

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 ЛАБОРАТОРНОЙ РАБОТЕ №8

по дисциплине

«Объектно-ориентированное программирование»

Приложение «Телефонный справочник»

Студент,
группы 5130201/40003

_____ Четвергов И.С

Доцент

_____ Глазунов В.В

«_____» _____ 20 ____ г.

Санкт-Петербург, 2025

Содержание

| | |
|--|-----------|
| Введение | 4 |
| 1 Постановка задачи | 6 |
| 1.1 Цель работы | 6 |
| 1.2 Требования к хранимым данным | 6 |
| 1.3 Функциональные требования | 6 |
| 1.3.1 Подзадача 1: Добавление нового контакта | 7 |
| 1.3.2 Подзадача 2: Редактирование существующего контакта | 7 |
| 1.3.3 Подзадача 3: Удаление контакта | 8 |
| 1.3.4 Подзадача 4: Поиск и фильтрация контактов | 8 |
| 1.3.5 Подзадача 5: Сортировка данных | 8 |
| 1.3.6 Подзадача 6: Сохранение и загрузка данных | 9 |
| 1.4 Требования к валидации данных | 9 |
| 1.4.1 Фамилия, Имя, Отчество | 9 |
| 1.4.2 Телефонный номер | 10 |
| 1.4.3 Дата рождения | 10 |
| 1.4.4 Email | 10 |
| 1.5 Требования к пользовательскому интерфейсу | 11 |
| 1.6 Архитектурные требования | 11 |
| 2 Реализация | 12 |
| 2.1 Архитектура приложения | 12 |
| 2.2 Описание классов | 13 |
| 2.2.1 Структура Contact | 13 |
| 2.2.2 Класс ContactManager | 13 |
| 2.2.3 Класс ContactTableModel | 14 |
| 2.2.4 Класс Validator | 16 |
| 2.2.5 Класс MainWindow | 18 |
| 2.3 Механизм сигналов и слотов | 19 |
| 2.4 Класс QDataWidgetMapper | 20 |
| 2.5 Класс QSortFilterProxyModel | 21 |
| 2.6 Алгоритм добавления и редактирования контакта | 21 |
| 2.7 Алгоритм поиска и фильтрации | 22 |
| 2.8 Сохранение и загрузка данных | 23 |
| 3 Тестирование приложения | 26 |
| 3.1 Начальное состояние приложения | 26 |
| 3.2 Сценарий 1: Добавление нового контакта | 27 |
| 3.3 Сценарий 2: Валидация некорректных данных | 28 |
| 3.3.1 Пример 1: Некорректное имя | 28 |
| 3.3.2 Пример 2: Некорректный email | 28 |
| 3.3.3 Пример 3: Множественные ошибки | 29 |

| | | |
|-------------------|---|-----------|
| 3.4 | Сценарий 3: Редактирование существующего контакта | 29 |
| 3.5 | Сценарий 4: Управление телефонными номерами | 31 |
| 3.5.1 | Переключение между типами номеров | 32 |
| 3.6 | Сценарий 5: Удаление контакта | 32 |
| 3.7 | Сценарий 6: Поиск контактов | 33 |
| 3.7.1 | Поиск по фамилии | 33 |
| 3.7.2 | Поиск по другим полям | 34 |
| 3.7.3 | Очистка поиска | 34 |
| 3.8 | Сценарий 7: Сортировка данных | 34 |
| 3.8.1 | Сортировка по фамилии | 34 |
| 3.8.2 | Сортировка по дате рождения | 34 |
| 3.9 | Сценарий 8: Работа с файлами | 35 |
| 3.9.1 | Сохранение в файл | 35 |
| 3.9.2 | Загрузка из файла | 36 |
| 3.10 | Сценарий 9: Использование календаря для выбора даты | 36 |
| 3.11 | Тестирование регулярных выражений | 37 |
| 3.12 | Результаты тестирования | 38 |
| 4 | Заключение | 39 |
| 4.1 | Реализованная функциональность | 39 |
| 4.2 | Использованные технологии и концепции | 40 |
| 5 | Список использованных источников | 41 |
| Приложение | | 42 |
| Приложение А. | Файл сборки CMakeLists.txt | 42 |
| Приложение Б. | Заголовочные файлы | 43 |
| Приложение В. | Исходные файлы | 45 |
| Приложение Д. | Заголовочный файл MainWindow | 51 |
| Приложение Е. | Исходные файлы MainWindow | 52 |
| Приложение Ж. | Точка входа программы | 62 |
| Приложение З. | Пример файла данных | 62 |

Введение

В современном мире организация и управление контактными данными является важной задачей как для личного использования, так и для корпоративных систем. Телефонные справочники — одно из базовых приложений, демонстрирующих принципы работы с данными, их хранением, обработкой и представлением пользователю.

Данная лабораторная работа посвящена разработке приложения «Телефонный справочник» с использованием фреймворка **Qt** — мощной кроссплатформенной библиотеки для создания графических пользовательских интерфейсов на языке C++.

Фреймворк Qt

Qt — это полнофункциональный фреймворк для разработки кроссплатформенных приложений с графическим интерфейсом. Набор инструментов QT включает в себя:

- **Модуль Qt Widgets** — набор готовых виджетов;
- **Систему сигналов и слотов** — механизм межобъектного взаимодействия, позволяющий связывать события (сигналы) с обработчиками (слотами);
- **Архитектуру Model/View** — паттерн проектирования, разделяющий данные (модель) и их представление (вид);
- **Систему метаобъектов (Meta-Object System)** - основа для расширений C++, таких как сигналы и слоты, информация о типе в реальном времени и динамические свойства;
- **Встроенные средства работы с JSON** — классы для работы с файловой системой (**QFile**), сериализацией данных (**QJsonDocument**), регулярными выражениями (**QRegularExpression**) и др.

Предметная область

Приложение «Телефонный справочник» предназначено для управления контактной информацией о людях. Каждый контакт включает:

- Персональные данные: имя, фамилию, отчество;
- Адрес проживания;
- Дату рождения;

- Электронную почту;
- Набор телефонных номеров различных типов (рабочий, домашний, служебный).

Приложение должно обеспечивать:

- Добавление, редактирование и удаление записей;
- Валидацию пользовательского ввода с помощью регулярных выражений;
- Поиск и сортировку записей;
- Сохранение и загрузку данных в файл формата JSON;
- Удобный табличный интерфейс для просмотра данных.

Цель работы

Целью данной работы является изучение принципов разработки desktop-приложений на Qt с использованием архитектуры Model/View, освоение механизма сигналов и слотов, работы с табличными представлениями данных, валидации пользовательского ввода и сериализации данных в формат JSON.

1 Постановка задачи

1.1 Цель работы

Целью данной работы является разработка desktop-приложения «Телефонный справочник» на базе фреймворка Qt 6 с использованием архитектуры Model/View. В процессе реализации необходимо:

- освоить механизм сигналов и слотов Qt;
- изучить работу с табличными представлениями данных через QTableView;
- реализовать валидацию пользовательского ввода с помощью регулярных выражений;
- освоить сериализацию и десериализацию данных в формат JSON.

1.2 Требования к хранимым данным

Приложение должно обеспечивать хранение контактной информации о людях. Каждая запись контакта представляет собой структуру данных со следующими полями:

- **Имя** — строка типа `QString`, содержащая имя контакта;
- **Фамилия** — строка типа `QString`, содержащая фамилию контакта;
- **Отчество** — строка типа `QString` (может быть пустой);
- **Адрес** — строка типа `QString` с адресом проживания;
- **Дата рождения** — тип `QDate`, дата в формате ISO 8601;
- **Email** — строка типа `QString`, адрес электронной почты;
- **Телефонные номера** — ассоциативный контейнер типа `QMap<QString, QString>`, где ключ — тип номера (рабочий, домашний, служебный), значение — телефонный номер в виде строки.

1.3 Функциональные требования

Приложение должно реализовывать следующие подзадачи:

1.3.1 Подзадача 1: Добавление нового контакта

Описание: Пользователь должен иметь возможность добавить новый контакт в справочник.

Требования к реализации:

- Форма ввода должна содержать поля для обязательных данных контакта;
- Для ввода даты рождения использовать виджет `QDateEdit` с возможностью выбора через всплывающий календарь `QCalendarWidget`;
- Для выбора типа телефонного номера использовать виджет `QComboBox` с предустановленными значениями: «Рабочий», «Домашний», «Служебный»;
- Перед сохранением контакта необходимо выполнить валидацию всех полей согласно правилам, описанным в разделе 1.4;
- В случае ошибок валидации вывести сообщение с указанием проблемных полей через диалоговое окно `QMessageBox`;
- После успешного добавления контакт должен появиться в табличном представлении;
- Данные должны автоматически сохраняться в файл формата JSON.

1.3.2 Подзадача 2: Редактирование существующего контакта

Описание: Пользователь должен иметь возможность изменить данные выбранного контакта.

Требования к реализации:

- При выборе строки в таблице данные контакта должны автоматически загружаться в форму ввода при помощи `QDataWidgetMapper`;
- Кнопка «Редактировать» должна переводить приложение в специальный режим «редактирования»;
- В режиме редактирования таблица должна стать неактивной (запрещён выбор других строк);
- Кнопка «Добавить» должна изменить текст на «Сохранить»;
- Должна появиться кнопка «Отмена» для отката изменений;
- После сохранения изменения должны отразиться в таблице и автоматически записаться в файл JSON.

1.3.3 Подзадача 3: Удаление контакта

Описание: Пользователь должен иметь возможность удалить выбранный контакт из справочника.

Требования к реализации:

- Кнопка «Удалить» должна быть активна только при выбранной строке в таблице;
- После удаления контакт должен исчезнуть из табличного представления;
- Изменения должны автоматически сохраняться в файл JSON;
- Поля формы должны очиститься.

1.3.4 Подзадача 4: Поиск и фильтрация контактов

Описание: Пользователь должен иметь возможность найти контакт по введённому запросу.

Требования к реализации:

- Использовать виджет QLineEdit для ввода поискового запроса;
- Поиск должен выполняться по всем полям таблицы одновременно (имя, фамилия, отчество, адрес, email, телефон);
- Фильтрация должна происходить в реальном времени при вводе текста;
- Для реализации использовать класс QSortFilterProxyModel, обрабатывающий основную модель данных;
- Поиск должен быть регистронезависимым;
- При очистке поля поиска должны отображаться все контакты.

1.3.5 Подзадача 5: Сортировка данных

Описание: Пользователь должен иметь возможность упорядочить контакты по любому полю.

Требования к реализации:

- Сортировка должна выполняться через клик по заголовку столбца в таблице;
- Использовать встроенный механизм сортировки QTableView с активированным флагом setSortingEnabled(true);
- Повторный клик по заголовку столбца должен менять направление сортировки (возрастание/убывание).

1.3.6 Подзадача 6: Сохранение и загрузка данных

Описание: Данные справочника должны сохраняться на локальное устройство и загружаться при запуске приложения.

Требования к реализации:

- Формат хранения данных: JSON;
- Для работы с файловой системой использовать класс `QFile`;
- Для сериализации и десериализации использовать классы `QJsonDocument`, `QJsonObject`, `QJsonArray`;
- Имя файла по умолчанию: `contacts.json`;
- При запуске приложения должна автоматически выполняться попытка загрузки данных из файла;
- Пользователь должен иметь возможность сохранить данные в произвольный файл через диалоговое окно `QFileDialog::getSaveFileName()`;
- Пользователь должен иметь возможность загрузить данные из произвольного файла через диалоговое окно `QFileDialog::getOpenFileName()`;
- После каждой операции добавления, редактирования или удаления данные должны автоматически сохраняться в файл по умолчанию.

