинских букв или цифр. Обязательно наличие символа «@». Доменное имя — одна или более латинских букв или цифр. После домена обязательно должна следовать точка. Доменная зона — одна или более латинских букв или цифр. Пробелы в строке должны автоматически удаляться.

3 Реализация

3.1 Архитектура приложения

Приложение построено с использованием объектно-ориентированного подхода и разделения ответственности между компонентами. Структура проекта организована следующим образом: в директории `headers` располагаются заголовочные файлы классов: Contact.h для структуры данных контакта, ContactDialog.h для диалогового окна редактирования и PhoneBook.h для главного окна приложения. Директория `src` содержит файлы реализации соответствующих классов, а также файл main.cpp с точкой входа в программу. Файл project.pro описывает конфигурацию проекта для системы сборки qmake.

Архитектура приложения включает несколько ключевых компонентов. Класс Contact представляет собой модель данных для хранения информации об одном контакте. Класс PhoneBook является главным окном приложения и содержит таблицу для отображения всех контактов, а также кнопки управления. Класс ContactDialog реализует диалоговое окно для добавления и редактирования контактов с полной валидацией вводимых данных.

3.2 Используемые библиотечные классы Qt

В процессе разработки приложения были использованы базовые классы ядра Qt, классы для работы с данными, а также виджеты графического интерфейса. Ниже приведено краткое описание основных библиотечных компонентов, применённых в реализации.

Базовые классы ядра Qt

- **QString** — класс для работы со строками. Используется для хранения текстовых полей контакта, обработки пользовательского ввода и формирования строк для записи в файл.
- **QList<T>** — контейнерный класс, реализующий список элементов. Применяется для хранения набора объектов `Contact` и списка телефонных номеров.
- **QDate** — класс для представления и проверки корректности дат. Используется для хранения даты рождения и выполнения валидации.
- **QFile** — класс для работы с файлами. Применяется при чтении и записи данных справочника.
- **QTextStream** — потоковый интерфейс для чтения и записи текстовых данных. Используется совместно с `QFile` для работы с UTF–8.

- **QRegularExpression** — класс регулярных выражений, применяемый при валидации строковых полей.

Классы, используемые при наследовании

- **QWidget** — базовый класс всех визуальных элементов. От него наследуется класс `PhoneBook`, реализующий главное окно приложения.
- **QDialog** — класс для создания диалоговых окон. От него наследуется `ContactDialog`, обеспечивающий ввод и редактирование данных контакта.

Классы графического интерфейса

- **QTableWidget** — табличный виджет, используемый для отображения списка контактов.
- **QTableWidgetItem** — элемент таблицы, применяемый для заполнения ячеек и хранения пользовательских данных (в частности, индекса контакта).
- **QLineEdit** — однострочное поле ввода. Используется для ввода имени, фамилии, отчества, адреса, email и поиска.
- **QDateEdit** — виджет для выбора даты с поддержкой календаря. Применяется для ввода даты рождения.
- **QPushButton** — кнопки управления (добавление, редактирование, удаление, сохранение, загрузка).
- **QComboBox** — выпадающий список, используемый для выбора поля поиска.
- **QMessageBox** — стандартные диалоговые окна для отображения предупреждений и запросов подтверждения.
- **QFormLayout**, **QVBoxLayout**, **QHBoxLayout** — классы компоновки, применяемые для организации интерфейса.

Данные классы обеспечивают работу пользовательского интерфейса, обработку данных, взаимодействие с файлом и реализацию механизма валидации.

3.3 Класс Contact

3.3.1 Структура класса Contact

Класс `Contact` представляет собой модель данных, описывающую один контакт телефонного справочника. Структура класса включает:

Поля:

- `QString lastName` — фамилия;
- `QString firstName` — имя;
- `QString middleName` — отчество;
- `QString address` — адрес проживания;
- `QDate birthDate` — дата рождения;
- `QString email` — адрес электронной почты;
- `QStringList phones` — список телефонных номеров.

Методы:

- `toString()` — преобразование объекта в строку для сохранения в файл;
- `fromString()` — восстановление объекта из строки;
- геттеры и сеттеры для всех полей.

Исходный код класса приведён в Приложении **??**.

3.3.2 Метод toString()

Метод формирует строковое представление объекта `Contact` в формате, пригодном для записи в текстовый файл. Поля объединяются фиксированным разделителем, что обеспечивает однозначный разбор данных при загрузке.

Алгоритм метода включает:

- преобразование даты в формат `yyyy-MM-dd`;
- объединение списка телефонов в строку;
- формирование итоговой строки с разделителями.

Исходный код метода приведён в Приложении **??**.

3.3.3 Метод fromString()

Метод выполняет обратную операцию — преобразует строку из файла в объект `Contact`. Основные этапы:

- разбиение строки по разделителю;
- проверка корректности количества полей;
- преобразование даты из строкового формата;
- разбиение строки телефонов на список.

Исходный код метода приведён в Приложении **??**.

3.3.4 Геттеры и сеттеры

Геттеры обеспечивают доступ к приватным полям, а сеттеры — контролируемое изменение данных. Сеттеры используются также при загрузке данных из файла и при передаче данных между окнами интерфейса.

Исходный код геттеров и сеттеров приведён в Приложении **??**.

3.3.5 Особенности хранения данных

Класс `Contact` обеспечивает:

- строгую типизацию полей (например, дата хранится в `QDate`);
- независимость внутреннего представления от форматов ввода;
- корректное преобразование данных при сохранении и загрузке;
- возможность расширения структуры без изменения логики интерфейса.

3.4 Класс ContactDialog

3.4.1 Структура класса ContactDialog

Класс `ContactDialog` наследуется от `QDialog` и реализует диалоговое окно для ввода и редактирования данных контакта.

Поля:

- `QLineEdit *lastNameEdit`;

- `QLineEdit *firstNameEdit`;
- `QLineEdit *middleNameEdit`;
- `QLineEdit *addressEdit`;
- `QDateEdit *birthDateEdit`;
- `QLineEdit *emailEdit`;
- `QLineEdit *phonesEdit`.

Методы:

- `validateName()` — проверка корректности ФИО;
- `validatePhone()` — проверка телефонного номера;
- `validateEmail()` — проверка адреса электронной почты;
- `validateAndAccept()` — комплексная проверка всех полей и подтверждение ввода;
- `setContact()` — заполнение формы существующими данными;
- `getContact()` — получение объекта `Contact` из данных формы.

Исходный код класса приведён в Приложении **??**.

3.4.2 Конструктор ContactDialog

Конструктор класса `ContactDialog` выполняет инициализацию диалогового окна и настройку всех элементов интерфейса. Логика конструктора включает следующие этапы:

1. Создание полей ввода:
   - `QLineEdit` для фамилии, имени, отчества, адреса и email;
   - `QDateEdit` для даты рождения с включённым календарём;
   - `QLineEdit` для списка телефонных номеров.
2. Настройка свойств виджетов:
   - установка текущей даты рождения по умолчанию;
   - ограничение максимальной даты рождения текущей датой минус один день;
   - установка placeholder-текста для поля телефонов.

3. Формирование компоновки:

   - создание `QFormLayout` для размещения полей ввода;
   - создание горизонтального блока кнопок OK/Cancel;
   - добавление всех элементов в основную компоновку.

4. Подключение сигналов и слотов:

   - сигнал `clicked()` кнопки OK подключается к слоту `validateAndAccept()`;
   - сигнал `clicked()` кнопки Cancel подключается к слоту `reject()` базового класса `QDialog`.

Исходный код конструктора приведён в Приложении **??**.

3.4.3 Метод validateName()

Метод проверяет корректность строковых полей ФИО.

Проверяются:

- отсутствие пустой строки после обрезки пробелов;
- корректность первой буквы (заглавная);
- корректность регистра остальных символов;
- отсутствие дефиса в начале и конце строки;
- соответствие регулярному выражению.

Исходный код метода приведён в Приложении **??**.

3.4.4 Метод validatePhone()

Метод проверяет корректность телефонного номера.

Проверяются:

- допустимые символы (цифры, пробелы, знак «+»);
- корректность международного или национального формата;
- минимальная длина нормализованного номера;
- отсутствие недопустимых символов.

Исходный код метода приведён в Приложении **??**.

3.4.5 Метод validateEmail()

Метод проверяет соответствие строки формату `username@domain.zone`.

Проверяются:

- наличие символа «@»;
- корректность имени пользователя;
- корректность доменного имени;
- наличие точки после домена;
- корректность доменной зоны.

Исходный код метода приведён в Приложении **??**.

3.4.6 Метод validateAndAccept()

Метод выполняет комплексную проверку всех полей формы.

Алгоритм:

- последовательный вызов методов валидации;
- вывод сообщения об ошибке при первой некорректности;
- создание объекта `Contact` при успешной проверке;
- закрытие диалога с результатом `Accepted`.

Исходный код метода приведён в Приложении **??**.

3.5 Класс PhoneBook

Класс `PhoneBook` представляет главное окно приложения и наследуется от `QWidget`. Он отвечает за отображение списка контактов, обработку пользовательских действий и взаимодействие с файловой системой.

3.5.1 Структура класса PhoneBook

Поля:

- `QTableWidget *table` — таблица для отображения контактов;
- `QLineEdit *searchEdit` — поле ввода поискового запроса;

- `QList<Contact> contacts` — внутренний список всех контактов;
- `QString filename` — путь к файлу данных.

Методы:

- `updateTable()` — обновление содержимого таблицы;
- `addContact()` — добавление нового контакта;
- `editContact()` — редактирование выбранного контакта;
- `deleteContact()` — удаление выбранного контакта;
- `searchByColumn(int)` — поиск по выбранному столбцу;
- `saveToFile()` — сохранение данных в файл;
- `loadFromFile()` — загрузка данных из файла.

Исходный код класса приведён в Приложении **??**.

3.5.2 Конструктор PhoneBook(QWidget *parent)

Конструктор класса `PhoneBook` выполняет инициализацию главного окна приложения, создание интерфейса и настройку всех элементов. Логика конструктора включает следующие этапы:

1. Настройка окна:
   - установка заголовка окна;
   - установка минимального размера;
   - определение пути к файлу данных.
2. Создание панели поиска:
   - создание поля ввода `QLineEdit`;
   - создание выпадающего списка `QComboBox` для выбора столбца поиска;
   - заполнение списка названиями столбцов.
3. Создание таблицы контактов:
   - создание `QTableWidget` с семью столбцами;
   - установка заголовков столбцов;
   - включение автоматической сортировки;
   - настройка режима выбора строк.

4. Создание панели кнопок:

   - создание кнопок «Добавить», «Редактировать», «Удалить», «Сохранить», «Загрузить».

5. Подключение сигналов и слотов:

   - сигнал `clicked()` кнопки «Добавить» -> слот `addContact()`;
   - сигнал `clicked()` кнопки «Редактировать» -> слот `editContact()`;
   - сигнал `clicked()` кнопки «Удалить» -> слот `deleteContact()`;
   - сигнал `clicked()` кнопки «Сохранить» -> слот `saveToFile()`;
   - сигнал `clicked()` кнопки «Загрузить» -> слот `loadFromFile()`;
   - сигнал изменения текста в поле поиска `textChanged()` вызывает лямбда-функцию, которая передаёт выбранный столбец в слот `searchByColumn(int)`.

6. Формирование основной компоновки:

   - размещение панели поиска;
   - размещение таблицы;
   - размещение панели кнопок;
   - установка компоновки в главное окно.

Исходный код конструктора приведён в Приложении **??**.

3.5.3 Метод updateTable()

Метод `updateTable()` синхронизирует визуальное представление таблицы с внутренним списком контактов.

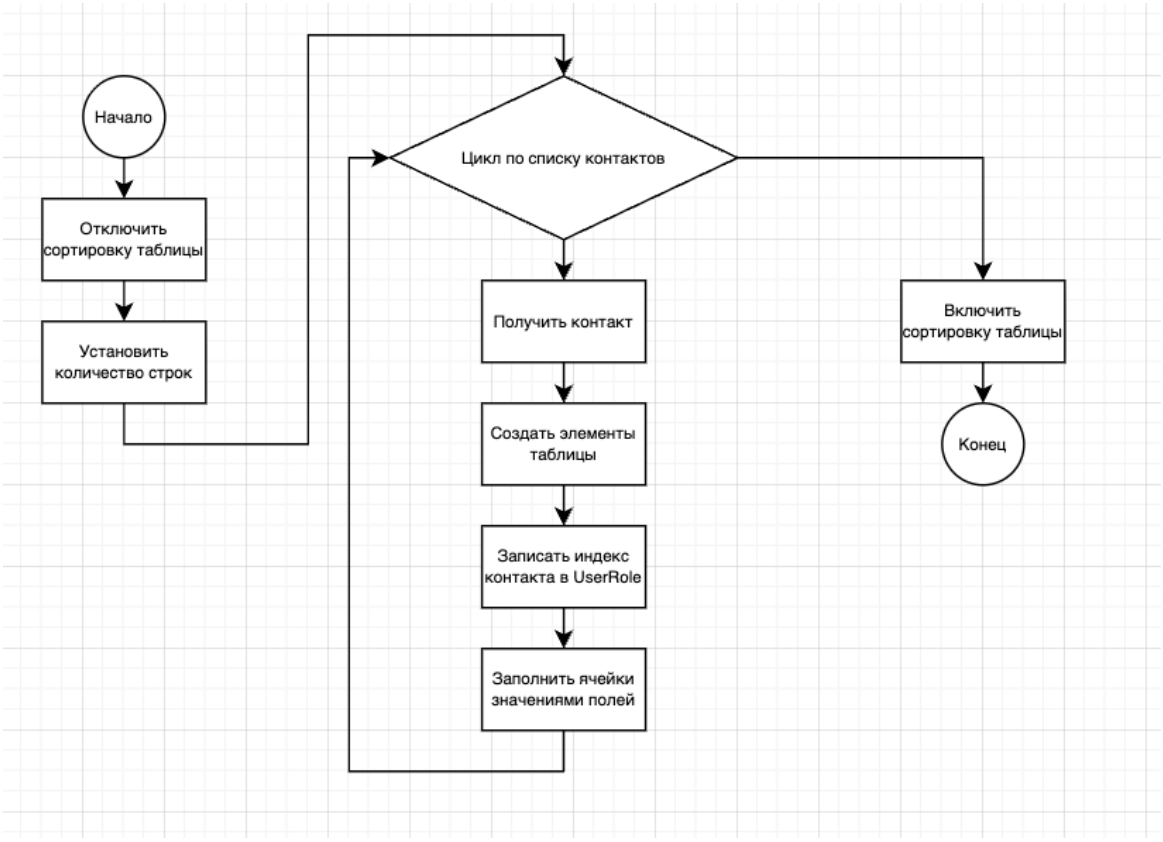

Рис. 1: Блок-схема алгоритма обновления таблицы контактов

Алгоритм работы включает:

1. временное отключение сортировки таблицы;
2. изменение количества строк в соответствии с размером списка;
3. для каждой строки:
   - получение объекта `Contact`;
   - создание элементов таблицы для каждого поля;
   - запись индекса контакта в `UserRole` первой ячейки;
   - заполнение остальных ячеек значениями полей;
4. повторное включение сортировки.

Исходный код метода приведён в Приложении **??**.

3.5.4 Метод addContact()

Метод `addContact()` отвечает за добавление нового контакта.

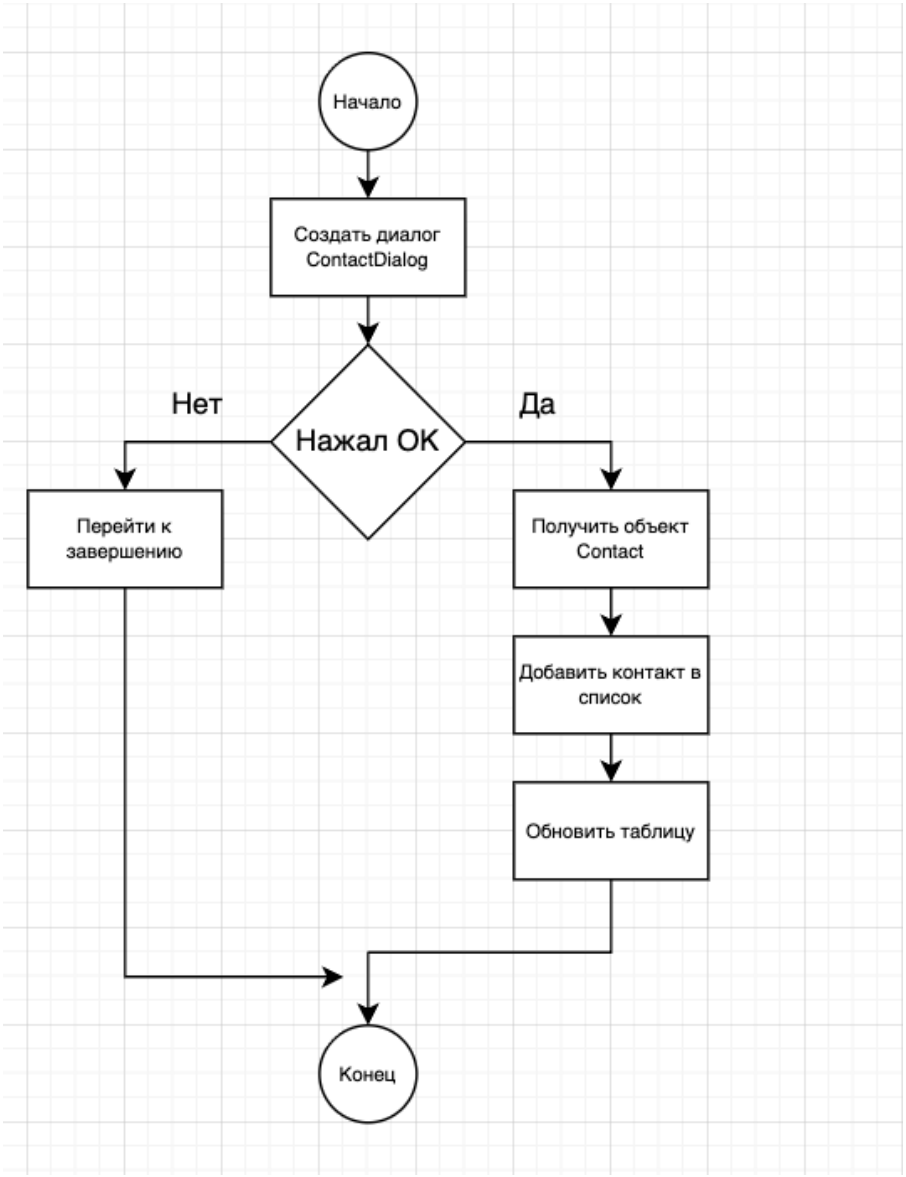

Рис. 2: Блок-схема алгоритма добавления нового контакта

Логика работы:

1. Создать диалоговое окно `ContactDialog`.
2. Открыть диалог и дождаться результата.
3. Если пользователь подтвердил ввод:
   - получить объект `Contact`;
   - добавить его в список `contacts`;
   - обновить таблицу вызовом `updateTable()`.

Исходный код метода приведён в Приложении ??.

3.5.5 Метод editContact()

Метод `editContact()` выполняет редактирование выбранного контакта.

Алгоритм:

1. Получить номер выбранной строки.
2. Если строка не выбрана — вывести предупреждение.
3. Извлечь реальный индекс контакта из `UserRole`.
4. Создать диалог `ContactDialog` и заполнить его текущими данными.
5. Если пользователь подтвердил изменения:
   - обновить объект в списке;
   - вызвать `updateTable()`.

Исходный код метода приведён в Приложении **??**.

3.5.6 Метод deleteContact()

Метод `deleteContact()` удаляет выбранный контакт.

Алгоритм:

1. Проверить, выбрана ли строка.
2. Если нет — вывести предупреждение.
3. Запросить подтверждение удаления через `QMessageBox`.
4. Если пользователь подтвердил:
   - извлечь индекс контакта из `UserRole`;
   - удалить элемент из списка `contacts`;
   - обновить таблицу.

Исходный код метода приведён в Приложении **??**.

3.5.7 Метод searchByColumn(int column)

Метод `searchByColumn()` выполняет фильтрацию строк таблицы.

Алгоритм:

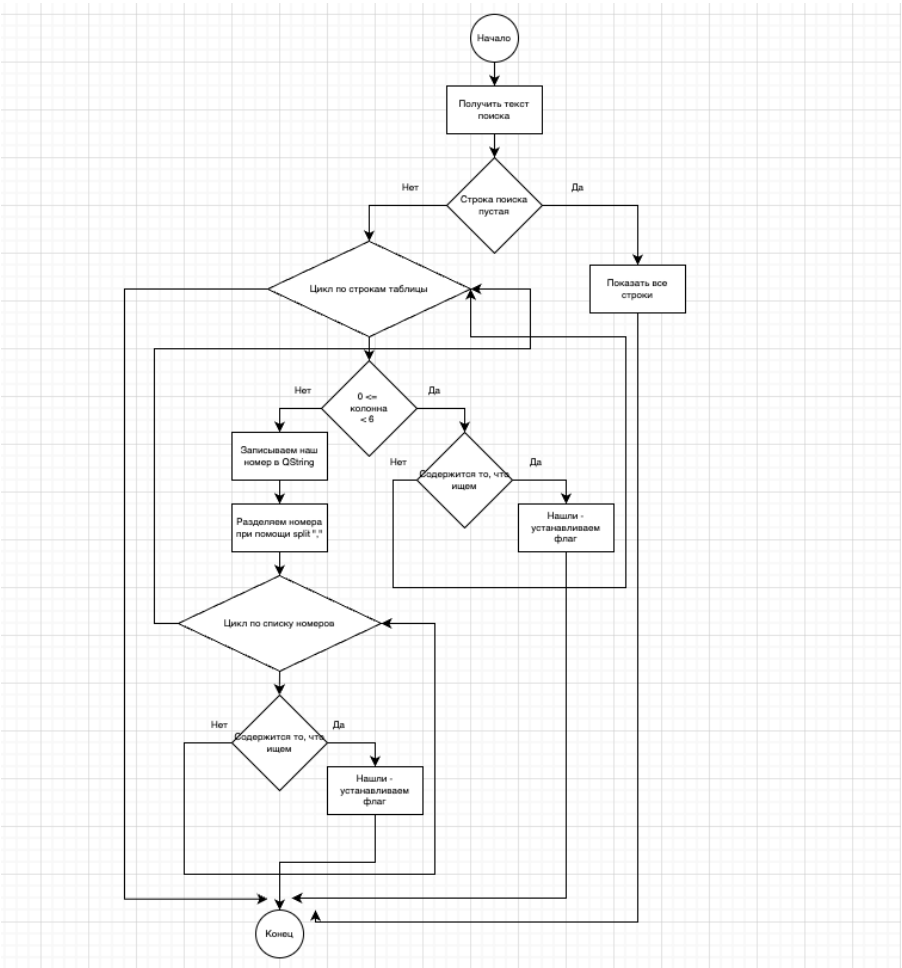

Рис. 3: Блок-схема алгоритма поиска по выбранному столбцу

Исходный код метода приведён в Приложении **??**.

3.5.8 Метод saveToFile()

Метод `saveToFile()` выполняет сохранение данных в текстовый файл.

Алгоритм:

1. Открыть файл в режиме записи.
2. Создать поток `QTextStream` с кодировкой UTF–8.
3. Для каждого контакта:
   - получить строковое представление через `toString()`;
   - записать строку в файл.
4. Закрыть файл.
5. Вывести сообщение об успешном сохранении.

Исходный код метода приведён в Приложении **??**.

3.5.9 Метод loadFromFile()

Метод `loadFromFile()` загружает данные из файла.

Алгоритм:

1. Открыть файл в режиме чтения.
2. Очистить текущий список контактов.
3. Читать файл построчно до конца.
4. Для каждой строки:
   - создать объект `Contact` через `fromString()`;
   - если данные корректны — добавить в список.
5. Закрыть файл.
6. Обновить таблицу.

Исходный код метода приведён в Приложении **??**.

3.5.10 Механизм сигналов и слотов

Класс `PhoneBook` использует механизм сигналов и слотов Qt для обработки действий пользователя. В конструкторе выполняются следующие подключения:

- кнопка «Добавить» -> слот `addContact()`;
- кнопка «Редактировать» -> слот `editContact()`;
- кнопка «Удалить» -> слот `deleteContact()`;
- кнопка «Сохранить» -> слот `saveToFile()`;
- кнопка «Загрузить» -> слот `loadFromFile()`;
- изменение текста в поле поиска вызывает лямбда-функцию, передающую выбранный столбец в `searchByColumn(int)`.

Исходный код подключений приведён в Приложении **??**.

3.6 Регулярные выражения, используемые в приложении

Валидация пользовательского ввода в приложении основана на использовании регулярных выражений. В данном разделе приведён общий синтаксис регулярных выражений, применяемый в Qt, а также частные выражения, использованные при проверке полей контакта.