1.4 Требования к валидации данных

Все вводимые пользователем данные должны проходить проверку на соответствие заданным правилам с использованием регулярных выражений (класс `QRegularExpression`):

1.4.1 Фамилия, Имя, Отчество

- Длина строки: от 2 до 49 символов (после удаления пробелов по краям);
- Первый символ: заглавная буква кириллицы или латиницы;
- Допустимые символы: буквы ф, пробелы, дефисы, цифры;
- Стока не может начинаться или заканчиваться дефисом;
- Последний символ: буква или цифра;
- Пробелы в начале и конце строки должны автоматически удаляться методом `QString::trimmed()`;
- Регулярное выражение: `^ [А-ЯА-З] [А-Яа-яА-За-з] [-]* [А-Яа-яА-За-з0-9]$`

1.4.2 Телефонный номер

- Формат: международный (с символом +) или национальный;
- Допустимые символы: цифры, символ +, пробелы, круглые скобки, дефисы;
- Примеры допустимых форматов:
 - +7 812 1234567
 - 8(800)123-1212
 - +38 (032) 123 11 11
- Регулярное выражение: `^(\+\+)?[\s\d()]-]+$`

1.4.3 Дата рождения

- Дата должна быть валидной (проверка через `QDate::isValid()`);
- Дата должна быть строго меньше текущей даты;
- Проверка выполняется сравнением с `QDate::currentDate()`;
- Для ввода используется виджет `QDateEdit` с форматом отображения dd.MM.yyyy;
- Виджет должен иметь всплывающий календарь (`setCalendarPopup(true)`).

1.4.4 Email

- Формат: `username@domain.zone`;
- Имя пользователя: одна или более латинских букв или цифр;
- Обязательное наличие символа @;
- Доменное имя: одна или более латинских букв или цифр;
- Обязательное наличие точки после домена;
- Зона домена: одна или более латинских букв или цифр;
- Пробелы в начале и конце строки должны автоматически удаляться;
- Регулярное выражение: `^ [a-zA-Z0-9] +@[a-zA-Z0-9] +\.\. [a-zA-Z0-9] +$`

1.5 Требования к пользовательскому интерфейсу

- Для табличного представления данных использовать виджет `QTableView`;
- Таблица должна поддерживать автоматическую сортировку по столбцам;
- Режим выбора: только одна строка одновременно (`SingleSelection`);
- Столбцы таблицы должны автоматически растягиваться на всю ширину окна (`QHeaderView::Stretch`);
- Форма ввода/редактирования должна располагаться в верхней части окна и содержать все необходимые поля;
- Кнопки управления (Добавить, Удалить, Редактировать) должны располагаться под формой ввода;
- Состояние кнопок (активность, видимость) должно зависеть от текущего контекста (выбрана ли строка, режим редактирования и т.д.).

1.6 Архитектурные требования

- Архитектурный паттерн: Model/View/Controller (MVC);
- Разделение ответственности:
 - Model (модель данных) — классы для хранения и управления данными;
 - View (представление) — виджеты Qt для отображения данных;
 - Controller (контроллер) — методы-слоты для обработки пользовательских действий;
- Для связи модели данных с табличным представлением использовать наследование от `QAbstractTableModel`;
- Для автоматической синхронизации формы ввода с выбранной строкой таблицы использовать класс `QDataWidgetMapper`;
- Для реализации фильтрации и сортировки без изменения исходных данных использовать класс `QSortFilterProxyModel`;
- Валидация данных должна быть вынесена в отдельный утилитарный класс со статическими методами.

2 Реализация

2.1 Архитектура приложения

Приложение построено на основе архитектурного паттерна **Model/View/Controller (MVC)**, который обеспечивает разделение ответственности между компонентами системы:

- **Model (Модель)** — классы `Contact`, `ContactManager`, `ContactTableModel`, отвечающие за хранение и управление данными;
- **View (Представление)** — класс `MainWindow` и используемые виджеты Qt (`QTableView`, `QLineEdit`, `QPushButton` и др.), отвечающие за отображение данных пользователю;
- **Controller (Контроллер)** — методы-слоты класса `MainWindow`, обрабатывающие пользовательский ввод и взаимодействие с моделью.

Дополнительно используется класс `Validator` для валидации данных и `QSortFilterProxyModel` для фильтрации и сортировки данных в таблице.

Структура директорий проекта:

```
1 lab8/
2 |-- CMakeLists.txt
3 |-- main.cc
4 |-- core/
5 |   |-- include/
6 |   |   |-- Contact.h
7 |   |   |-- ContactManager.h
8 |   |   |-- ContactTableModel.h
9 |   |   \-- Validator.h
10 |  \-- src/
11 |    |-- ContactManager.cc
12 |    |-- ContactTableModel.cc
13 |    \-- Validator.cc
14 |-- ui/
15 |   |-- include/
16 |   |   \-- MainWindow.h
17 |   \-- src/
18 |     |-- MainWindow.cc
19 |     |-- MainWindowUI.cc
20 |     \-- MainWindowLogic.cc
21 \-- data/
22   \-- contacts.json
```

Листинг 1: Структура директорий.

Структурная схема взаимодействия компонентов представлена на рис. 1.

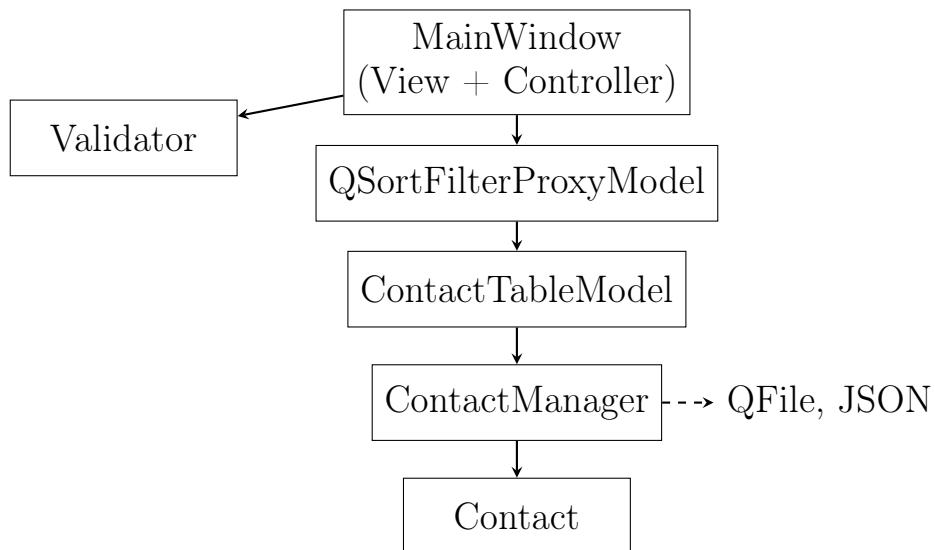


Рис. 1: Архитектура приложения

2.2 Описание классов

2.2.1 Структура Contact

Contact - структура, отвечающая за хранение одной записи контакта.

```
1 /* служит моделью QTable
2 struct Contact {
3     QString firstName_;
4     QString lastName_;
5     QString middleName_;
6     QString adress_;
7     QDate birthDate_;
8     QString email_;
9     QMap<QString, QString> phoneNumbers_;
10};
```

Листинг 2: Contact

2.2.2 Класс ContactManager

ContactManager — класс для управления коллекцией контактов. Инкапсулирует логику работы с данными и предоставляет интерфейс для работы с данными.

ContactManager() — стандартный конструктор по умолчанию. Его основная роль заключается в инициализации внутренней коллекции **QList<Contact> m_contacts**. Он создает пустой контейнер, готовый к загрузке или добавлению данных.

```

1 class ContactManager {
2 public:
3     // * --- CRUD операции ---
4     void addContact(const Contact& contact);
5     void removeContact(int index);
6     void updateContact(int index, const Contact& contact);
7
8     // * --- геттеры, доступ к данным ---
9     const QList<Contact>& getContacts() const;
10    QList<Contact>& getContactsMutable();
11    int contactCount() const;
12    Contact* getContact(int index);
13
14    // * --- работа с данными ---
15    bool loadFromFile(const QString& filePath);
16    bool saveToFile(const QString& filePath) const;
17
18 private:
19     QList<Contact> m_contacts; // основная коллекция
20 };

```

Листинг 3: ContactManager

2.2.3 Класс ContactTableModel

ContactTableModel — адаптер между ContactManager и QTableView. Наследуется от абстрактного класса QAbstractTableModel, который является частью архитектуры Model/View фреймворка Qt.

Конструктор ContactTableModel принимает указатель на ContactManager как основной источник данных. Его роль — установить неразрывную связь между моделью Qt и ядром приложения.

Конструктор инициализирует:

- указатель contactManager_, который становится единственным источником данных для методов rowCount(), columnCount() и data();
- базовый класс QAbstractTableModel через родительский объект parent.

```

1 class ContactTableModel : public QAbstractTableModel {
2     Q_OBJECT
3
4 public:
5     explicit ContactTableModel(ContactManager* contactManager,
6         QObject* parent = nullptr);
7
8     /* --- обязательные методы QAbstractTableModel ---
9      int rowCount(const QModelIndex& parent = QModelIndex())
10      const override;

```

```

9     int columnCount(const QModelIndex& parent = QModelIndex())
10    const override;
11    QVariant data(const QModelIndex& index, int role =
12      Qt::DisplayRole) const override;
13    QVariant headerData(int section, Qt::Orientation
14      orientation, int role = Qt::DisplayRole) const override;
15    bool setData(const QModelIndex& index, const QVariant&
16      value, int role = Qt::EditRole) override;
17    Qt::ItemFlags flags(const QModelIndex& index) const override;
18
19    /* --- методы-обертки CRUD операций (синхронизация модели)
20     void addContact(const Contact& contact);
21     void removeContact(int index);
22     void updateContact(int index, const Contact& contact);
23
24     const Contact& getContact(int row) const;
25     QString normalizeDigits(const QString& s) const;
26
27 public slots:
28     void resetTable();
29
30 private:
31     ContactManager* contactManager_;
32 };

```

Листинг 4: ContactManager

Механизм синхронизации Model и View:

При изменении данных модель должна уведомить View с помощью сигналов:

- `beginInsertRows()` и `endInsertRows()` — перед и после добавления строк;
- `beginRemoveRows()` и `endRemoveRows()` — перед и после удаления строк;
- `dataChanged()` — при изменении содержимого ячеек.

Эти методы обеспечивают автоматическое обновление `QTableView` при изменении данных в модели.

Метод `data()` является центральным элементом архитектуры Model/View. Он отвечает за предоставление данных из ядра (`ContactManager`) для отображения в представлении (`QTableView`).

Метод `data(index, role):`

`contact ← contactManager_->getContact(index.row())`

Если `contact == NULL` ИЛИ `index` невалиден:

Вернуть `QVariant()`

Конец_Если

Если role == Qt::DisplayRole ИЛИ role == Qt::EditRole:

Выбор по index.column():

- 0: Вернуть contact->firstName_
- 1: Вернуть contact->lastName_
- 2: Вернуть contact->middleName_
- 3: Вернуть contact->adress_
- 4: Вернуть contact->birthDate_
- 5: Вернуть contact->email_
- 6: Вернуть contact->phoneNumbers_ (объединенные запятой)

Конец_Выбора

// роль для поиска в общем(универсальном) формате

Иначе Если role == ContactTableModel::NormalizedPhonesRole:

СтроКаТелефонов ← contact->phoneNumbers_ (объединенные)

Вернуть this->normalizeDigits(СтроКаТелефонов)

Конец_Если

Вернуть QVariant()

2.2.4 Класс Validator

Validator — это утилитарный класс со статическими методами, инкапсулирующий всю проверки пользовательского ввода. Валидация реализована с помощью класса **QRegularExpression**.

```
1 class Validator {
2 public:
3     static bool isValidName(const QString& name);
4     static bool isValidPhoneNumber(const QString& number);
5     static bool isValidEmail(const QString& email);
6     static bool isValidBirthDate(const QDate& date);
7 }
```

Листинг 5: Validator

Используемое подмножество синтаксиса регулярных выражений:

- ^ и \$ — символы, обозначающие начало и конец строки, обеспечивающие проверку всей строки целиком.
- [] — символьный класс, позволяющий указать набор допустимых символов (например, [A-Za-z0-яа-я]).
- * и + — квантификаторы, обозначающие ноль или более (*) и один или более (+) повторений предыдущего элемента.

Ниже представлены спецификации валидаторов для каждого поля:

- **isValidName (Имя, Фамилия, Отчество)**

Регулярное выражение:

`^ [A-ЯА-З] [А-Яа-яА-За-з]*[А-Яа-яА-За-з0-9]$`

Описание: Требует, чтобы строка начиналась с заглавной буквы (русской или латинской) и содержала только буквы, пробелы или дефисы. Дополнительно проверяется, что строка не пуста и не начинается/заканчивается дефисом.

- **isValidPhoneNumber (Телефонный номер)**

Регулярное выражение: `^(\+\?)?[\s\d()]-+$`

Описание: Проверяет, что строка представляет собой допустимый телефонный номер. Допускается необязательный международный код (начинается с + и 1-3 цифры), за которым следует основной номер. Основная часть может содержать цифры, разделенные пробелами, дефисами (-) или точками (.). Это гибкое выражение для общих форматов телефонов.

- **isValidEmail (Email)**

Регулярное выражение:

`^ [a-zA-Z0-9]+@[a-zA-Z0-9]+\.\[a-zA-Z0-9]+$`

Описание: Требует наличия символа @ как минимум одной точки (.) в доменной части. Разрешает латинские буквы, цифры, точки и нижнее подчеркивание в имени пользователя, и латинские буквы, цифры и дефисы в доменном имени. Является упрощенным, но достаточным для большинства случаев.

- **isValidBirthDate (Дата рождения)**

Описание: Валидация даты рождения в приложении осуществляется не регулярным выражением, а через метод `QDate::isValid()` и сравнение с текущей датой. Требуется, чтобы дата была синтаксически корректной и не находилась в будущем.

Метод `isValidName()` реализует валидацию полей. Алгоритм включает предварительную очистку строки от лишних пробелов.

Метод `isValidName(name):`

`trimmedName ← name.trimmed()`

Если `trimmedName.isEmpty():`

 Вернуть Ложь

Конец_Если

Если `trimmedName.startsWith(' - ') ИЛИ trimmedName.endsWith(' - '):`

Вернуть Ложь
Конец_Если

```
Шаблон : QRegularExpression ←  
← ^ [А-ЯА-З] [А-Яа-яА-За-z -]* [А-Яа-яА-За-z0-9] $
```

Вернуть Шаблон.match(trimmedName).isValid()

2.2.5 Класс MainWindow

`MainWindow` — главное окно приложения. Наследуется от `QMainWindow` — базового класса Qt для создания главных окон приложений. Класс `MainWindow` наследуется от `QMainWindow`, а не от базового `QWidget`, что является стандартным архитектурным решением для создания полноценных настольных приложений в Qt.

`QWidget` — это фундаментальный класс, представляющий собой пустой холст (виджет), который не имеет встроенной структуры и требует ручной реализации всей логики компоновки (Layout).

`QMainWindow` является специализированным контейнером, который предоставляет готовую структуру главного окна приложения. Он содержит все стандартные зоны (меню, статус-бар) и автоматически управляет их компоновкой. Ключевым элементом архитектуры является **Central Widget** (Центральный виджет): основная область для размещения контента, где в данном приложении располагаются `QTableView` и элементы управления.

Макрос Q_OBJECT: обязателен для классов, использующих механизм сигналов и слотов. Добавляет метаинформацию о классе, необходимую для работы Meta-Object System.

Конструктор `MainWindow()` является точкой входа для построения всего пользовательского интерфейса и связующим звеном между Model и View.

Конструктор выполняет следующие ключевые действия:

- Инициализирует базовый класс `QMainWindow`.
- Создает основные компоненты: создает и инициализирует `ContactTableModel`, который получает доступ к встроенному члену `ContactManager`, а также создает `QSortFilterProxyModel` и устанавливает его как источник для `QTableView`.
- Инициализирует View: создает все виджеты ввода (`QLineEdit`, `QDateEdit`) и управляющие кнопки (`QPushButton`).
- Настраивает логику: связывает сигналы и слоты (раздел 2.3), вызывает вспомогательные методы `setupUi()` (компоновка виджетов), `setupMapper()` (связь формы и модели).

```

1 class MainWindow : public QMainWindow {
2     Q_OBJECT
3
4 public:
5     explicit MainWindow(QWidget *parent = nullptr);
6     ~MainWindow() = default;
7
8 private slots:
9     // * --- слоты для кнопок (CRUD) ---
10    void onAddButtonClicked();
11    void onRemoveButtonClicked();
12    void onEditButtonClicked();
13    void onSaveButtonClicked();
14    void onLoadButtonClicked();
15    void onCancelButtonClicked();
16
17    // * --- слоты для взаимодействия с таблицей и поиском ---
18    void onSearchTextChanged(const QString& text);
19    void onSelectionChanged();
20    void onPhoneTypeChanged(const QString& type);
21
22 private:
23     // основные компоненты ядра и модели
24     ContactManager contactManager_;
25     ContactTableModel* tableModel_;
26     QSortFilterProxyModel* proxyModel_;
27
28     // компоненты View/Mapper
29     QTableView* tableView_;
30     QDataWidgetMapper* mapper_;
31
32     // состояние
33     bool isInEditMode_ = false;
34     void setEditingMode(bool isInEditMode);
35
36     // вспомогательные методы
37     void setupUi();
38     void setupMapper();
39     Contact getCurrentContactFromForm() const;
40     bool validateContact(const Contact& contact);
41 };

```

Листинг 6: Прототип класса MainWindow

2.3 Механизм сигналов и слотов

Qt использует механизм **сигналов и слотов** для межобъектного взаимодействия. Сигнал — это событие, генерируемое объектом (например, нажатие кнопки), а слот — функция, вызываемая в ответ на сигнал.

Установка всех связей между компонентами View (виджетами) и Controller

(слотами MainWindow) выполняется в конструкторе класса MainWindow. Это гарантирует, что все объекты приложения готовы к взаимодействию сразу после создания главного окна.

Полный перечень установленных связей представлен в Листинге 7:

```
1 // --- Конструктор MainWindow::MainWindow() ---
2
3 // 1. Связи для CRUD-операций и управления файлами (кнопки)
4 connect(addButton_,           &QPushButton::clicked,
5         this,                  &MainWindow::onAddButtonClicked);
6 connect(removeButton_,        &QPushButton::clicked,
7         this,                  &MainWindow::onRemoveButtonClicked);
8 connect(editButton_,          &QPushButton::clicked,
9         this,                  &MainWindow::onEditButtonClicked);
10 connect(cancelButton_,        &QPushButton::clicked,
11        this,                  &MainWindow::onCancelButtonClicked);
12 connect(saveButton_,          &QPushButton::clicked,
13        this,                  &MainWindow::onSaveButtonClicked);
14 connect(loadButton_,          &QPushButton::clicked,
15        this,                  &MainWindow::onLoadButtonClicked);
16
17 // 2. Связи для поиска и взаимодействия с таблицей
18 connect(searchInput_,         &QLineEdit::textChanged,
19         this,                  &MainWindow::onSearchTextChanged);
20 connect(tableView_->selectionModel(),
21             &QItemSelectionModel::currentChanged,
22             this,                  &MainWindow::onSelectionChanged);
23
24 // 3. Связь для выбора типа телефона (обработка перегруженного сигнала)
25 connect(phoneTypeComboBox_,   static_cast<void (QComboBox::*)(const
26                                         QString&)>(&QComboBox::currentTextChanged),
27         this,                  &MainWindow::onPhoneTypeChanged);
```

Листинг 7: Установка всех связей Сигналов и Слотов в MainWindow.

2.4 Класс QDataWidgetMapper

QDataWidgetMapper — класс Qt, автоматически связывающий столбцы модели с виджетами формы. Когда пользователь выбирает строку в таблице, маппер автоматически заполняет поля формы данными из выбранной строки.

Настройка маппера:

```
1 mapper_ = new QDataWidgetMapper(this);
2 mapper_->setModel(proxyModel_);
3 mapper_->setSubmitPolicy(QDataWidgetMapper::ManualSubmit);
4
5 mapper_->addMapping(firstNameInput_, 0);
6 mapper_->addMapping(lastNameInput_, 1);
```

```

7 mapper_->addMapping(middleNameInput_, 2);
8 mapper_->addMapping(addressInput_, 3);
9 mapper_->addMapping(birthDateInput_, 4, "date");
10 mapper_->addMapping(emailInput_, 5);

```

Листинг 8: Настройка QDataWidgetMapper

Режимы отправки данных:

- **AutoSubmit** — изменения применяются сразу при изменении виджета;
- **ManualSubmit** — изменения применяются только при явном вызове `mapper_->submit()`.

В данном приложении используется **ManualSubmit**, чтобы изменения применялись только после нажатия кнопки "Сохранить".

2.5 Класс QSortFilterProxyModel

QSortFilterProxyModel — прокси-модель, которая оборачивает базовую модель и добавляет возможности сортировки и фильтрации без изменения исходных данных.

Использование:

```

1 proxyModel_ = new QSortFilterProxyModel(this);
2 proxyModel_->setSourceModel(tableModel_);
3 proxyModel_->setFilterCaseSensitivity(Qt::CaseInsensitive);
4
5 tableView_->setModel(proxyModel_);
6 tableView_->setSortingEnabled(true);

```

Листинг 9: Настройка прокси-модели

При вводе текста в поле поиска вызывается метод:

```

1 void MainWindow::onSearchTextChanged(const QString& text) {
2     proxyModel_->setFilterRegularExpression(text);
3 }

```

Листинг 10: Фильтрация данных

Прокси-модель автоматически фильтрует строки, содержащие введённый текст, и обновляет представление.

2.6 Алгоритм добавления и редактирования контакта

Кнопка "Добавить" используется в двух режимах:

- **Режим добавления** — создание нового контакта;
- **Режим редактирования** — сохранение изменений существующего контакта.

Метод `onAddButtonClicked()` выполняет двойную функцию, обрабатывая как добавление нового контакта, так и сохранение изменений в существующем (режим редактирования). Логика метода зависит от флага состояния `isEditMode_`. В обоих случаях обязательной является предварительная валидация данных. В режиме редактирования метод также отвечает за ручное обновление поля телефона, поскольку оно не может быть автоматически обработано `QDataWidgetMapper`.