3.6.1 Общий синтаксис регулярных выражений

Регулярные выражения позволяют описывать шаблоны строк с использованием специальных конструкций. Наиболее важные элементы синтаксиса:

- **Квантификаторы:**
  - `+` — один или более повторений;
  - `*` — ноль или более повторений;
  - `?` — ноль или одно повторение;
  - `{n,m}` — от `n` до `m` повторений.
- **Якоря:**
  - `^` — начало строки;
  - `$` — конец строки.
- **Символьные классы:**
  - `[abc]` — любой символ из набора;
  - `[a-z]` — диапазон символов;
  - `\d` — цифра;
  - `\s` — пробельный символ;
  - `\p{L}` — любая буква любого алфавита.
- **Группировка:**
  - `( ... )` — логическая группа символов.

Qt использует класс `QRegularExpression`.

3.6.2 Регулярные выражения, применённые в курсовой работе

Ниже приведены частные регулярные выражения, использованные для проверки корректности полей контакта.

1. Проверка имени, фамилии и отчества

```
^[\p{L}\d\s-]+$
```

Пояснение:

- `^` и `$` — строка должна полностью соответствовать шаблону;
- `\p{L}` — буквы любых алфавитов;
- `\d` — допускаются цифры;
- `\s` — пробелы внутри строки;
- `-` — разрешён дефис;
- `+` — один или более символов.

Это выражение используется после проверки регистра первой буквы и запрета дефиса в начале / конце строки.

2. Проверка телефонного номера

```
^\+?\d[\d\s]*$
```

Пояснение:

- `^` — начало строки;
- `?` — необязательный символ «+»;
- `\d` — номер должен начинаться с цифры или «+цифра»;
- `[\d\s]*` — цифры и пробелы в любом количестве;
- `$` — конец строки.

После проверки строка нормализуется до формата «+цифры».

3. Проверка адреса электронной почты

```
^[A-Za-z0-9]+@[A-Za-z0-9]+\.[A-Za-z0-9]+$
```

Пояснение:

- `[A-Za-z0-9]+` — имя пользователя: буквы и цифры;
- `@` — обязательный разделитель;
- `[A-Za-z0-9]+` — доменное имя;

- `.` — точка между доменом и зоной;
- `[A-Za-z0-9]+` — доменная зона.

Пробелы предварительно удаляются автоматически.

4. Проверка даты рождения

Дата проверяется не регулярным выражением, а средствами класса `QDate`, однако валидация включает:

- корректность календарной даты;
- проверку, что дата строго меньше текущей.

Таким образом, регулярные выражения используются только для строковых полей.

4 Тестирование приложения

В данном разделе приведено описание стартового окна приложения, доступных пользователю элементов интерфейса, а также сценарии тестирования в формате use-case. Для каждой функциональной задачи представлены корректные и некорректные варианты взаимодействия.

4.1 Описание стартового окна

После запуска приложения пользователь видит главное окно `PhoneBook`. Интерфейс включает следующие элементы:

- **Панель поиска**:
  - поле ввода поискового текста (`QLineEdit`);
  - выпадающий список выбора столбца (`QComboBox`);
  - оба элемента активны сразу после запуска.
- **Таблица контактов** (`QTableWidget`):
  - отображает список контактов (при первом запуске — пустая);
  - сортировка по столбцам активна;
  - режим выбора — по строкам;
  - редактирование ячеек пользователем отключено.
- **Панель кнопок управления**:
  - «Добавить» — активна;
  - «Редактировать» — активна, но требует выбранной строки;
  - «Удалить» — активна, но требует выбранной строки;
  - «Сохранить» — активна всегда;
  - «Загрузить» — активна всегда.

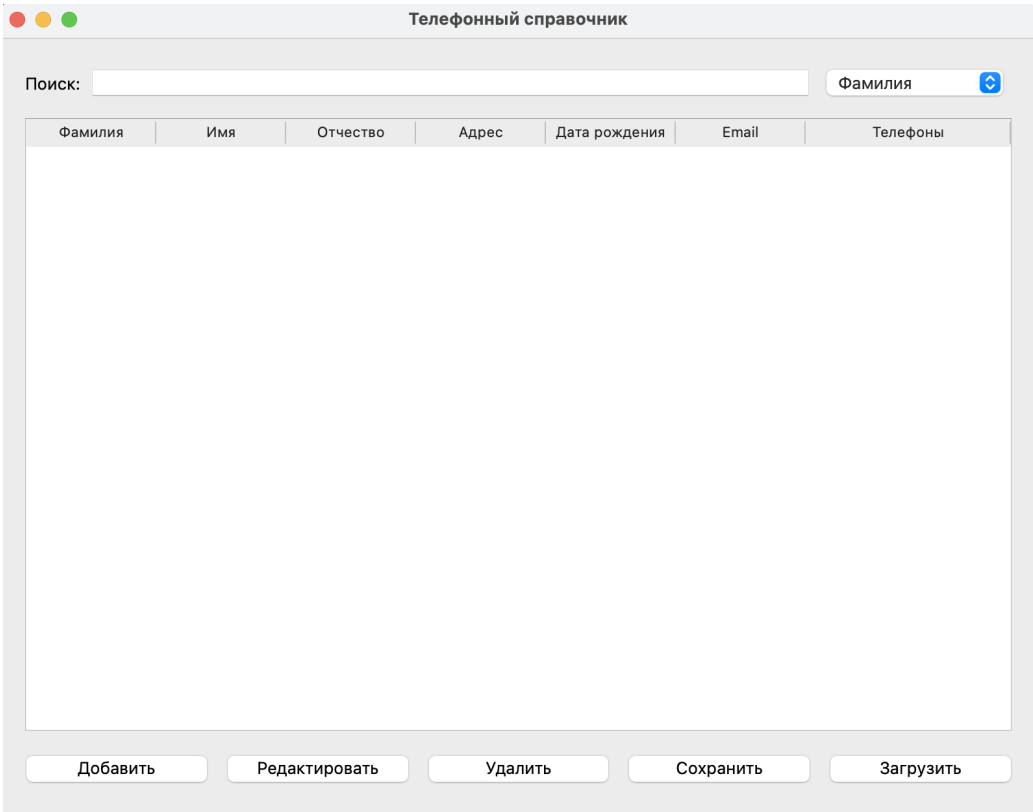

Рис. 4: Стартовое окно приложения: панель поиска, таблица контактов и панель управления

Таким образом, тестировщик сразу видит доступные действия и может проверить корректность поведения интерфейса при различных сценариях.

4.2 Use-case сценарии тестирования

Ниже приведены сценарии тестирования основных функций приложения, соответствующих задачам, описанным в разделе ??.

4.2.1 Use-case 1: Добавление нового контакта

Цель: пользователь добавляет корректный контакт в справочник.

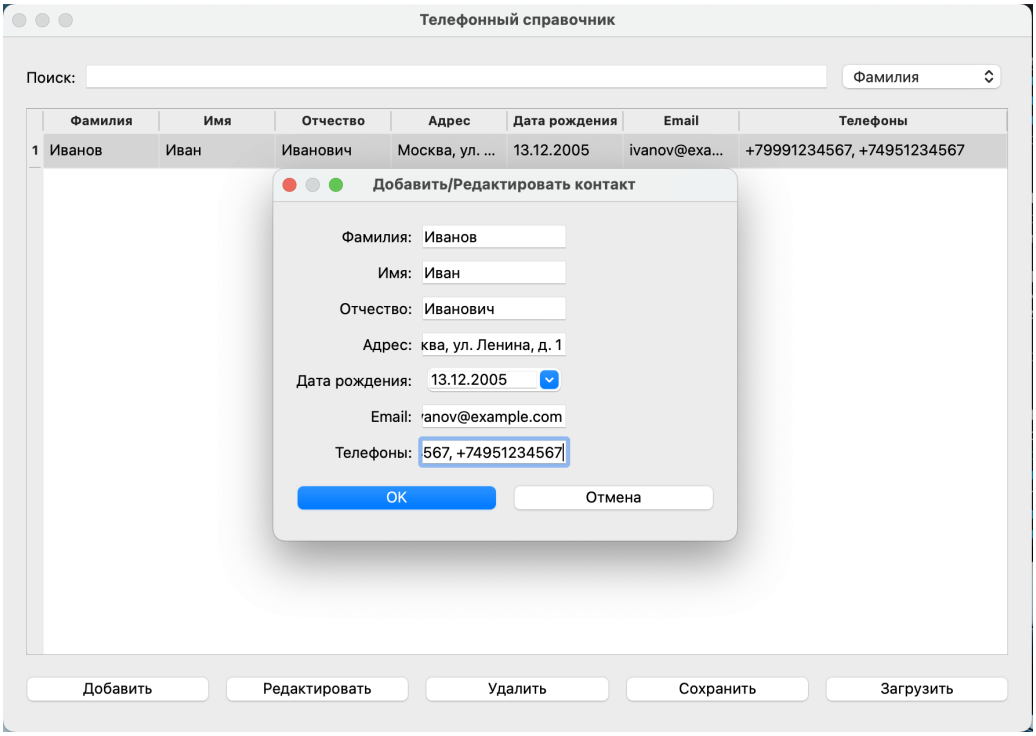

Рис. 5: Диалоговое окно добавления нового контакта

Основной сценарий (корректный):

1. Пользователь нажимает кнопку «Добавить».
2. Открывается диалоговое окно ввода данных.
3. Пользователь заполняет все поля корректными значениями:
   - ФИО — корректный формат;
   - дата рождения — меньше текущей;
   - email — соответствует шаблону;
   - телефоны — корректные номера.
4. Пользователь нажимает «ОК».
5. Контакт появляется в таблице.

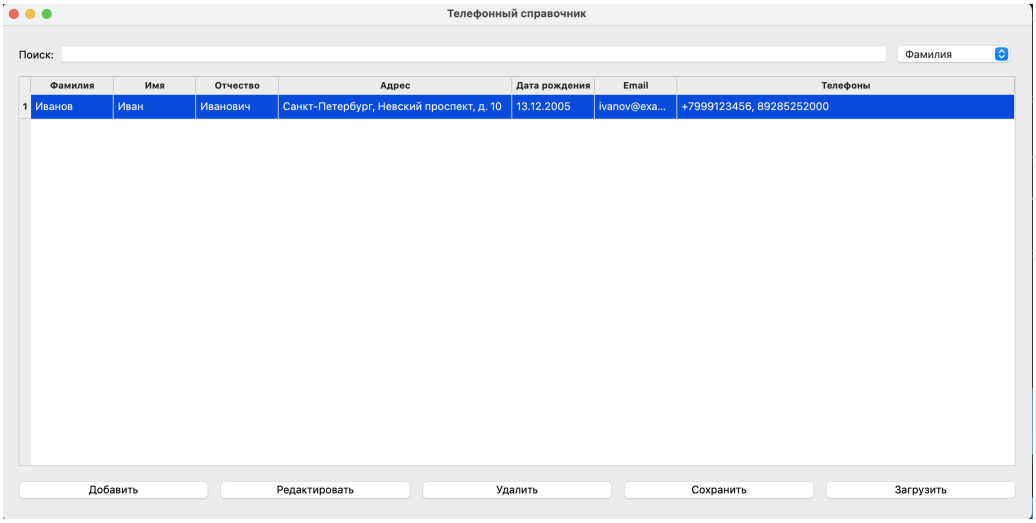

Рис. 6: Таблица после успешного добавления контакта

Альтернативные сценарии (некорректные):

- Поле ФИО пустое (см. рисунок **??**) -> выводится предупреждение, диалог не закрывается.
- ФИО начинается с маленькой буквы или недопустимого символа «-» (см. рисунки **??**) и **??**) -> предупреждение.
- Email без символа «@» (см. рисунок **??**) -> предупреждение.
- Телефон содержит недопустимые символы (см. рисунки **??** и **??**) -> предупреждение.
- Дата рождения больше текущей (см. рисунок **??**) -> предупреждение.