Метод `onAddButtonClicked()`:

```
newContact ← this->getCurrentContactFromForm()
```

Если Не `validateContact(newContact)`:

 Вернуть // (Показывает QMessageBox и прерывает выполнение)

Конец_Если

Если `isEditMode_`: // режим редактирования

```
proxyIndex ← tableView_->currentIndex()
```

```
modelIndex ← proxyModel_->mapToSource(proxyIndex)
```

```
ОбъектКонтакта ← tableView_->getContact(modelIndex.row())
```

// Ручное обновление не-маппированных данных (телефон)

```
ТипТелефона : QString ← phoneTypeComboBox_->currentText()
```

```
НомерТелефона : QString ← mainPhoneInput_->text()
```

```
ОбъектКонтакта.phoneNumbers_.clear()
```

```
ОбъектКонтакта.phoneNumbers_.insert(ТипТелефона, НомерТелефона)
```

Если `mapper_->submit()`:

 Вызвать `QMessageBox.information`

 "Данные успешно отредактированы")

Иначе:

 Вызвать `QMessageBox.warning("Ошибка сохранения")`

Конец_Если

Иначе: // режим добавления

 Вызвать `tableView_->addContact(newContact)`

 Вызвать `QMessageBox.information("Контакт успешно добавлен")`

Конец_Если

Вызвать `contactManager_.saveToFile("contacts.json")`

Вызвать `this->setEditMode(Ложь)`

2.7 Алгоритм поиска и фильтрации

При вводе текста в поле поиска происходит фильтрация отображаемых записей. Используется регулярное выражение для поиска подстроки во всех столбцах таблицы.

Метод `onSearchTextChanged()` реализует поиск по записям. Он демонстрирует адаптивное поведение: если введенный текст состоит только из цифр, прокси-модель переключается на использование **пользовательской роли** `NormalizedPhonesRole` для поиска по неформатированным телефонным номерам. Если же текст содержит буквы, поиск ведется по стандартной роли (`Qt::DisplayRole`) во всех отображаемых столбцах.

Метод `onSearchTextChanged(text):`

```
trimmedText ← text.trimmed()
```

Если `trimmedText.isEmpty()`:

```
proxyModel_->setFilterRegularExpression(QRegularExpression())
```

Вернуть

Конец_Если

```
Цифры : QString ← trimmedText.extractDigits()
```

Если Не Цифры.isEmpty() // режим поиска по телефону

```
proxyModel_->setFilterRole(
```

```
ContactTableModel::NormalizedPhonesRole)
```

```
regex : QRegularExpression ← QRegularExpression::escape(Цифры)
```

Иначе // режим поиска по тексту

```
proxyModel_->setFilterRole(Qt::DisplayRole)
```

```
regex : QRegularExpression ←
```

```
←QRegularExpression::escape(trimmedText)
```

Конец_Если

```
proxyModel_->setFilterRegularExpression(regex)
```

2.8 Сохранение и загрузка данных

Данные сохраняются в файл `contacts.json` в формате JSON с использованием классов `QJsonDocument`, `QJsonObject` и `QJsonArray`. Файл представляет собой массив объектов, где каждый объект соответствует одной записи контакта (`Contact`). Использование стандартного формата JSON обеспечивает интероперабельность и простоту анализа данных сторонними системами.

Спецификация формата JSON

Файл данных является корневым JSON `Array`, содержащим массив объектов контактов.

Ниже представлено детальное описание формата каждого поля в объекте контакта:

Таблица 1: Спецификация формата данных JSON

| Ключ | Тип данных | Описание и формат |
|--------------|------------|---------------------------------------|
| firstName | String | Имя контакта. |
| lastName | String | Фамилия контакта. |
| middleName | String | Отчество контакта. |
| address | String | Физический адрес. |
| birthDate | String | Дата рождения в формате "YYYY-MM-DD". |
| email | String | Электронная почта. |
| phoneNumbers | Object | Коллекция телефонных номеров. |

Спецификация поля phoneNumbers:

Поле `phoneNumbers` является вложенным JSON-объектом, который хранит пары "Тип телефона" → "Номер телефона" где и ключ, и значение являются строками. Такая структура позволяет хранить неограниченное количество телефонов для одного контакта.

```

1 [ 
2 {
3     "firstName": "Имя",
4     "lastName": "Фамилия",
5     "middleName": "Отчество",
6     "address": "улица дом кв",
7     "birthDate": "2005-10-14",
8     "email": "yourmail@mail.com",
9     "phoneNumbers": {
10         "Рабочий": "+711 111 11 11",
11         "Домашний": "+722 222 22 22",
12         "Служебный": "+733 333 33 33"
13     }
14 }
15 ]

```

Листинг 11: Пример contacts.json

Метод `saveToFile()`, расположенный в `ContactManager`, инкапсулирует логику сериализации коллекции контактов в файл формата JSON. Он использует API Qt для работы с JSON (`QJsonArray`, `QJsonObject`) и файловую систему (`QFile`). Алгоритм последовательно проходит по всем объектам `Contact`, преобразуя каждый из них в `QJsonObject`.

Метод `saveToFile(file_path)`:

`file : QFile ← QFile(file_path)`

Если Не `file.open(QIODevice::WriteOnly)`:

 Вернуть Ложь

Конец_Если

 МассивКонтактов : `QJsonArray ← QJsonArray()`

Для Каждого контакта В m_contacts:

ОбъектJson ← QJsonObject()

Вызвать ОбъектJson.insert("firstName контакта.firstName_")

Вызвать ОбъектJson.insert("lastName контакта.lastName_")

Вызвать ОбъектJson.insert("middleName контакта.middleName_")

Вызвать ОбъектJson.insert("address контакта.adress_")

Вызвать ОбъектJson.insert("birthDate контакта.birthDate_.toString())")

Вызвать ОбъектJson.insert("email контакта.email_")

ОбъектТелефонов : QJsonObject ← QJsonObject()

Для Каждой Пары (ТИП, НОМЕР) В контакта.phoneNumbers_:

Вызвать ОбъектТелефонов.insert(ТИП, НОМЕР)

Конец_Цикла

Вызвать ОбъектJson.insert("phoneNumbers ОбъектТелефонов")

Вызвать МассивКонтактов.append(ОбъектJson)

Конец_Цикла

Документ : QJsonDocument ← QJsonDocument(МассивКонтактов)

Вызвать file.write(Документ.toJson())

Вызвать file.close()

Вернуть Истина

При запуске приложения автоматически вызывается:

```
1 if (contactManager_.loadFromFile("contacts.json"))
2     tableView_->resetTable();
```

Листинг 12: Автозагрузка при старте

Это обеспечивает автоматическую загрузку ранее сохранённых данных.

3 Тестирование приложения

В данном разделе представлены основные сценарии использования приложения с визуальными иллюстрациями работы интерфейса.

3.1 Начальное состояние приложения

При первом запуске приложения отображается главное окно с пустой таблицей контактов и формой ввода данных (рис. 2). Если файл `contacts.json` существует, данные загружаются автоматически.

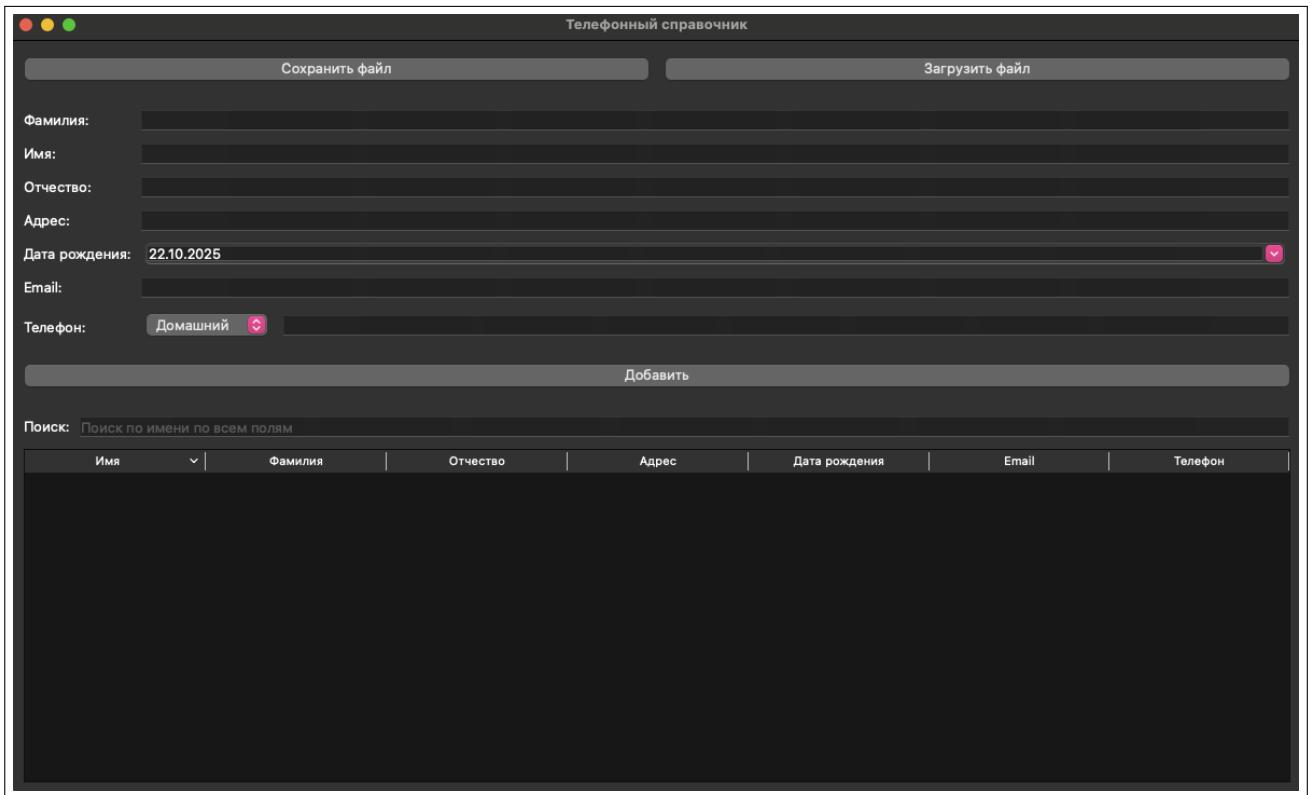


Рис. 2: Начальное состояние приложения

Элементы интерфейса:

- Форма ввода данных (верхняя часть окна);
- Кнопки управления: «Добавить», «Удалить», «Редактировать»;
- Поле поиска;
- Таблица для отображения контактов;
- Кнопки "Сохранить файл" и "Загрузить файл".

3.2 Сценарий 1: Добавление нового контакта

Пользователь заполняет все обязательные поля формы (рис. 3):

- Фамилия: Колмогоров
- Имя: Андрей
- Отчество: Николаевич
- Адрес: Москва Охотный Ряд Университет Ломоносовский проспект (МГУ)
- Дата рождения: 25.04.1903 (выбор через календарь)
- Email: kolmogorovs@probability.axioms
- Тип телефона: Рабочий
- Номер телефона: +7 (777) - 777 77 77

The screenshot shows a contact form with the following fields filled in:

| | |
|----------------|--|
| Фамилия: | Колмогоров |
| Имя: | Андрей |
| Отчество: | Николаевич |
| Адрес: | Москва Охотный Ряд Университет Ломоносовский проспект (МГУ) |
| Дата рождения: | 25.04.1903 |
| Email: | kolmogorovs@probability.axioms |
| Телефон: | Домашний <input type="button" value="+"/> +7 (777) - 777 77 77 |

At the bottom of the form is a button labeled "Добавить" (Add).

Рис. 3: Заполнение формы для добавления контакта

После нажатия кнопки "Добавить" происходит:

1. Валидация всех полей с помощью класса `Validator`;
2. Если данные корректны — добавление контакта в модель;
3. Автоматическое сохранение в файл `contacts.json`;
4. Отображение нового контакта в таблице;
5. Очистка полей формы;
6. Показ сообщения об успешном добавлении (рис. 4).

Фамилия: Колмогоров
Имя: Андрей
Отчество: Николаевич
Адрес: Москва Охотный Ряд Университет Ломоносовский про
Дата рождения: 25.04.1903
Email: kolmogorovs@probability.axioms
Телефон: Домашний +7 (777) - 777 77 77

Контакт успешно добавлен.
OK

Добавить

Поиск: Поиск по имени по всем полям

| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон |
|--------|------------|------------|------------------------|---------------|--|---------|
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... +7 (777) - 777 77 77 | |

Рис. 4: Сообщение об успешном добавлении контакта

После добавления контакт отображается в таблице (рис. 5). Таблица автоматически обновляется благодаря механизму Model/View.

| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон |
|--------|------------|------------|------------------------|---------------|--|---------|
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... +7 (777) - 777 77 77 | |

Рис. 5: Новый контакт добавлен в таблицу

3.3 Сценарий 2: Валидация некорректных данных

При попытке добавить контакт с некорректными данными система выводит сообщение об ошибке с указанием проблемных полей.

3.3.1 Пример 1: Некорректное имя

Имя начинается не с заглавной буквы или содержит недопустимые символы (рис. 6).

Фамилия: Марков
Имя: андрей
Отчество: Андреевич
Адрес: Санкт Петербург, Санкт-Петербургская академия наук
Дата рождения: 14.06.1856
Email: markovs@chain.spb
Телефон: Домашний +7 123 456 78 90

Пожалуйста, исправьте:
Неверное Имя.
OK

Добавить Удалить Редактировать

Рис. 6: Ошибка валидации имени

3.3.2 Пример 2: Некорректный email

Email не соответствует формату user@domain.zone (рис. 7).

A screenshot of a contact management application interface. On the left, there is a form with fields for Family Name (Фамилия), Name (Имя), Middle Name (Отчество), Address (Адрес), Date of Birth (Дата рождения), Email (Email), and Phone (Телефон). The Email field contains 'markovs@chainspb'. A modal dialog box in the center displays a blue folder icon and the message 'Пожалуйста, исправьте: Неверный Email.' (Please correct: Invalid Email.) with an 'OK' button. Below the form are buttons for 'Добавить' (Add), 'Удалить' (Delete), and 'Редактировать' (Edit).

Рис. 7: Ошибка валидации email

3.3.3 Пример 3: Множественные ошибки

При наличии нескольких ошибок система выводит все обнаруженные проблемы в одном сообщении (рис. 8).

A screenshot of a contact management application interface, similar to Figure 7. The form fields are identical, but the validation message in the central modal dialog is more extensive: 'Пожалуйста, исправьте: Неверное Имя. Неверная Фамилия. Неверный Email.' (Please correct: Invalid Name. Invalid Surname. Invalid Email.). The rest of the interface, including buttons and layout, is consistent with Figure 7.

Рис. 8: Множественные ошибки валидации

3.4 Сценарий 3: Редактирование существующего контакта

Пользователь выбирает строку в таблице, кликая по ней. При этом:

- Данные контакта автоматически загружаются в форму благодаря QDataWidgetMapper;
- Активируются кнопки "Удалить" и "Редактировать";
- В поле телефона отображается номер выбранного типа (рис. 9).

| Фамилия: | Колмогоров | | | | | | | | | | | | | | | | | | | | | |
|---|---|------------|--------------------------|---------------|---------------------------|----------------------|-------|---------|--------|------------|------------|------------------------|------------|---------------------------|----------------------|--------|--------|-----------|--------------------------|------------|-------------------|------------------|
| Имя: | Андрей | | | | | | | | | | | | | | | | | | | | | |
| Отчество: | Николаевич | | | | | | | | | | | | | | | | | | | | | |
| Адрес: | Москва Охотный Ряд Университет Ломоносовский проспект (МГУ) | | | | | | | | | | | | | | | | | | | | | |
| Дата рождения: | 25.04.1903 | | | | | | | | | | | | | | | | | | | | | |
| Email: | kolmogorovs@probability.axioms | | | | | | | | | | | | | | | | | | | | | |
| Телефон: | Домашний +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | |
| <input type="button" value="Добавить"/> <input type="button" value="Удалить"/> <input type="button" value="Редактировать"/> | | | | | | | | | | | | | | | | | | | | | | |
| Поиск: <input type="text" value="Поиск по имени по всем полям"/> | | | | | | | | | | | | | | | | | | | | | | |
| <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Имя</th> <th>Фамилия</th> <th>Отчество</th> <th>Адрес</th> <th>Дата рождения</th> <th>Email</th> <th>Телефон</th> </tr> </thead> <tbody> <tr> <td>Андрей</td> <td>Колмогоров</td> <td>Николаевич</td> <td>Москва Охотный Ряд ...</td> <td>25.04.1903</td> <td>kolmogorovs@probabilit...</td> <td>+7 (777) - 777 77 77</td> </tr> <tr> <td>Андрей</td> <td>Марков</td> <td>Андреевич</td> <td>Санкт Петербург, Санк...</td> <td>14.06.1856</td> <td>markovs@chain.spb</td> <td>+7 123 456 78 90</td> </tr> </tbody> </table> | | Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 |
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | | | | | | | | | | | | | | | | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | | | | | | | | | | | | | | | | |

Рис. 9: Выбор контакта для редактирования

После нажатия кнопки "Редактировать":

- Приложение переходит в режим редактирования (`isEditMode_ = true`);
- Кнопка "Добавить" меняет текст на "Сохранить";
- Кнопки "Удалить" и "Редактировать" скрываются;
- Появляется кнопка "Отмена";
- Таблица становится неактивной (нельзя выбрать другую строку);
- Все поля формы становятся доступны для редактирования (рис. 10).

| Фамилия: | Колмогоров | | | | | | | | | | | | | | | | | | | | | |
|---|---|------------|--------------------------|---------------|---------------------------|----------------------|-------|---------|--------|------------|------------|------------------------|------------|---------------------------|----------------------|--------|--------|-----------|--------------------------|------------|-------------------|------------------|
| Имя: | Андрей | | | | | | | | | | | | | | | | | | | | | |
| Отчество: | Николаевич | | | | | | | | | | | | | | | | | | | | | |
| Адрес: | Москва Охотный Ряд Университет Ломоносовский проспект (МГУ) | | | | | | | | | | | | | | | | | | | | | |
| Дата рождения: | <input type="text" value="25.04.1903"/> | | | | | | | | | | | | | | | | | | | | | |
| Email: | kolmogorovs@probability.axioms | | | | | | | | | | | | | | | | | | | | | |
| Телефон: | Домашний +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | |
| <input type="button" value="Сохранить"/> <input type="button" value="Отмена"/> | | | | | | | | | | | | | | | | | | | | | | |
| Поиск: <input type="text" value="Поиск по имени по всем полям"/> | | | | | | | | | | | | | | | | | | | | | | |
| <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Имя</th> <th>Фамилия</th> <th>Отчество</th> <th>Адрес</th> <th>Дата рождения</th> <th>Email</th> <th>Телефон</th> </tr> </thead> <tbody> <tr> <td>Андрей</td> <td>Колмогоров</td> <td>Николаевич</td> <td>Москва Охотный Ряд ...</td> <td>25.04.1903</td> <td>kolmogorovs@probabilit...</td> <td>+7 (777) - 777 77 77</td> </tr> <tr> <td>Андрей</td> <td>Марков</td> <td>Андреевич</td> <td>Санкт Петербург, Санк...</td> <td>14.06.1856</td> <td>markovs@chain.spb</td> <td>+7 123 456 78 90</td> </tr> </tbody> </table> | | Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 |
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | | | | | | | | | | | | | | | | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | | | | | | | | | | | | | | | | |

Рис. 10: Режим редактирования контакта

Пользователь изменяет необходимые поля и нажимает "Сохранить". Происходит:

1. Валидация изменённых данных;
2. Обновление контакта в модели через `mapper_->submit()`;

3. Ручное обновление телефонных номеров;
4. Автоматическое сохранение в файл;
5. Возврат в режим просмотра;
6. Показ сообщения об успешном редактировании (рис. 11).

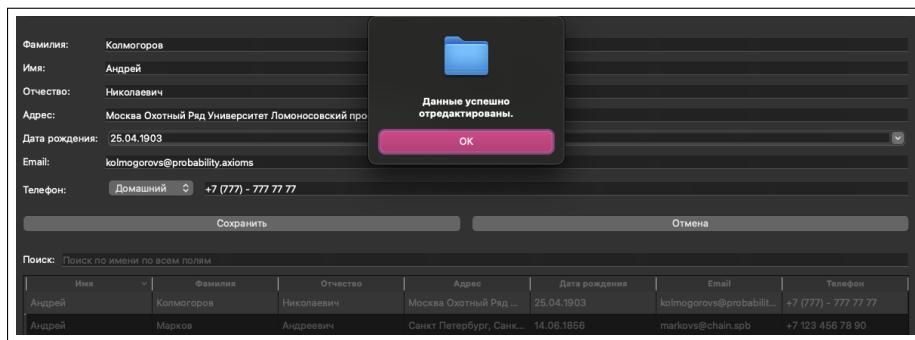


Рис. 11: Успешное редактирование контакта

При нажатии кнопки "Отмена":

- Вызывается `mapper_->revert()`, откатывающий все изменения;
- Приложение возвращается в режим просмотра;
- Поля формы очищаются;
- Таблица снова становится активной (рис. 12).

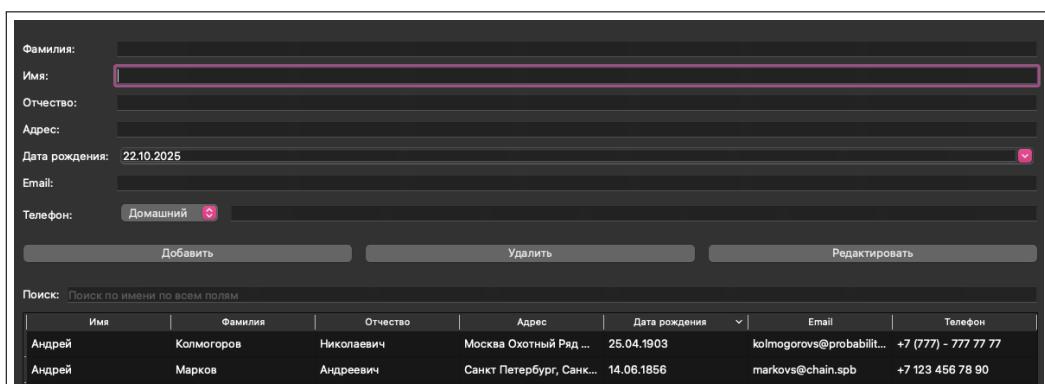


Рис. 12: Отмена редактирования

3.5 Сценарий 4: Управление телефонными номерами

Каждый контакт может иметь несколько телефонных номеров разных типов (рабочий, домашний, служебный). При добавлении или редактировании контакта:

1. Пользователь выбирает тип номера из выпадающего списка;
2. Вводит номер телефона;
3. При сохранении номер добавляется в QMap<QString, QString> с ключом, соответствующим типу (рис. 13).

Фамилия: Колмогоров
Имя: Андрей
Отчество: Николаевич
Адрес: Москва Охотный Ряд Университет Ломоносовский проспект (МГУ)
Дата рождения: 25.04.1903
Email: kolmogorovs@probability.axioms
Телефон: Рабочий
✓ Домашний +7 (777) - 777 77 77
Добавить Удалить Редактировать

Рис. 13: Выбор типа телефонного номера

3.5.1 Переключение между типами номеров

При выборе контакта в таблице пользователь может переключаться между типами номеров, выбирая нужный тип в выпадающем списке. При этом поле телефона автоматически обновляется, отображая соответствующий номер (рис. 14).