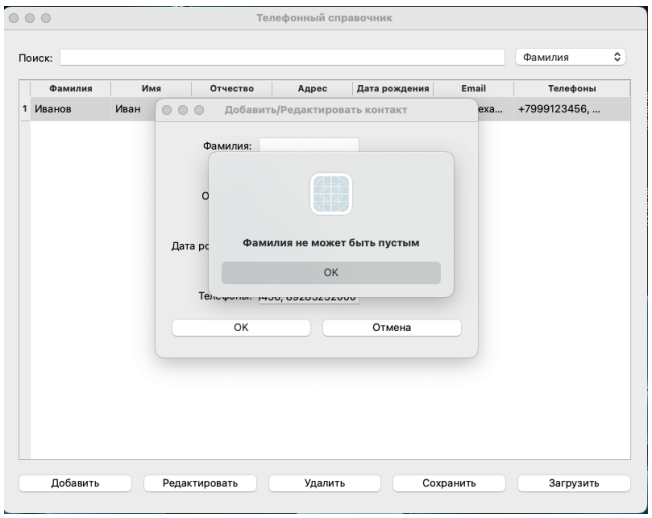

Рис. 7: Случай некорректной записи

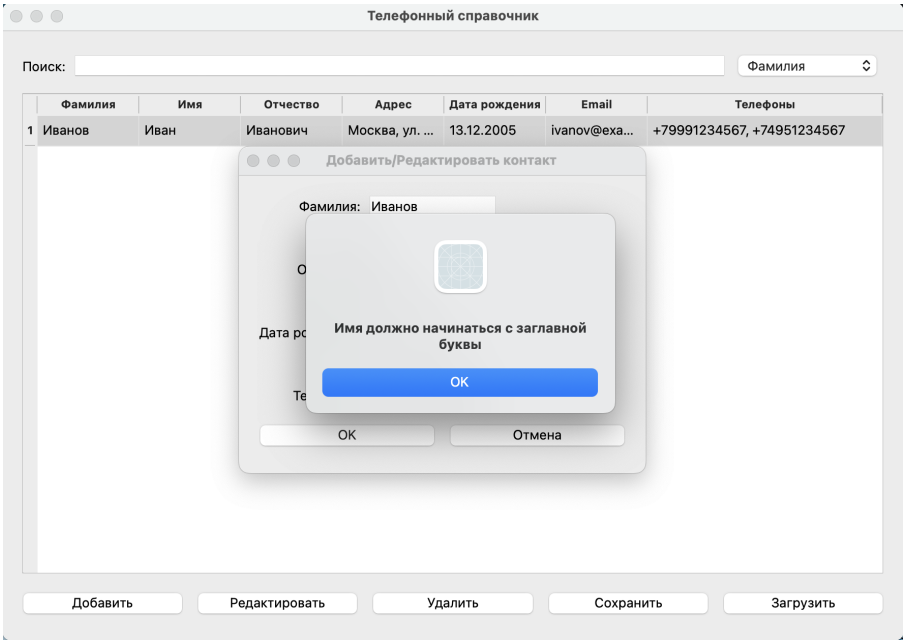

Рис. 8: Случай некорректной записи

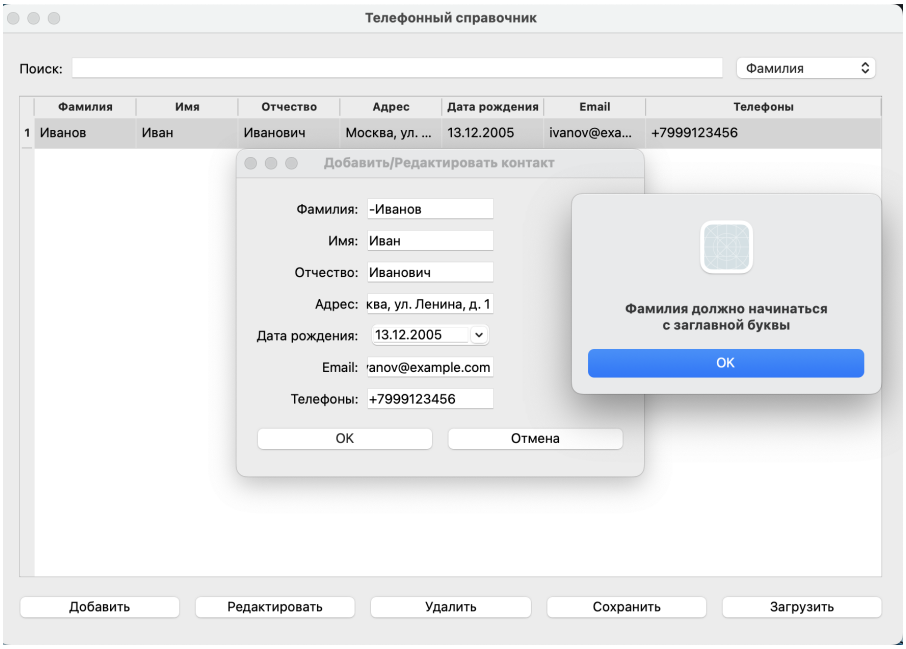

Рис. 9: Случай некорректной записи

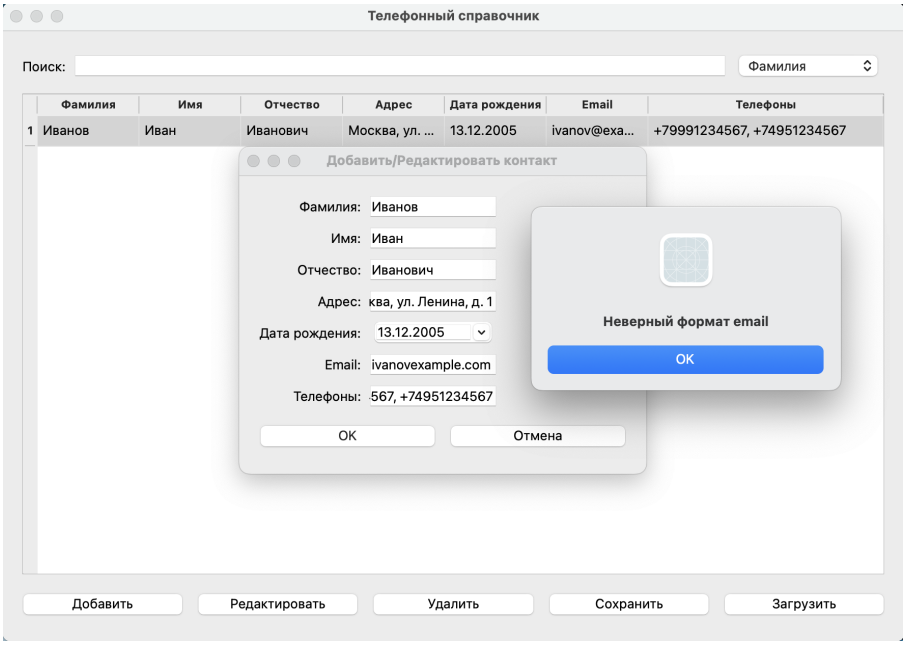

Рис. 10: Случай некорректной записи

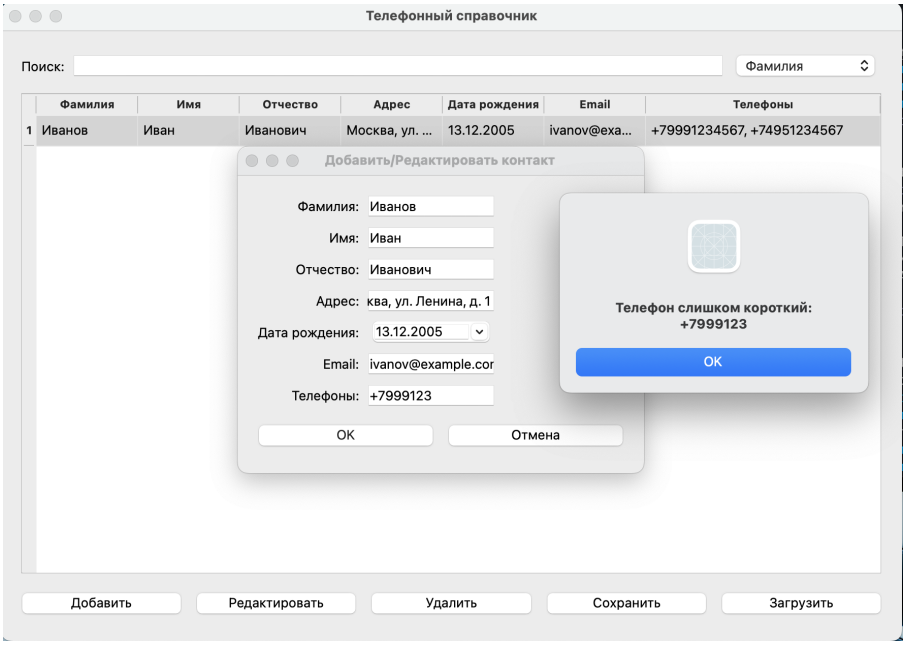

Рис. 11: Случай некорректной записи

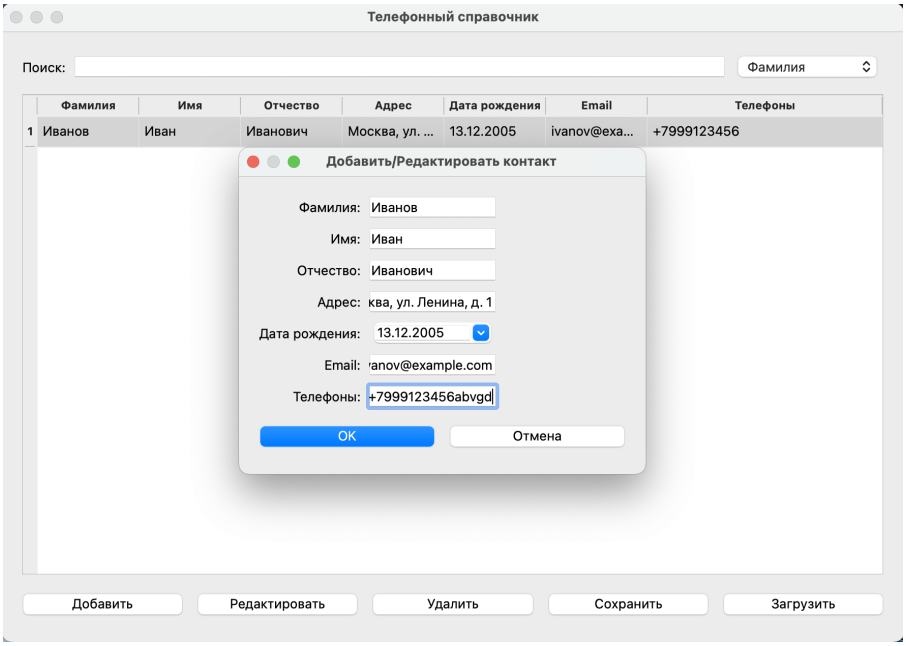

Рис. 12: Случай некорректной записи

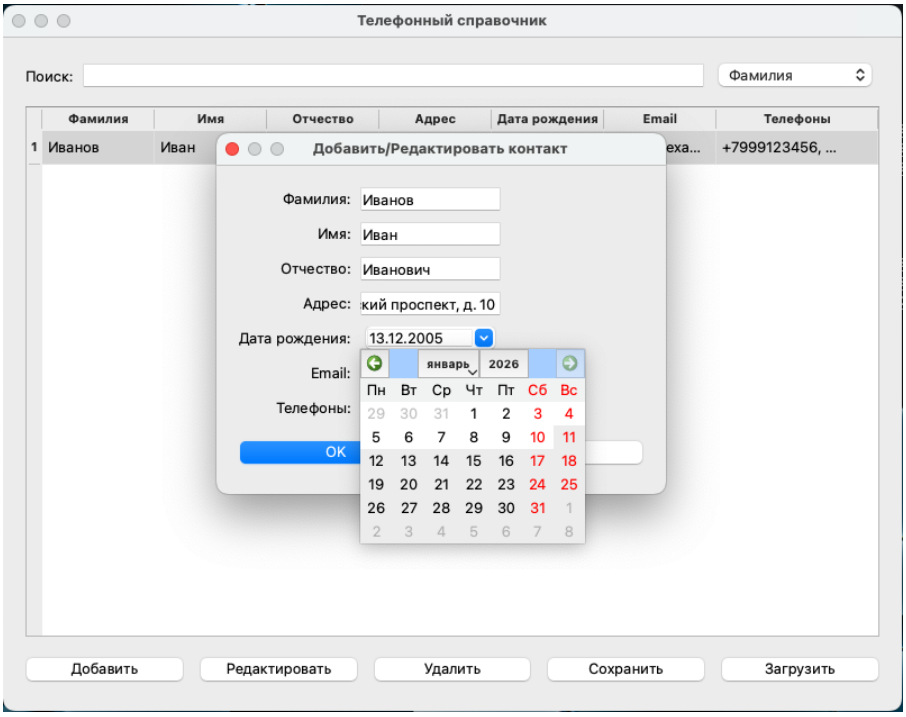

Рис. 13: Случай некорректной записи

4.2.2 Use-case 2: Редактирование существующего контакта

Цель: пользователь изменяет данные выбранного контакта.
Основной сценарий (корректный):

1. Пользователь выбирает строку в таблице.
2. Нажимает кнопку «Редактировать».

3. В диалоге изменяет необходимые поля.