Фамилия: Колмогоров
Имя: Андрей
Отчество: Николаевич
Адрес: Москва Охотный Ряд Университет Ломоносовский проспект (МГУ)
Дата рождения: 25.04.1903
Email: kolmogorovs@probability.axioms
Телефон: Домашний
Добавить Удалить Редактировать

Рис. 14: Переключение между типами телефонов

3.6 Сценарий 5: Удаление контакта

Пользователь выбирает строку и нажимает кнопку "Удалить" (рис. 15).

| Фамилия: | Колмогоров | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|--|--------------------------|--|---------------------------|----------------------|--|-----|---------|----------|-------|---------------|-------|---------|--------|------------|------------|------------------------|------------|---------------------------|----------------------|--------|------------|------------|------------------------|------------|---------------------------|----------------------|--------|--------|-----------|--------------------------|------------|-------------------|------------------|
| Имя: | Андрей | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Отчество: | Николаевич | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Адрес: | Москва Охотный Ряд Университет Ломоносовский проспект (МГУ) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Дата рождения: | 25.04.1903 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Email: | kolmogorovs@probability.axioms | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Телефон: | Домашний +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <input type="button" value="Добавить"/> | | <input type="button" value="Удалить"/> | | <input type="button" value="Редактировать"/> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Поиск: Поиск по имени по всем полям <table border="1"> <thead> <tr> <th>Имя</th> <th>Фамилия</th> <th>Отчество</th> <th>Адрес</th> <th>Дата рождения</th> <th>Email</th> <th>Телефон</th> </tr> </thead> <tbody> <tr> <td>Андрей</td> <td>Колмогоров</td> <td>Николаевич</td> <td>Москва Охотный Ряд ...</td> <td>25.04.1903</td> <td>kolmogorovs@probabilit...</td> <td>+7 (777) - 777 77 77</td> </tr> <tr> <td>Андрей</td> <td>Колмогоров</td> <td>Николаевич</td> <td>Москва Охотный Ряд ...</td> <td>25.04.1903</td> <td>kolmogorovs@probabilit...</td> <td>+7 (777) - 777 77 77</td> </tr> <tr> <td>Андрей</td> <td>Марков</td> <td>Андреевич</td> <td>Санкт Петербург, Санк...</td> <td>14.06.1856</td> <td>markovs@chain.spb</td> <td>+7 123 456 78 90</td> </tr> </tbody> </table> | | | | | | | | Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 |
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Рис. 15: Выбор контакта для удаления

После удаления:

- Контакт исчезает из таблицы;
- Изменения автоматически сохраняются в файл;
- Поля формы очищаются;
- Выбирается следующая строка в таблице (рис. 16).

| Фамилия: | Колмогоров | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|--|--------------------------|--|---------------------------|----------------------|--|-----|---------|----------|-------|---------------|-------|---------|--------|------------|------------|------------------------|------------|---------------------------|----------------------|--------|--------|-----------|--------------------------|------------|-------------------|------------------|
| Имя: | Андрей | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Отчество: | Николаевич | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Адрес: | Москва Охотный Ряд Университет Ломоносовский проспект (МГУ) | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Дата рождения: | 25.04.1903 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Email: | kolmogorovs@probability.axioms | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Телефон: | Домашний +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <input type="button" value="Добавить"/> | | <input type="button" value="Удалить"/> | | <input type="button" value="Редактировать"/> | | | | | | | | | | | | | | | | | | | | | | | | |
| Поиск: Поиск по имени по всем полям <table border="1"> <thead> <tr> <th>Имя</th> <th>Фамилия</th> <th>Отчество</th> <th>Адрес</th> <th>Дата рождения</th> <th>Email</th> <th>Телефон</th> </tr> </thead> <tbody> <tr> <td>Андрей</td> <td>Колмогоров</td> <td>Николаевич</td> <td>Москва Охотный Ряд ...</td> <td>25.04.1903</td> <td>kolmogorovs@probabilit...</td> <td>+7 (777) - 777 77 77</td> </tr> <tr> <td>Андрей</td> <td>Марков</td> <td>Андреевич</td> <td>Санкт Петербург, Санк...</td> <td>14.06.1856</td> <td>markovs@chain.spb</td> <td>+7 123 456 78 90</td> </tr> </tbody> </table> | | | | | | | | Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 |
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | | | | | | | | | | | | | | | | | | | | | | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | | | | | | | | | | | | | | | | | | | | | | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | | | | | | | | | | | | | | | | | | | | | | |

Рис. 16: Результат удаления контакта

3.7 Сценарий 6: Поиск контактов

3.7.1 Поиск по фамилии

Пользователь вводит текст в поле поиска. Система автоматически фильтрует записи, оставляя только те, которые содержат введённую подстроку в любом из полей (рис. 17).

| Поиск: Колмогоров | | | | | | | |
|-------------------|------------|------------|------------------------|---------------|---------------------------|----------------------|--|
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | |

Рис. 17: Поиск по фамилии "Колмогоров"

3.7.2 Поиск по другим полям

Поиск работает по всем полям таблицы. Например, можно искать по фамилии, email или даже части телефонного номера (рис. 18).

| Поиск: @cha | | | | | | | |
|-------------|---------|-----------|--------------------------|---------------|-------------------|------------------|--|
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | |

Рис. 18: Поиск по части email

3.7.3 Очистка поиска

При очистке поля поиска отображаются все контакты (рис. 19).

| Поиск: Поиск по имени по всем полям | | | | | | | |
|-------------------------------------|------------|------------|--------------------------|---------------|---------------------------|----------------------|--|
| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | |

Рис. 19: Очистка поиска — все контакты видны

3.8 Сценарий 7: Сортировка данных

3.8.1 Сортировка по фамилии

Пользователь может кликнуть на заголовок столбца "Фамилия" для сортировки контактов по имени в алфавитном порядке. Повторный клик меняет направление сортировки (возрастание на убывание) (рис. 20).

| Имя | Фамилия | Отчество | Адрес | Дата рождения | Email | Телефон | |
|--------|------------|------------|--------------------------|---------------|---------------------------|----------------------|--|
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | kolmogorovs@probabilit... | +7 (777) - 777 77 77 | |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | markovs@chain.spb | +7 123 456 78 90 | |

Рис. 20: Сортировка по фамилии

3.8.2 Сортировка по дате рождения

Клик по столбцу "Дата рождения" сортирует по возрасту (рис. 21).

| Поиск: Поиск по имени по всем полям | | | | | | | |
|-------------------------------------|------------|------------|--------------------------|---------------|--|--|------------------|
| Имя | Фамилия | Отчество | Адрес | Дата рождения | | Email | Телефон |
| Андрей | Марков | Андреевич | Санкт Петербург, Санк... | 14.06.1856 | | markovs@chain.spb | +7 123 456 78 90 |
| Андрей | Колмогоров | Николаевич | Москва Охотный Ряд ... | 25.04.1903 | | kolmogorovs@probabil... +7 (777) - 777 77 77 | |

Рис. 21: Сортировка по дате рождения

3.9 Сценарий 8: Работа с файлами

3.9.1 Сохранение в файл

При нажатии кнопки "Сохранить файл" открывается диалоговое окно выбора пути сохранения (рис. 22). Пользователь может выбрать имя файла и директорию.

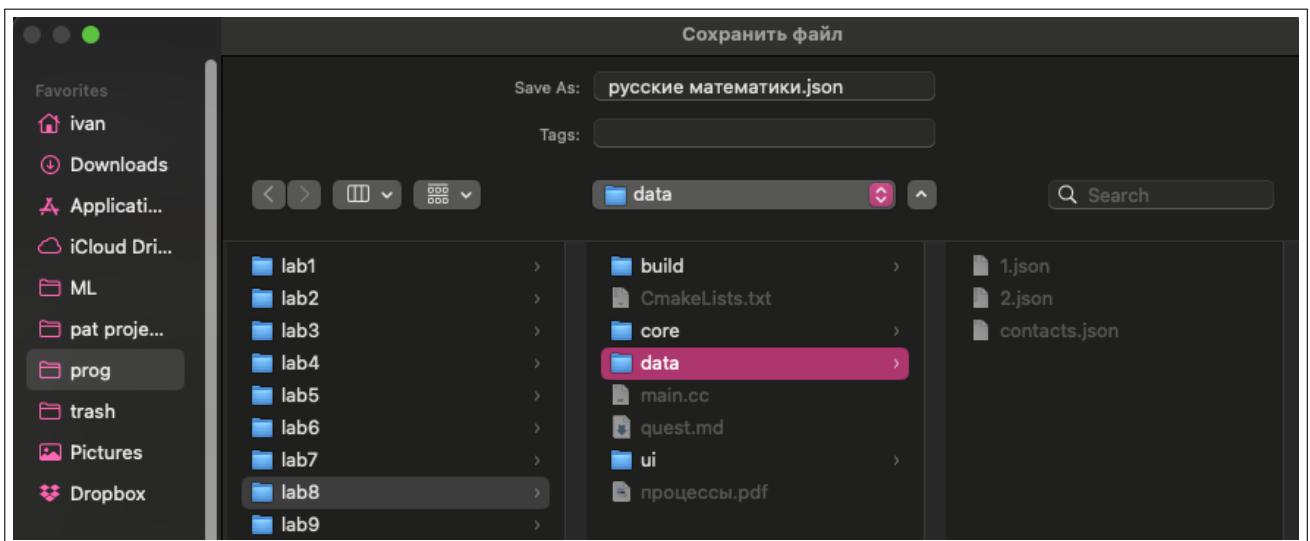


Рис. 22: Диалог сохранения файла

После сохранения отображается сообщение об успехе (рис. 23).

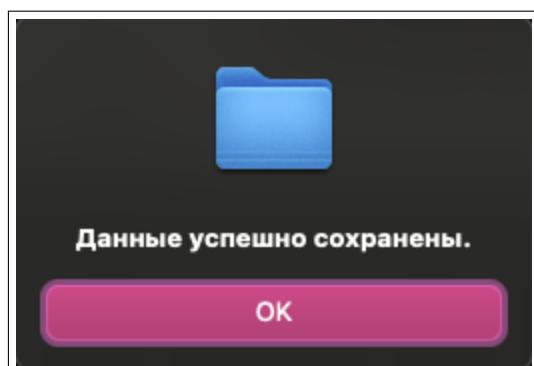


Рис. 23: Успешное сохранение файла

3.9.2 Загрузка из файла

При нажатии кнопки "Загрузить файл" открывается диалоговое окно выбора файла (рис. 24).

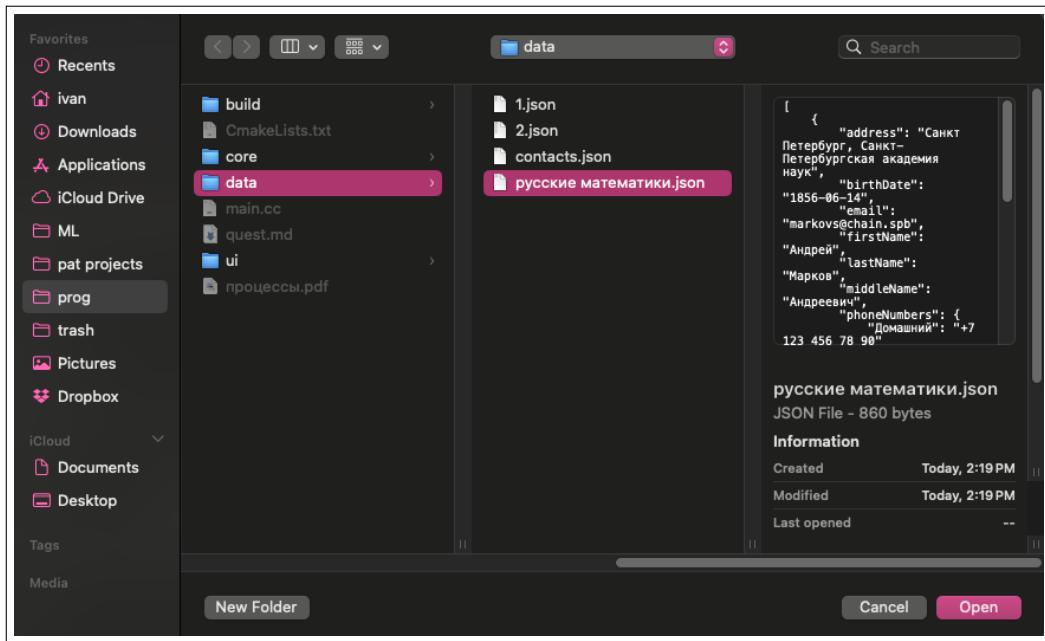


Рис. 24: Диалог загрузки файла

После успешной загрузки:

- Текущие данные заменяются данными из файла;
- Таблица полностью обновляется;
- Отображается сообщение об успешной загрузке (рис. 25).

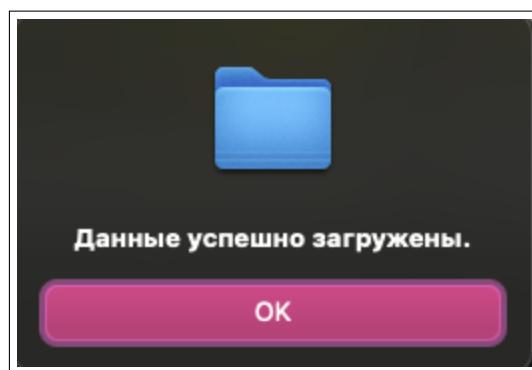


Рис. 25: Успешная загрузка файла

3.10 Сценарий 9: Использование календаря для выбора даты

Виджет QDateTimeEdit позволяет выбирать дату через календарь. При клике на иконку календаря открывается виджет QCalendarWidget (рис. 26).

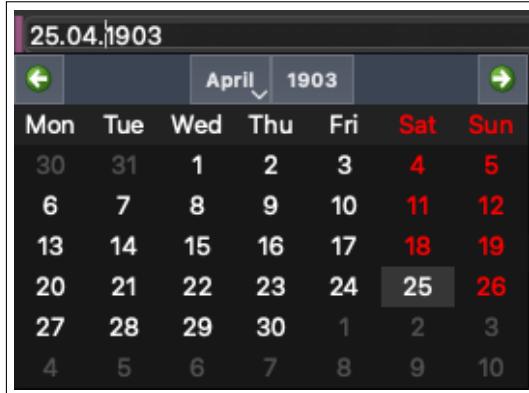


Рис. 26: Виджет календаря для выбора даты

3.11 Тестирование регулярных выражений

В таблице 1 представлены результаты тестирования валидации данных с различными входными значениями.

Таблица 2: Результаты тестирования валидации

| Поле | Значение | Результат | Комментарий |
|---------|------------------|------------------|-----------------------|
| Имя | Иван | Успех | Корректно |
| Имя | иван | Ошибка валидации | Не с заглавной буквы |
| Имя | Иван-Петр | Успех | Дефис разрешён |
| Имя | -Иван | Ошибка валидации | Начинается с дефиса |
| Имя | Иван123 | Успех | Цифры разрешены |
| Email | test@mail.ru | Успех | Корректно |
| Email | test@mail | Ошибка валидации | Нет зоны домена |
| Email | testmail.ru | Ошибка валидации | Нет символа @ |
| Email | test @mail.ru | Ошибка валидации | Пробел в email |
| Email | test@mail.ru.com | Ошибка валидации | Больше одной точки |
| Телефон | +79991234567 | Успех | Международный формат |
| Телефон | 8(999)123-45-67 | Успех | С скобками и дефисами |
| Телефон | +7 999 123 45 67 | Успех | С пробелами |
| Телефон | абвгд | Ошибка валидации | Только буквы |
| Дата | 14.10.2005 | Успех | Меньше текущей |
| Дата | 14.10.2030 | Ошибка валидации | Больше текущей |

3.12 Результаты тестирования

В результате тестирования были проверены следующие аспекты:

1. **Функциональность CRUD-операций:** все операции добавления, чтения, обновления и удаления работают корректно;
2. **Валидация данных:** все некорректные данные отсекаются, пользователь получает понятные сообщения об ошибках;
3. **Поиск и фильтрация:** система корректно фильтрует данные по введённому запросу;
4. **Сортировка:** сортировка по всем полям работает корректно в обоих направлениях;
5. **Работа с файлами:** сохранение и загрузка данных происходят без потери информации;
6. **Интерфейс:** все элементы интерфейса реагируют на действия пользователя, состояние кнопок меняется в зависимости от контекста.

4 Заключение

В результате выполнения лабораторной работы было разработано приложение «Телефонный справочник» с графическим интерфейсом на базе фреймворка Qt 6. Приложение реализует все требования, поставленные в задании, и демонстрирует практическое применение изученных концепций объектно-ориентированного программирования.

4.1 Реализованная функциональность

В ходе работы были успешно реализованы следующие функциональные подзадачи, что подтверждается конкретными техническими решениями:

1. Подзадача 1: Добавление нового контакта.

- Реализован метод `ContactTableModel::addContact(const Contact&)`, который добавляет новый контакт во внутреннюю коллекцию `ContactManager` после успешной валидации.
- Валидация всех полей (ФИО, Email, Телефон) выполняется статическими методами класса `Validator` с использованием регулярных выражений `QRegularExpression`.
- Для ввода даты рождения используется `QDateEdit` с всплывающим календарём `QCalendarWidget`.

2. Подзадача 2: Редактирование существующего контакта.

- Синхронизация данных таблицы с полями формы реализована через `QDataWidgetMapper`, который обеспечивает автоматическую загрузку данных при выборе строки.
- Изменения сохраняются вызовом `mapper_->submit()` в слоте `onAddButtonClicked()`.
- Откат изменений выполняется кнопкой «Отмена» через `mapper_->revert()`.

3. Подзадача 3: Удаление контакта.

- Удаление контакта из модели выполняется методом `ContactTableModel::removeContact(int index)`, что приводит к автоматическому обновлению `QTableView`.
- Кнопка «Удалить» активна только при выбранной строке в таблице.

4. Подзадача 4: Поиск и фильтрация контактов.

- Фильтрация реализована с помощью класса `QSortFilterProxyModel`, который оборачивает `ContactTableModel`.

- Поиск по тексту использует `Qt::DisplayRole` (регистронезависимый поиск по всем столбцам), а поиск по цифрам телефона — пользовательскую роль `ContactTableModel::NormalizedPhonesRole`, что позволяет искать по неформатированным номерам.

5. Подзадача 5: Сортировка данных.

- Сортировка активирована установкой `QTableView::setSortingEnabled(true)` и автоматически обрабатывается классом `QSortFilterProxyModel`.
- Доступна сортировка по любому столбцу в прямом и обратном порядке.

6. Подзадача 6: Сохранение и загрузка данных.

- Хранение данных осуществляется в формате JSON. Сериализация и десериализация реализованы с использованием `QJsonDocument`, `QJsonObject` и `QJsonArray` в классе `ContactManager`.
- Данные автоматически сохраняются в файл `contacts.json` после каждой операции CRUD и загружаются при старте приложения.

4.2 Использованные технологии и концепции

В процессе разработки были изучены и применены следующие ключевые технологии и концепции:

- **Операционная система:** macOS;
- **Язык программирования:** C++ 17;
- **Компилятор:** Clang 16.0.0;
- **Фреймворк:** Qt 6.9.2;
- **Система сборки:** CMake 4.0.3 с настройкой с Qt (MOC, UIC, RCC);
- **Архитектура:** Model/View/Controller (MVC), с реализацией `QAbstractTableModel` и `QSortFilterProxyModel`;
- **Взаимодействие:** Механизм сигналов и слотов, обеспечивающий связьность компонентов;
- **Валидация:** Класс `QRegularExpression` для проверки пользовательского ввода.

5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Qt Documentation. *Qt 6 Documentation*. URL: <https://doc.qt.io/qt-6/> (дата обращения: 21.10.2025).
2. Qt Documentation. *Qt Widgets Module*. URL: <https://doc.qt.io/qt-6/qtwidgets-index.html> (дата обращения: 21.10.2025).
3. Qt Documentation. *Model/View Programming*. URL: <https://doc.qt.io/qt-6/model-view-programming.html> (дата обращения: 21.10.2025).
4. Qt Documentation. *Signals & Slots*. URL: <https://doc.qt.io/qt-6/signalsandslots.html> (дата обращения: 21.10.2025).
5. Qt Documentation. *QTableView Class*. URL: <https://doc.qt.io/qt-6/qtableview.html> (дата обращения: 21.10.2025).
6. Qt Documentation. *QAbstractTableModel Class*. URL: <https://doc.qt.io/qt-6/qabstracttablemodel.html> (дата обращения: 21.10.2025).
7. Qt Documentation. *QSortFilterProxyModel Class*. URL: <https://doc.qt.io/qt-6/qsortfilterproxymodel.html> (дата обращения: 21.10.2025).
8. Qt Documentation. *QDataWidgetMapper Class*. URL: <https://doc.qt.io/qt-6/qdatawidgetmapper.html> (дата обращения: 21.10.2025).
9. Qt Documentation. *QRegularExpression Class*. URL: <https://doc.qt.io/qt-6/qregularexpression.html> (дата обращения: 21.10.2025).
10. Qt Documentation. *JSON Support in Qt*. URL: <https://doc.qt.io/qt-6/json.html> (дата обращения: 21.10.2025).
11. CMake Documentation. *Using Qt with CMake*. URL: <https://cmake.org/cmake/help/latest/manual/cmake-qt.7.html> (дата обращения: 21.10.2025).
12. cppreference.com. *C++ Standard Library Reference*. URL: <https://en.cppreference.com/> (дата обращения: 21.10.2025).
13. gemini.google.com. *AI assistent*. URL: <https://gemini.google.com> (дата обращения: 21.10.2025).

Приложение

Исходный код проекта размещён в открытом репозитории **GitHub** для проверки корректности реализации и воспроизводимости результатов.

Ссылка на GitHub репозиторий. URL:

<https://github.com/ivanchetvergov/seм3/tree/master/oop/lab8>
(дата обращения: 04.11.25).