4. Нажимает «OK».

5. Изменения отображаются в таблице.

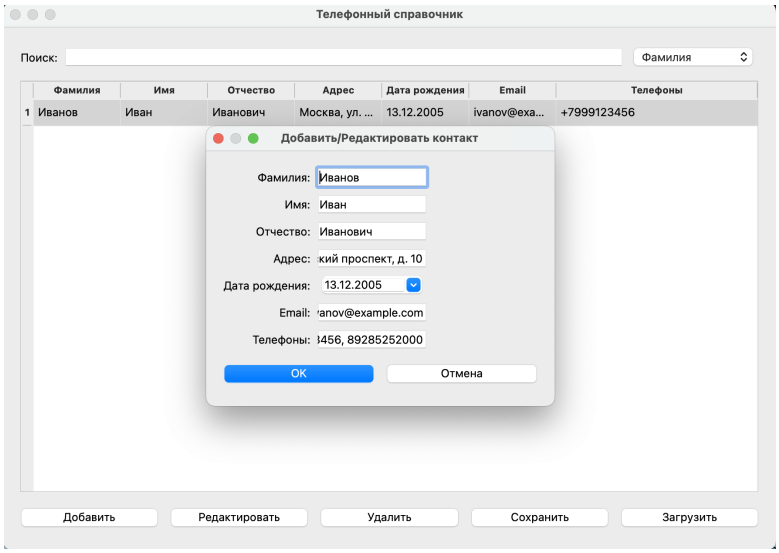

Рис. 14: Диалог редактирования выбранного контакта

Некорректные сценарии:

- Пользователь нажимает «Редактировать» без выбранной строки (см. рисунок **??**) -> предупреждение.
- Вводит некорректный email (см. рисунок **??**) -> предупреждение.

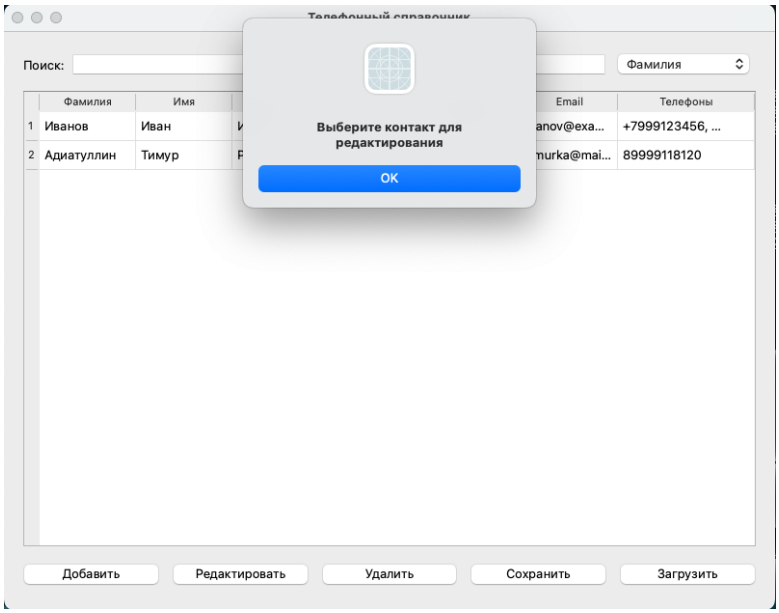

Рис. 15: Некорректный сценарий редактирования

4.2.3 Use-case 3: Удаление контакта

Цель: пользователь удаляет выбранный контакт.
Основной сценарий:

1. Пользователь выбирает строку.
2. Нажимает кнопку «Удалить».
3. Подтверждает удаление в диалоговом окне -> контакт исчезает из таблицы.
4. Пользователь нажимает «Нет» в диалоге подтверждения -> контакт остаётся.

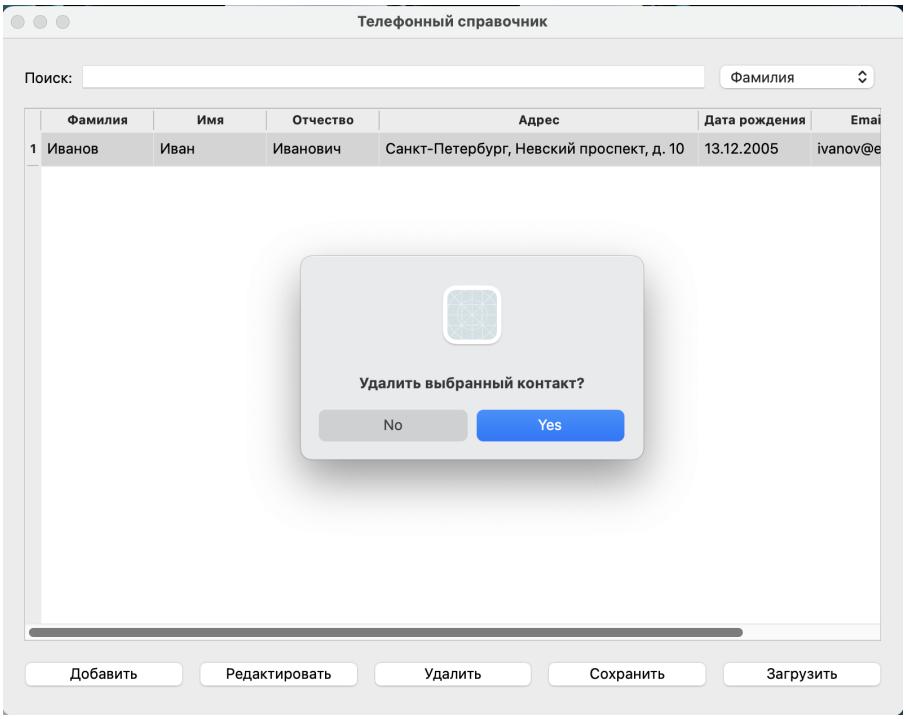

Рис. 16: Диалог подтверждения удаления контакта

Некорректные сценарии:

- Нажатие «Удалить» без выбранной строки (см. рисунок **??**) -> предупреждение.

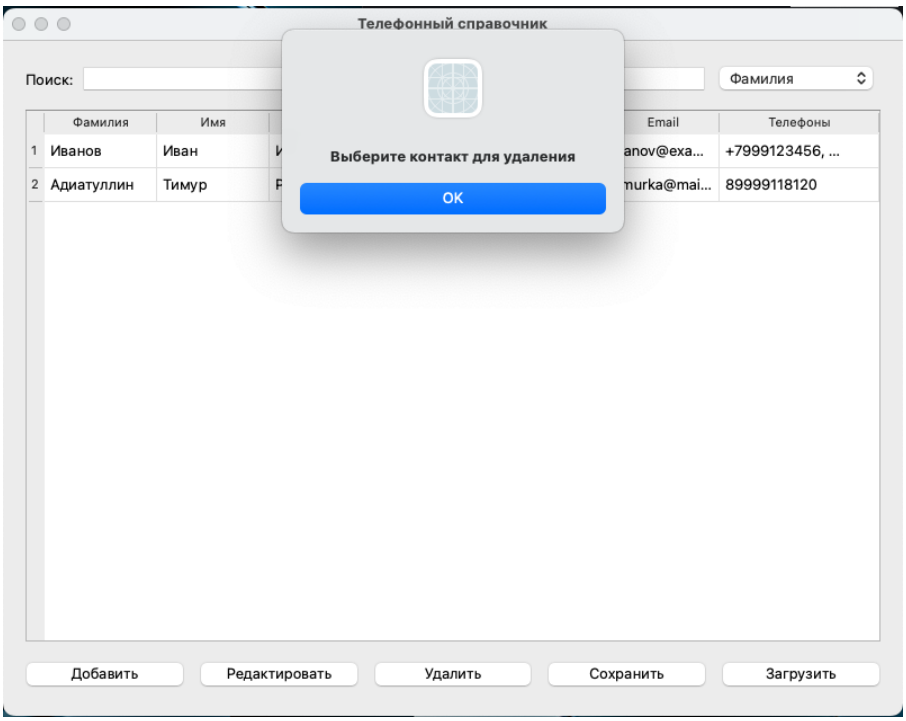

Рис. 17: Диалог подтверждения удаления контакта

4.2.4 Use-case 4: Поиск по выбранному столбцу

Цель: пользователь фильтрует контакты по введённому тексту.
Корректный сценарий:

1. Пользователь вводит текст в поле поиска.
2. Выбирает столбец (например, «Фамилия»), как показано на рисунке **??**.
3. Таблица динамически скрывает строки, не содержащие совпадений.

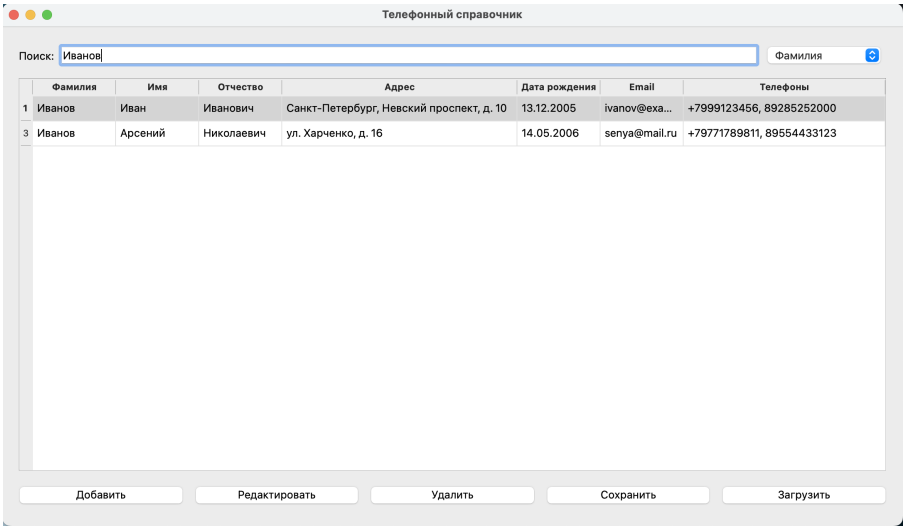

Рис. 18: Результат поиска по выбранному столбцу

Некорректные сценарии:

- Поиск по пустому тексту (см. рисунок **??**) -> должны отображаться все строки.
- Поиск по столбцу «Телефоны» с несколькими номерами (см. рисунки **??** и **??**) -> проверяется каждый номер.

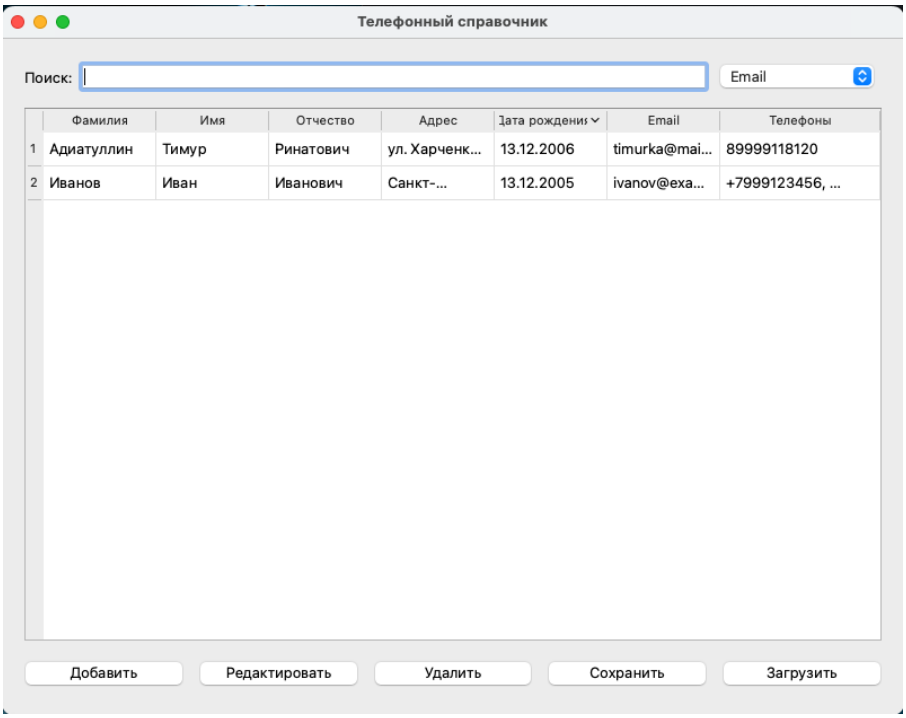

Рис. 19: Некорректный сценарий поиска

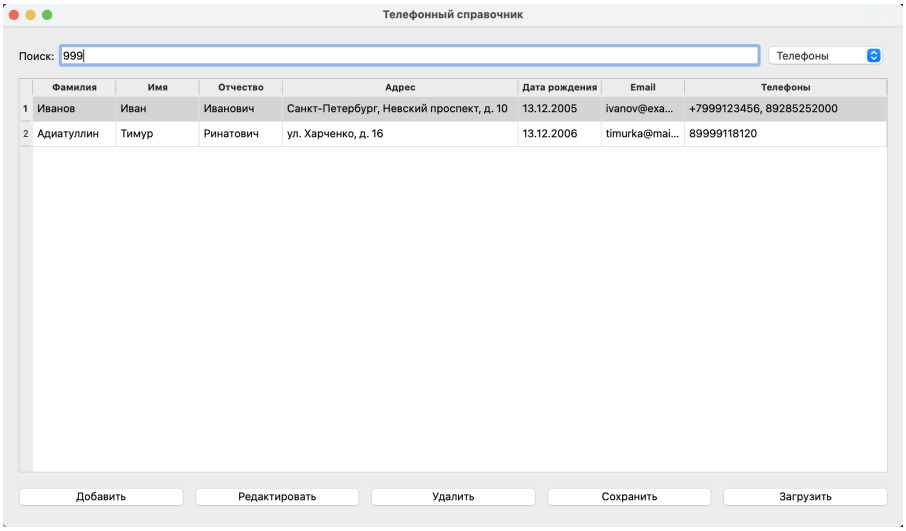

Рис. 20: Некорректный сценарий поиска

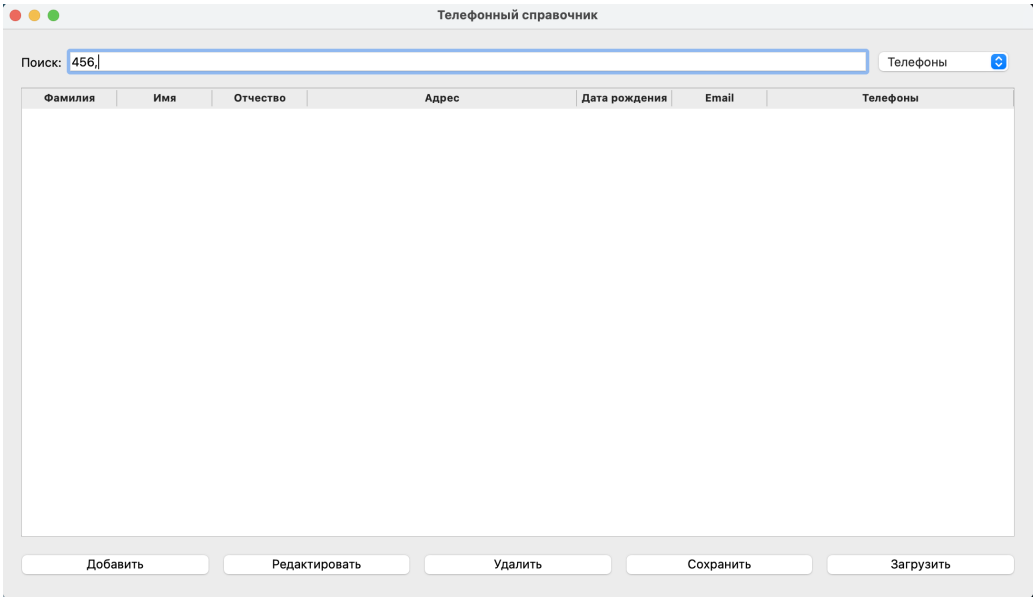

Рис. 21: Некорректный сценарий поиска

4.2.5 Use-case 5: Сохранение данных в файл

Корректный сценарий:

1. Пользователь нажимает «Сохранить».
2. Файл успешно создаётся или перезаписывается.
3. Появляется сообщение об успешном сохранении (см. рисунок **??**).

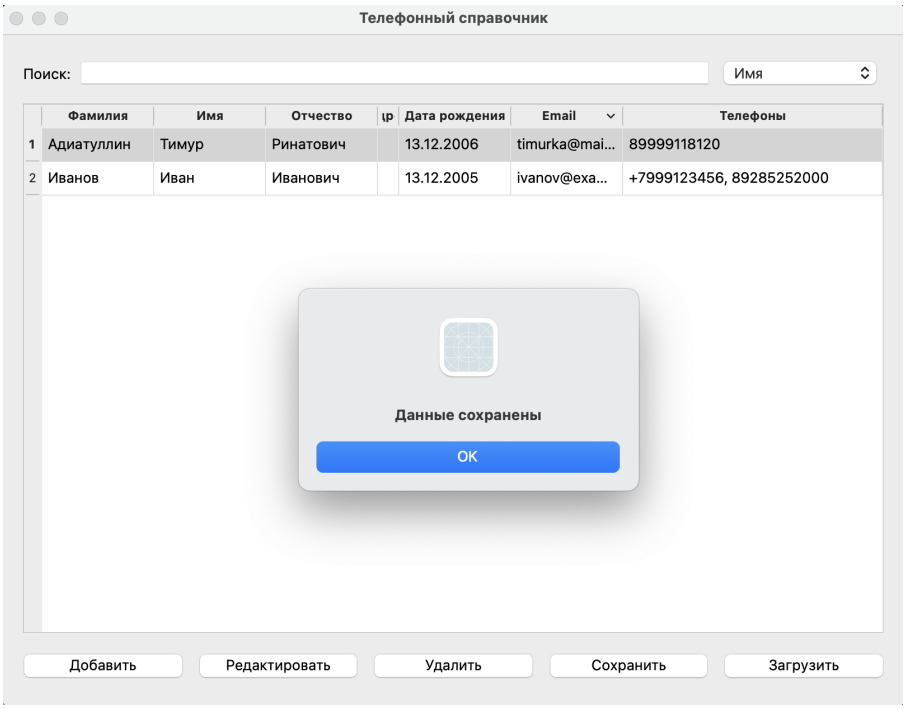

Рис. 22: Сообщение об успешном сохранении данных

4.2.6 Use-case 6: Загрузка данных из файла

Корректный сценарий:

1. Пользователь нажимает «Загрузить».
2. Файл открывается.
3. Контакты корректно отображаются в таблице (см. рисунок **??**).

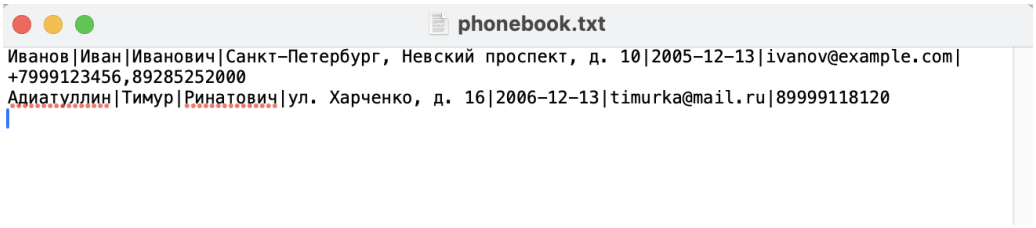

Рис. 23: Записи в файле

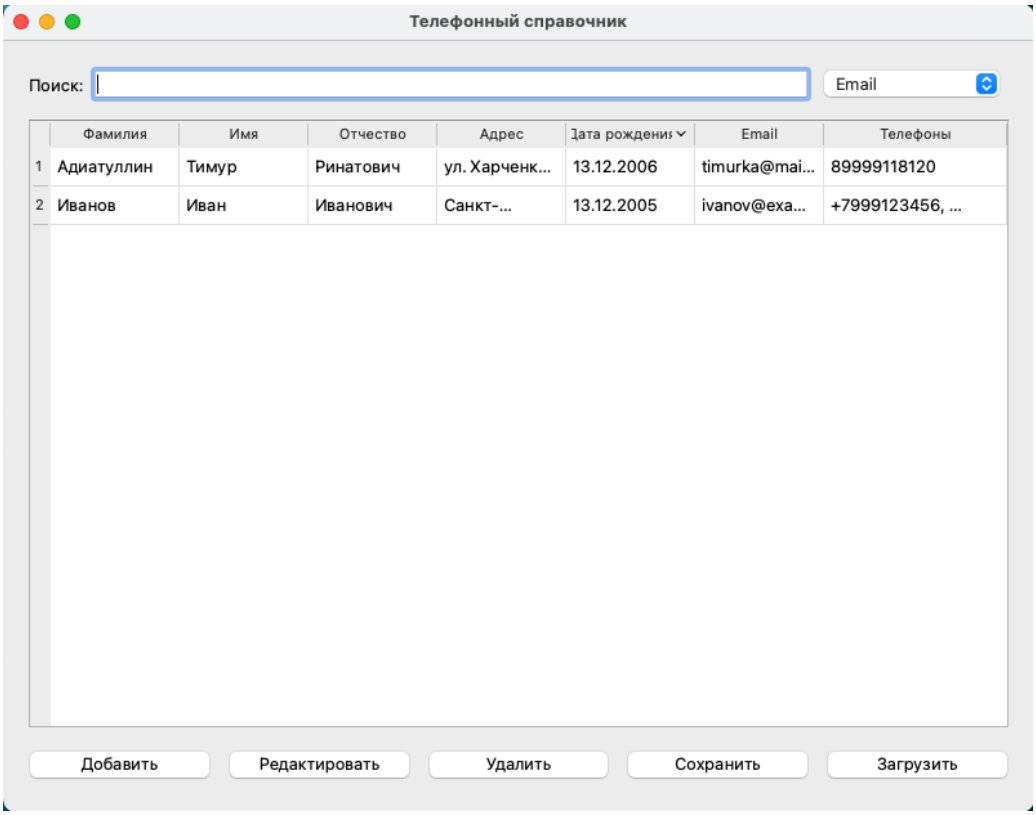

Рис. 24: Таблица после загрузки данных из файла

4.2.7 Сортировка по выбранным столбцам

Приложение поддерживает сортировку данных по любому столбцу таблицы. Сортировка выполняется автоматически при нажатии пользователем на заголовок столбца. Повторное нажатие изменяет направление сортировки

(по возрастанию или по убыванию). При этом корректно сохраняется связь между строкой таблицы и реальным индексом контакта, что обеспечивает правильную работу функций редактирования и удаления.

Ниже приведены скриншоты, демонстрирующие работу сортировки.

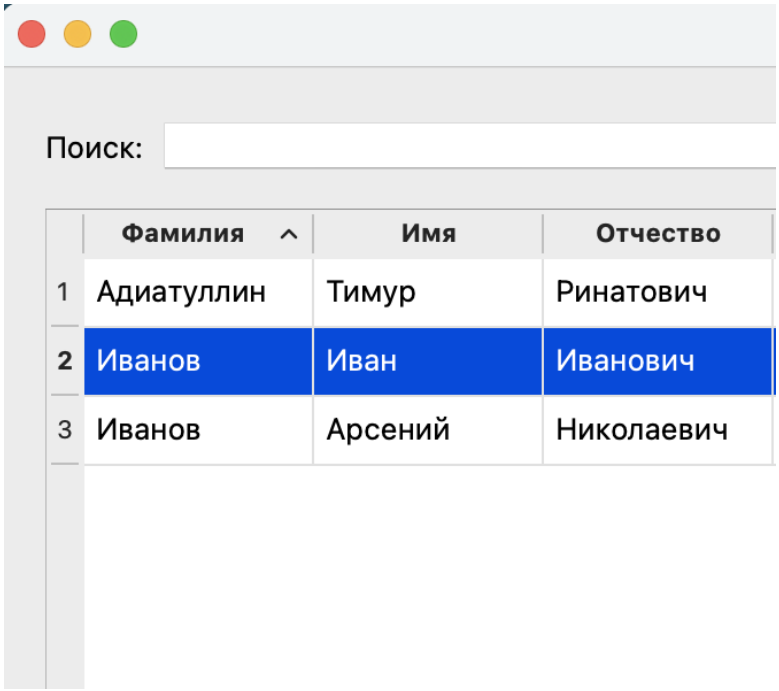

Рис. 25: Сортировка по фамилии

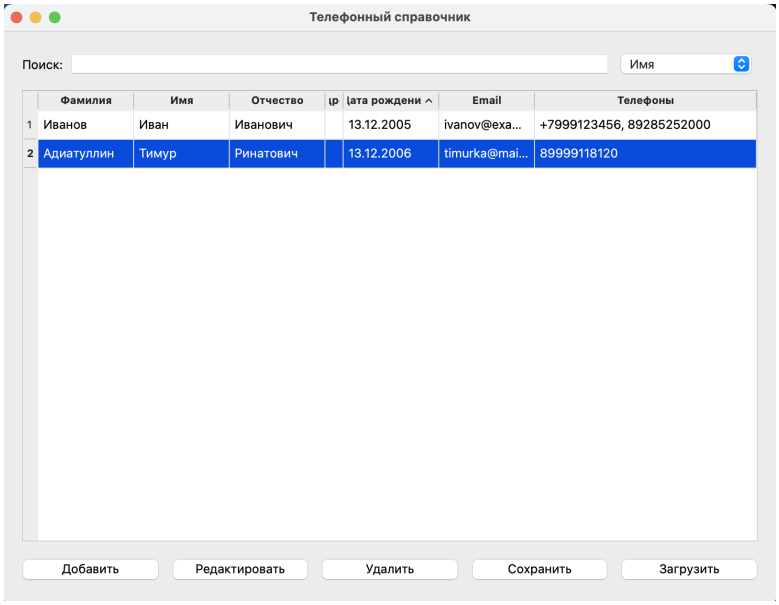

Рис. 26: Сортировка по дате рождения

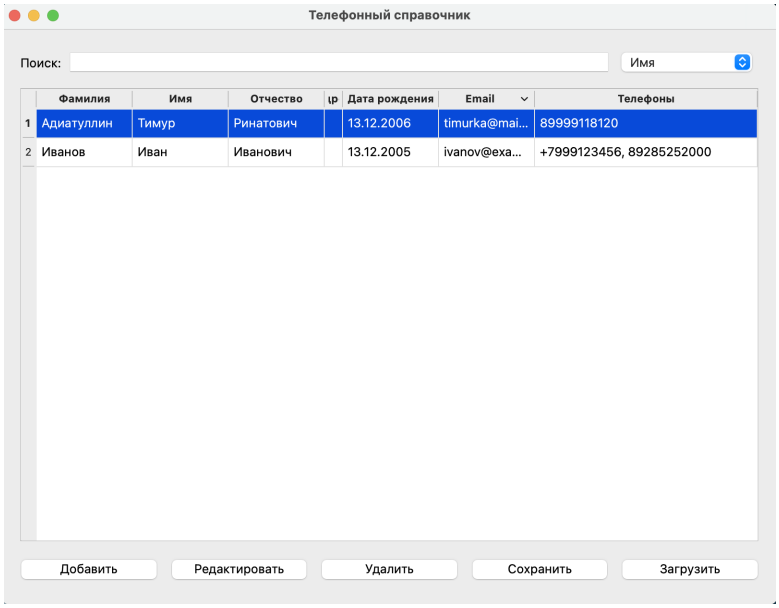

Рис. 27: Сортировка по электронной почте

Заключение

В ходе выполнения курсовой работы была разработана программная система «Телефонный справочник», реализованная с использованием фреймворка Qt. Работа охватывала полный цикл создания приложения: от проектирования структуры данных до разработки пользовательского интерфейса, механизмов валидации и тестирования. Ниже приведены результаты по каждому пункту постановки задачи.

1. Разработана структура данных для хранения информации о контакте. Создан класс `Contact`, включающий поля фамилии, имени, отчества, адреса, даты рождения, электронной почты и списка телефонных номеров. Реализованы методы `toString()` и `fromString()` для конвертации объекта в строку и восстановления данных при загрузке.

2. Определён формат хранения данных. Выбран текстовый файл с фиксированным разделителем полей. Реализована нормализация строк, удаление лишних пробелов и запрет использования разделителя внутри значений. Формат полностью поддерживается методами класса `Contact`.

3. Реализован механизм загрузки данных из файла. Метод `loadFromFile()` класса `PhoneBook` открывает файл, читает строки до конца, корректно обрабатывает пустые и некорректные строки, создаёт объекты `Contact` и формирует внутренний список контактов.

4. Реализован механизм сохранения данных в файл. Метод `saveToFile()` записывает текущий список контактов в файл в строгом текстовом формате. Используется поток `QTextStream` с кодировкой UTF–8, что обеспечивает корректность хранения данных.

5. Реализована функциональность добавления контактов. Метод `addContact()` открывает диалог `ContactDialog`, выполняет валидацию дан-

ных и добавляет новый объект `Contact` в список. После добавления вызывается `updateTable()` для обновления интерфейса.

6. Реализована функциональность редактирования контактов. Метод `editContact()` корректно определяет реальный индекс контакта через `UserRole`, открывает диалог с предзаполненными данными и выполняет повторную валидацию. После подтверждения изменения сохраняются и отображаются в таблице.

7. Реализована функциональность удаления контактов. Метод `deleteContact()` удаляет выбранный контакт из списка, предварительно запрашивая подтверждение через `QMessageBox`. Удаление корректно отражается в таблице благодаря вызову `updateTable()`.

8. Разработан пользовательский интерфейс. Создано главное окно `PhoneBook` с таблицей `QTableWidget`, панелью поиска, кнопками управления и поддержкой сортировки по столбцам. Реализован механизм поиска по выбранному полю методом `searchByColumn()`, включая обработку нескольких телефонных номеров в одной ячейке.

9. Реализована валидация всех полей контакта. В классе `ContactDialog` разработаны методы `validateName()`, `validatePhone()` и `validateEmail()`, использующие регулярные выражения и дополнительные проверки (корректность регистра, запрет дефиса в начале/конце, проверка даты рождения через `QDate`). Валидация выполняется перед созданием объекта `Contact`.

Таким образом, все задачи, поставленные в начале работы, были полностью выполнены. Разработанное приложение демонстрирует принципы объектно-ориентированного программирования, работу с пользовательским вводом, файловыми операциями, регулярными выражениями и элементами графического интерфейса Qt.

5 Список использованных источников

1. Qt Documentation. Qt 6 Documentation. URL: https://doc.qt.io/qt-6/ (дата обращения: 30.11.2025).

2. Qt Documentation. QRegularExpression Class. URL: https://doc.qt.io/qt-6/qregularexpression.html (дата обращения: 04.12.2025).

3. cppreference.com. C++ Standard Library Reference. URL: https://en.cppreference.com/ (дата обращения: 02.12.2025).

Приложения

A Исходный код программы

A.1 Файл Contact.h

```
#pragma once

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

class Contact{
private:
    QString lastName;
    QString firstName;
    QString middleName;
    QString address;
    QDate birthDate;
    QString email;
    QStringList phones;

public:
    Contact();

    QString getLastName() const;
    QString getFirstName() const;
    QString getMiddleName() const;
    QString getAddress() const;
    QDate getBirthDate() const;
    QString getEmail() const;
    QStringList getPhones() const;

    void setLastName(const QString &value);
    void setFirstName(const QString &value);
    void setMiddleName(const QString &value);
    void setAddress(const QString &value);
    void setBirthDate(const QDate &value);
    void setEmail(const QString &value);
    void setPhones(const QStringList &value);

    QString toString() const;
    static Contact fromString(const QString &str);
};
```

Листинг 1: Contact.h

A.2 Файл Contact.cpp

```cpp
#include "headers/Contact.h"

Contact::Contact() {
    birthDate = QDate::currentDate();
}

QString Contact::getLastName() const { return lastName; }
QString Contact::getFirstName() const { return firstName; }
QString Contact::getMiddleName() const { return middleName; }
QString Contact::getAddress() const { return address; }
QDate Contact::getBirthDate() const { return birthDate; }
QString Contact::getEmail() const { return email; }
QStringList Contact::getPhones() const { return phones; }

void Contact::setLastName(const QString &value) { lastName = value; }
void Contact::setFirstName(const QString &value) { firstName = value; }
void Contact::setMiddleName(const QString &value) { middleName = value; }
void Contact::setAddress(const QString &value) { address = value; }
void Contact::setBirthDate(const QDate &value) { birthDate = value; }
void Contact::setEmail(const QString &value) { email = value; }
void Contact::setPhones(const QStringList &value) { phones = value; }

QString Contact::toString() const {
    return lastName + "|" + firstName + "|" + middleName + "|" +
           address + "|" + birthDate.toString("yyyy-MM-dd") + "|" +
           email + "|" + phones.join(",");
}

Contact Contact::fromString(const QString &str) {
    Contact contact;
    QStringList parts = str.split('|');

    if (parts.size() == 7) {
        contact.setLastName(parts[0]);
        contact.setFirstName(parts[1]);
        contact.setMiddleName(parts[2]);
        contact.setAddress(parts[3]);
        contact.setBirthDate(QDate::fromString(parts[4], "yyyy-MM-dd"));
        contact.setEmail(parts[5]);
        contact.setPhones(parts[6].split(','));
    }

    return contact;
```

```
}
```

Листинг 2: Contact.cpp

A.3 Файл ContactDialog.h

```
#pragma once

#include <QDialog>
#include <QLineEdit>
#include <QDateEdit>
#include "Contact.h"

class ContactDialog : public QDialog{
    Q_OBJECT

private:
    QLineEdit *lastNameEdit;
    QLineEdit *firstNameEdit;
    QLineEdit *middleNameEdit;
    QLineEdit *addressEdit;
    QDateEdit *birthDateEdit;
    QLineEdit *emailEdit;
    QLineEdit *phonesEdit;

    bool validateName(const QString &name, const QString &
       fieldName);
    bool validateEmail(const QString &email);
    bool validatePhones(const QString &phones);
    bool validateAddress(const QString &address);
    QString normalizePhone(const QString &phone) const;

private slots:
    void validateAndAccept();

public:
    ContactDialog(QWidget *parent = nullptr);
    Contact getContact() const;
    void setContact(const Contact &contact);
};
```

Листинг 3: ContactDialog.h

A.4 Файл ContactDialog.cpp

```
#include "headers/ContactDialog.h"
#include <QFormLayout>
#include <QHBoxLayout>
#include <QPushButton>
```

```
#include <QMessageBox>
#include <QRegularExpression>
#include <QDate>

ContactDialog::ContactDialog(QWidget *parent) : QDialog(parent){
    setWindowTitle("Add/Edit Contact");
    setMinimumWidth(400);

    QFormLayout *layout = new QFormLayout(this);

    lastNameEdit = new QLineEdit(this);
    firstNameEdit = new QLineEdit(this);
    middleNameEdit = new QLineEdit(this);
    addressEdit = new QLineEdit(this);
    birthDateEdit = new QDateEdit(this);
    birthDateEdit->setCalendarPopup(true);
    birthDateEdit->setDate(QDate::currentDate().addYears(-20));
    birthDateEdit->setMaximumDate(QDate::currentDate().addDays
       (-1));

    emailEdit = new QLineEdit(this);
    phonesEdit = new QLineEdit(this);
    phonesEdit->setPlaceholderText("Enter phones separated by
       commas");

    layout->addRow("Last name:", lastNameEdit);
    layout->addRow("First name:", firstNameEdit);
    layout->addRow("Middle name:", middleNameEdit);
    layout->addRow("Address:", addressEdit);
    layout->addRow("Birth date:", birthDateEdit);
    layout->addRow("Email:", emailEdit);
    layout->addRow("Phones:", phonesEdit);

    QHBoxLayout *buttonLayout = new QHBoxLayout();
    QPushButton *okButton = new QPushButton("OK", this);
    QPushButton *cancelButton = new QPushButton("Cancel", this);

    buttonLayout->addWidget(okButton);
    buttonLayout->addWidget(cancelButton);
    layout->addRow(buttonLayout);

    connect(okButton, &QPushButton::clicked, this, &ContactDialog
       ::validateAndAccept);
    connect(cancelButton, &QPushButton::clicked, this, &QDialog::
       reject);
}

bool ContactDialog::validateName(const QString &name, const
   QString &fieldName){
    QString trimmed = name.trimmed();

    if (trimmed.isEmpty()){
```

```
52          QMessageBox::warning(this, "Error", fieldName + " cannot
               be empty");
53          return false;
54      }
55
56      if (!trimmed[0].isUpper()){
57          QMessageBox::warning(this, "Error", fieldName + " must
               start with an uppercase letter");
58          return false;
59      }
60
61      if (trimmed.startsWith('-') || trimmed.endsWith('-')){
62          QMessageBox::warning(this, "Error", fieldName + " cannot
               start or end with a hyphen");
63          return false;
64      }
65
66      QRegularExpression nameRegex("^[\\p{L}\\d\\s-]+$");
67      if (!nameRegex.match(trimmed).hasMatch()){
68          QMessageBox::warning(this, "Error", fieldName + " may
               contain only letters, digits, hyphen and space");
69          return false;
70      }
71
72      QString normalized = trimmed.left(1).toUpper() + trimmed.mid
           (1).toLower();
73      if (normalized != trimmed){
74          QMessageBox::warning(this, "Error", fieldName + " must be
                in the format: First letter uppercase, the rest
               lowercase");
75          return false;
76      }
77      return true;
78  }
79
80  bool ContactDialog::validateEmail(const QString &email){
81      QString trimmed = email.trimmed();
82      trimmed.remove(' ');
83
84      if (trimmed.isEmpty()){
85          QMessageBox::warning(this, "Error", "Email cannot be
               empty");
86          return false;
87      }
88
89      QRegularExpression emailRegex("^[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[
           a-zA-Z0-9]+$");
90      if (!emailRegex.match(trimmed).hasMatch()){
91          QMessageBox::warning(this, "Error", "Invalid email format
               ");
92          return false;
93      }
```

```
94      return true;
95  }
96
97  QString ContactDialog::normalizePhone(const QString &phone) const
        {
98      QString result;
99      for (QChar c : phone){
100         if (c.isDigit() || c == '+'){
101             result += c;
102         }
103     }
104     return result;
105 }
106
107 bool ContactDialog::validatePhones(const QString &phones){
108     if (phones.trimmed().isEmpty()){
109         QMessageBox::warning(this, "Error", "At least one phone
                number is required");
110         return false;
111     }
112
113     if (phones.contains('(') || phones.contains(')')){
114         QMessageBox::warning(this, "Error", "Invalid input, phone
                must not contain parentheses");
115         return false;
116     }
117
118     QStringList phoneList = phones.split(',');
119     for (const QString &phone : phoneList){
120         QString normalized = normalizePhone(phone.trimmed());
121         if (normalized.length() < 10){
122             QMessageBox::warning(this, "Error", "Phone number is
                    too short: " + phone);
123             return false;
124         }
125     }
126     return true;
127 }
128
129 bool ContactDialog::validateAddress(const QString &address){
130     QString trimmed = address.trimmed();
131
132     if (trimmed.contains('|')){
133         QMessageBox::warning(this, "Error", "Address cannot
                contain the '|' character");
134         return false;
135     }
136
137     return true;
138 }
139
140 void ContactDialog::validateAndAccept(){
```

```
141     if (!validateName(lastNameEdit->text(), "Last name")) return;
142     if (!validateName(firstNameEdit->text(), "First name"))
            return;
143     if (!middleNameEdit->text().trimmed().isEmpty() && !
            validateName(middleNameEdit->text(), "Middle name"))
            return;
144     if (!validateAddress(addressEdit->text())) return;
145     if (!validateEmail(emailEdit->text())) return;
146     if (!validatePhones(phonesEdit->text())) return;
147
148     accept();
149 }
150
151 Contact ContactDialog::getContact() const{
152     Contact contact;
153     contact.setLastName(lastNameEdit->text().trimmed());
154     contact.setFirstName(firstNameEdit->text().trimmed());
155     contact.setMiddleName(middleNameEdit->text().trimmed());
156     contact.setAddress(addressEdit->text().trimmed());
157     contact.setBirthDate(birthDateEdit->date());
158
159     QString email = emailEdit->text().trimmed();
160     email.remove(' ');
161     contact.setEmail(email);
162
163     QStringList phoneList = phonesEdit->text().split(',');
164     QStringList normalizedPhones;
165     for (const QString &phone : phoneList){
166         normalizedPhones.append(normalizePhone(phone.trimmed()));
167     }
168     contact.setPhones(normalizedPhones);
169
170     return contact;
171 }
172
173 void ContactDialog::setContact(const Contact &contact){
174     lastNameEdit->setText(contact.getLastName());
175     firstNameEdit->setText(contact.getFirstName());
176     middleNameEdit->setText(contact.getMiddleName());
177     addressEdit->setText(contact.getAddress());
178     birthDateEdit->setDate(contact.getBirthDate());
179     emailEdit->setText(contact.getEmail());
180     phonesEdit->setText(contact.getPhones().join(", "));
181 }
```

Листинг 4: ContactDialog.cpp

A.5 Файл PhoneBook.h

```
1 #pragma once
2
```

```
#include <QWidget>
#include <QTableWidget>
#include <QLineEdit>
#include <QList>
#include "Contact.h"

class PhoneBook : public QWidget {
    Q_OBJECT

private:
    QTableWidget *table;
    QLineEdit *searchEdit;
    QList<Contact> contacts;
    QString filename;

    void updateTable();

private slots:
    void addContact();
    void editContact();
    void deleteContact();
    void searchByColumn(int column);
    void saveToFile();
    void loadFromFile();

public:
    PhoneBook(QWidget *parent = nullptr);
};
```

Листинг 5: PhoneBook.h

A.6 Файл PhoneBook.cpp

```
#include "headers/PhoneBook.h"
#include "headers/ContactDialog.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QMessageBox>
#include <QFile>
#include <QTextStream>
#include <QHeaderView>
#include <QDir>
#include <QComboBox>

PhoneBook::PhoneBook(QWidget *parent) : QWidget(parent){
    filename = QDir::homePath() + "/phonebook.txt";

    setWindowTitle("Phone Book");
    setMinimumSize(800, 600);
```

```
19
20      QVBoxLayout *mainLayout = new QVBoxLayout(this);
21
22      QHBoxLayout *searchLayout = new QHBoxLayout();
23      QLabel *searchLabel = new QLabel("Search:", this);
24      searchEdit = new QLineEdit(this);
25
26      QComboBox *fieldCombo = new QComboBox(this);
27      fieldCombo->addItem("Last name", 0);
28      fieldCombo->addItem("First name", 1);
29      fieldCombo->addItem("Middle name", 2);
30      fieldCombo->addItem("Address", 3);
31      fieldCombo->addItem("Birth date", 4);
32      fieldCombo->addItem("Email", 5);
33      fieldCombo->addItem("Phones", 6);
34
35      searchLayout->addWidget(searchLabel);
36      searchLayout->addWidget(searchEdit);
37      searchLayout->addWidget(fieldCombo);
38      mainLayout->addLayout(searchLayout);
39
40      connect(searchEdit, &QLineEdit::textChanged, this, [this,
           fieldCombo](){
41          int column = fieldCombo->currentData().toInt();
42          searchByColumn(column);
43      });
44
45      table = new QTableWidget(this);
46      table->setSelectionMode(QAbstractItemView::SingleSelection);
47      table->setColumnCount(7);
48      QStringList headers;
49      headers << "Last name" << "First name" << "Middle name" << "
           Address"
50              << "Birth date" << "Email" << "Phones";
51      table->setHorizontalHeaderLabels(headers);
52      table->horizontalHeader()->setStretchLastSection(true);
53      table->setSelectionBehavior(QAbstractItemView::SelectRows);
54      table->setSortingEnabled(true);
55      mainLayout->addWidget(table);
56
57      QHBoxLayout *buttonLayout = new QHBoxLayout();
58      QPushButton *addButton = new QPushButton("Add", this);
59      QPushButton *editButton = new QPushButton("Edit", this);
60      QPushButton *deleteButton = new QPushButton("Delete", this);
61      QPushButton *saveButton = new QPushButton("Save", this);
62      QPushButton *loadButton = new QPushButton("Load", this);
63
64      buttonLayout->addWidget(addButton);
65      buttonLayout->addWidget(editButton);
66      buttonLayout->addWidget(deleteButton);
67      buttonLayout->addWidget(saveButton);
68      buttonLayout->addWidget(loadButton);
```

```
69      mainLayout->addLayout(buttonLayout);
70
71      connect(addButton, &QPushButton::clicked, this, &PhoneBook::
            addContact);
72      connect(editButton, &QPushButton::clicked, this, &PhoneBook::
            editContact);
73      connect(deleteButton, &QPushButton::clicked, this, &PhoneBook
            ::deleteContact);
74      connect(saveButton, &QPushButton::clicked, this, &PhoneBook::
            saveToFile);
75      connect(loadButton, &QPushButton::clicked, this, &PhoneBook::
            loadFromFile);
76
77      loadFromFile();
78  }
79
80  void PhoneBook::addContact(){
81      ContactDialog dialog(this);
82      if (dialog.exec() == QDialog::Accepted){
83          Contact contact = dialog.getContact();
84          contacts.append(contact);
85          updateTable();
86      }
87  }
88
89  void PhoneBook::editContact(){
90      int row = table->currentRow();
91      if (row < 0){
92          QMessageBox::warning(this, "Error", "Select a contact to
                edit");
93          return;
94      }
95
96      int index = table->item(row, 0)->data(Qt::UserRole).toInt();
97      ContactDialog dialog(this);
98      dialog.setContact(contacts[index]);
99
100     if (dialog.exec() == QDialog::Accepted){
101         contacts[index] = dialog.getContact();
102         updateTable();
103     }
104 }
105
106 void PhoneBook::deleteContact(){
107     int row = table->currentRow();
108     if (row < 0){
109         QMessageBox::warning(this, "Error", "Select a contact to
                delete");
110         return;
111     }
112
113     QMessageBox::StandardButton reply;
```

```
114     reply = QMessageBox::question(this, "Confirmation",
115                                   "Delete selected contact?",
116                                   QMessageBox::Yes | QMessageBox
                                          ::No);
117 
118     if (reply == QMessageBox::Yes){
119         int index = table->item(row, 0)->data(Qt::UserRole).toInt
                ();
120         contacts.removeAt(index);
121         updateTable();
122     }
123 }
124 
125 void PhoneBook::updateTable(){
126     table->setSortingEnabled(false);
127     table->setRowCount(contacts.size());
128 
129     for (int i = 0; i < contacts.size(); i++){
130         const Contact &c = contacts[i];
131         QTableWidgetItem *lastNameItem = new QTableWidgetItem(c.
                getLastName());
132         lastNameItem->setData(Qt::UserRole, i);
133         table->setItem(i, 0, lastNameItem);
134 
135         table->setItem(i, 1, new QTableWidgetItem(c.getFirstName
                ()));
136         table->setItem(i, 2, new QTableWidgetItem(c.getMiddleName
                ()));
137         table->setItem(i, 3, new QTableWidgetItem(c.getAddress())
                );
138         table->setItem(i, 4, new QTableWidgetItem(c.getBirthDate
                ().toString("dd.MM.yyyy")));
139         table->setItem(i, 5, new QTableWidgetItem(c.getEmail()));
140         table->setItem(i, 6, new QTableWidgetItem(c.getPhones().
                join(", ")));
141     }
142     table->setSortingEnabled(true);
143 }
144 
145 void PhoneBook::saveToFile(){
146     QFile file(filename);
147     if (!file.open(QIODevice::WriteOnly | QIODevice::Text)){
148         QMessageBox::warning(this, "Error", "Failed to open file
                for writing");
149         return;
150     }
151 
152     QTextStream out(&file);
153     out.setEncoding(QStringConverter::Utf8);
154 
155     for (const Contact &c : contacts){
156         out << c.toString() << "\n";
```

```
157     }
158
159     file.close();
160     QMessageBox::information(this, "Success", "Data saved
            successfully");
161 }
162
163 void PhoneBook::loadFromFile(){
164     QFile file(filename);
165     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)){
166         return;
167     }
168
169     contacts.clear();
170     QTextStream in(&file);
171     in.setEncoding(QStringConverter::Utf8);
172
173     while (!in.atEnd()){
174         QString line = in.readLine();
175         Contact contact = Contact::fromString(line);
176         if (!contact.getLastName().isEmpty()){
177             contacts.append(contact);
178         }
179     }
180
181     file.close();
182     updateTable();
183 }
184
185 void PhoneBook::searchByColumn(int column){
186     QString text = searchEdit->text().trimmed().toLower();
187
188     if (text.isEmpty()){
189         for (int i = 0; i < table->rowCount(); i++)
190             table->setRowHidden(i, false);
191         return;
192     }
193
194     for (int i = 0; i < table->rowCount(); i++){
195         bool found = false;
196
197         if (column >= 0 && column < 6){
198             if (table->item(i, column) &&
199                 table->item(i, column)->text().toLower().contains
                        (text)){
200                 found = true;
201             }
202         } else {
203             if (table->item(i, 6)){
204                 QString phones = table->item(i, 6)->text().
                        toLower();
```

```
205                 QStringList phoneList = phones.split(",", Qt::
                        SkipEmptyParts);
206
207                 for (const QString &phone : phoneList){
208                     if (phone.trimmed().contains(text)){
209                         found = true;
210                         break;
211                     }
212                 }
213             }
214         }
215         table->setRowHidden(i, !found);
216     }
217 }
```

Листинг 6: PhoneBook.cpp

A.7 Файл main.cpp

```
1  #include <QApplication>
2  #include "headers/PhoneBook.h"
3
4  int main(int argc, char *argv[]){
5      QApplication app(argc, argv);
6
7      PhoneBook phoneBook;
8      phoneBook.show();
9
10     return app.exec();
11 }
```

Листинг 7: main.cpp