Приложение А. Файл сборки CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.16)
2
3 project(lab8 VERSION 1.0 LANGUAGES CXX)
4
5 set(CMAKE_AUTOMOC ON)
6 set(CMAKE_AUTOUIC ON)
7 set(CMAKE_AUTORCC ON)
8
9 find_package(Qt6 REQUIRED COMPONENTS Widgets)
10
11 set(SOURCES
12     main.cc
13     ui/src/mainwindow.cc
14     ui/src/mainwindowui.cc
15     ui/src/mainwindowlogic.cc
16     core/src/contactmanager.cc
17     core/src/validator.cc
18     core/src/contacttablemodel.cc
19 )
20
21 set(HEADERS
22     ui/include/mainwindow.h
23     core/include/contact.h
24     core/include/contactmanager.h
25     core/include/validator.h
26     core/include/contacttablemodel.h
27 )
28
29 add_executable(${PROJECT_NAME}
30     ${SOURCES}
31     ${HEADERS}
32 )
33
34 target_include_directories(${PROJECT_NAME} PRIVATE
35     ${PROJECT_SOURCE_DIR}/ui/include
36     ${PROJECT_SOURCE_DIR}/core/include
37 )
38
39 target_link_libraries(${PROJECT_NAME} PRIVATE Qt6::Widgets)
```

Листинг 13: CMakeLists.txt

Приложение Б. Заголовочные файлы (core/include/)

```
1 // Contact.h
2 #pragma once
3
4 #include <QString>
5 #include <QDate>
6 #include < QMap>
7 #include <QUuid>
8
9 /* служит ORM моделью QTable */
10 struct Contact {
11     Q_GADGET
12 public:
13     QString firstName_;
14     QString lastName_;
15     QString middleName_;
16     QString adress_;
17     QDate birthDate_;
18     QString email_;
19     QMap<QString, QString> phoneNumbers_;
20     QUuid id_ = QUuid::createUuid();
21 };
```

Листинг 14: Contact.h

```
1 // ContactManager.h
2 #pragma once
3
4 #include < QList>
5 #include "Contact.h"
6
7 /* класс для управления коллекцией контактов */
8 class ContactManager {
9 public:
10     ContactManager() = default;
11     ~ContactManager() = default;
12
13     /* --- CRUD операции ---
14     void addContact(const Contact& contact);
15     void removeContact(int index);
16     void updateContact(int index, const Contact& contact);
17
18     /* --- геттеры, доступ к данным ---
19     const QList<Contact>& getContacts() const;
20     QList<Contact>& getContactsMutable();
21     int contactCount() const;
22     Contact* getContact(int index);
23
24     /* --- работа с данными ---
25     bool loadFromFile(const QString& filePath);
26     bool saveToFile(const QString& filePath) const;
27 }
```

```

28 private:
29     QList<Contact> m_contacts;
30 };

```

Листинг 15: ContactManager.h

```

1 // ContactTableModel.h
2 #pragma once
3 #include "Contact.h"
4 #include "ContactManager.h"
5
6 #include <QAbstractTableModel>
7 #include <QList>
8 #include <QVariant>
9
10 /* класс-адаптер (mapper) ContactManager -> QTableView
11 class ContactTableModel : public QAbstractTableModel {
12     Q_OBJECT
13
14 public:
15     explicit ContactTableModel(ContactManager* contactManager,
16                                 QObject* parent = nullptr);
17
18     /* --- обязательные методы QAbstractTableModel ---
19     int rowCount(const QModelIndex& parent = QModelIndex())
20         const override;
21     int columnCount(const QModelIndex& parent = QModelIndex())
22         const override;
23     QVariant data(const QModelIndex& index,
24                   int role = Qt::DisplayRole) const override;
25     QVariant headerData(int section, Qt::Orientation orientation,
26                         int role = Qt::DisplayRole) const
27         override;
28     bool setData(const QModelIndex& index, const QVariant& value,
29                  int role = Qt::EditRole) override;
30     Qt::ItemFlags flags(const QModelIndex& index) const override;
31
32     /* --- методы-обертки CRUD операций ---
33     void addContact(const Contact& contact);
34     void removeContact(int index);
35     void updateContact(int index, const Contact& contact);
36
37     /* --- геттеры ---
38     const Contact& getContact(int row) const;
39
40 public slots:
41     void resetTable();
42
43 private:
44     ContactManager* contactManager_;
45     static constexpr int ContactIdRole = Qt::UserRole + 1;
46 };

```

Листинг 16: ContactTableModel.h

```

1 // Validator.h
2 #pragma once
3
4 #include <QString>
5 #include <QDate>
6
7 /* класс утилита для валидации данных
8 class Validator {
9 public:
10     static bool isValidName(const QString& name);
11     static bool isValidPhoneNumber(const QString& number);
12     static bool isValidEmail(const QString& email);
13     static bool isValidBirthDate(const QDate& date);
14 };

```

Листинг 17: Validator.h

Приложение В. Исходные файлы (core/src/)

```

1 // ContactManager.cc
2 #include "ContactManager.h"
3 #include <QFile>
4 #include <QJsonDocument>
5 #include <QJsonObject>
6 #include <QJsonArray>
7 #include <QDebug>
8
9 void ContactManager::addContact(const Contact& contact) {
10     m_contacts.append(contact);
11 }
12
13 void ContactManager::removeContact(int index) {
14     if (index >= 0 && index < m_contacts.size())
15         m_contacts.removeAt(index);
16 }
17
18 void ContactManager::updateContact(int index, const Contact&
19 contact) {
20     if (index >= 0 && index < m_contacts.size()) {
21         m_contacts[index] = contact;
22     }
23 }
24
25 const QList<Contact>& ContactManager::getContacts() const {
26     return m_contacts;
27 }
28
29 bool ContactManager::saveToFile(const QString& filePath) const {
30     QJsonArray contactArray;
31
32     for (const auto& contact : m_contacts) {

```

```

32     QJsonObject contactObject;
33     contactObject["firstName"] = contact.firstName_;
34     contactObject["lastName"] = contact.lastName_;
35     contactObject["middleName"] = contact.middleName_;
36     contactObject["address"] = contact.adress_;
37     contactObject["birthDate"] =
38         contact.birthDate_.toString(Qt::ISODate);
39     contactObject["email"] = contact.email_;
40
41     QJsonObject phonesObject;
42     for (auto it = contact.phoneNumbers_.constBegin();
43          it != contact.phoneNumbers_.constEnd(); ++it) {
44         phonesObject[it.key()] = it.value();
45     }
46     contactObject["phoneNumbers"] = phonesObject;
47     contactArray.append(contactObject);
48 }
49
50 QJsonDocument doc(contactArray);
51 QFile file(filePath);
52 if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
53     file.write(doc.toJson());
54     file.close();
55     return true;
56 }
57
58 qWarning() << "Could not save file:" << file.errorString();
59 return false;
}
60
61
62 bool ContactManager::loadFromFile(const QString& filePath) {
63     QFile file(filePath);
64     if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
65         qWarning() << "Could not open file:" <<
66             file.errorString();
67         return false;
68     }
69
70     QByteArray data = file.readAll();
71     file.close();
72
73     QJsonDocument doc = QJsonDocument::fromJson(data);
74     if (doc.isNull() || !doc.isArray()) {
75         qWarning() << "Invalid JSON document.";
76         return false;
77     }
78
79     m_contacts.clear();
80     QJsonArray contactArray = doc.array();
81     for (const QJsonValue& value : contactArray) {
82         QJsonObject contactObject = value.toObject();

```

```

83     contact.firstName_ =
84         contactObject["firstName"].toString();
85     contact.lastName_ = contactObject["lastName"].toString();
86     contact.middleName_ =
87         contactObject["middleName"].toString();
88     contact.adress_ = contactObject["address"].toString();
89     contact.birthDate_ = QDate::fromString(
90         contactObject["birthDate"].toString(), Qt::ISODate);
91     contact.email_ = contactObject["email"].toString();

92     QJsonObject phonesObject =
93         contactObject["phoneNumbers"].toObject();
94     for (const QString& key : phonesObject.keys()) {
95         contact.phoneNumbers_[key] =
96             phonesObject[key].toString();
97     }
98     m_contacts.append(contact);
99 }
100
101 QList<Contact>& ContactManager::getContactsMutable() {
102     return m_contacts;
103 }
104
105 Contact* ContactManager::getContact(int index) {
106     if (index >= 0 && index < m_contacts.size()) {
107         return &m_contacts[index];
108     }
109     return nullptr;
110 }
111
112 int ContactManager::contactCount() const {
113     return m_contacts.size();
114 }
```

Листинг 18: ContactManager.cc

```

1 // ContactTableModel.cc
2 #include "ContactTableModel.h"
3
4 ContactTableModel::ContactTableModel(ContactManager*
5     contactManager,
6                                     QObject* parent)
7     : QAbstractTableModel(parent),
8     contactManager_(contactManager) {}
9
10 int ContactTableModel::rowCount(const QModelIndex& parent) const
11 {
12     Q_UNUSED(parent);
13     return contactManager_->getContacts().size();
14 }
```

```

14 int ContactTableModel::columnCount(const QModelIndex& parent)
15     const {
16     Q_UNUSED(parent);
17     return 7;
18 }
19 QVariant ContactTableModel::data(const QModelIndex &index,
20                                 int role) const {
21     if (!index.isValid() ||
22         index.row() >= contactManager_->getContacts().size()) {
23         return QVariant();
24     }
25
26     const QList<Contact>& contacts =
27         contactManager_->getContacts();
28     const Contact& contact = contacts.at(index.row());
29
30     if (role == Qt::DisplayRole || role == Qt::EditRole) {
31         switch (index.column()) {
32             case 0: return contact.firstName_;
33             case 1: return contact.lastName_;
34             case 2: return contact.middleName_;
35             case 3: return contact.adress_;
36             case 4: return contact.birthDate_;
37             case 5: return contact.email_;
38             case 6: return
39                     contact.phoneNumbers_.values().join(", ");
40         }
41     }
42
43     if (role == ContactIdRole) {
44         return contact.id_;
45     }
46
47     return QVariant();
48 }
49
50 QVariant ContactTableModel::headerData(int section,
51                                       Qt::Orientation
52                                         orientation,
53                                         int role) const {
54     if (role != Qt::DisplayRole || orientation !=
55         Qt::Horizontal) {
56         return QVariant();
57     }
58
59     switch (section) {
60         case 0: return "Имя";
61         case 1: return "Фамилия";
62         case 2: return "Отчество";
63         case 3: return "Адрес";
64         case 4: return "Дата рождения";
65         case 5: return "Email";

```

```

62         case 6: return "Телефон";
63     default: return QVariant();
64 }
65
66
67 bool ContactTableModel::setData(const QModelIndex& index,
68                                 const QVariant& value,
69                                 int role) {
70     if (index.isValid() && role == Qt::EditRole) {
71         QList<Contact>& contacts =
72             contactManager_->getContactsMutable();
73         Contact& contact = contacts[index.row()];
74
75         switch (index.column()) {
76             case 0: contact.firstName_ = value.toString(); break;
77             case 1: contact.lastName_ = value.toString(); break;
78             case 2: contact.middleName_ = value.toString();
79                 break;
80             case 3: contact.adress_ = value.toString(); break;
81             case 4: contact.birthDate_ = value.toDate(); break;
82             case 5: contact.email_ = value.toString(); break;
83             case 6: contact.phoneNumbers_["Рабочий"] =
84                     value.toString(); break;
85             default: return false;
86         }
87
88         emit dataChanged(index, index, {role});
89         return true;
90     }
91     return false;
92 }
93
94 Qt::ItemFlags ContactTableModel::flags(const QModelIndex& index)
95     const {
96     return QAbstractTableModel::flags(index) |
97             Qt::ItemIsEditable;
98 }
99
100 void ContactTableModel::addContact(const Contact& contact) {
101     int newRow = contactManager_->getContacts().size();
102     beginInsertRows(QModelIndex(), newRow, newRow);
103     contactManager_->addContact(contact);
104     endInsertRows();
105 }
106
107 void ContactTableModel::removeContact(int index) {
108     if (index >= 0 && index <
109         contactManager_->getContacts().size()) {
110         beginRemoveRows(QModelIndex(), index, index);
111         contactManager_->removeContact(index);
112         endRemoveRows();
113     }
114 }
```

```

110
111 void ContactTableModel::updateContact(int index,
112                                     const Contact& contact) {
113     if (index >= 0 && index <
114         contactManager_->getContacts().size()) {
115         contactManager_->updateContact(index, contact);
116         emit dataChanged(this->index(index, 0),
117                           this->index(index, columnCount() - 1));
118     }
119 }
120
121 const Contact& ContactTableModel::getContact(int row) const {
122     return *contactManager_->getContact(row);
123 }
124
125 void ContactTableModel::resetTable() {
126     beginResetModel();
127     endResetModel();
128 }
```

Листинг 19: ContactTableModel.cc

```

1 #include "Validator.h"
2 #include <QRegularExpression>
3 #include <QDate>
4
5 bool Validator::isValidName(const QString& name) {
6     QString trimmedName = name.trimmed();
7
8     if (trimmedName.isEmpty()) return false;
9
10    if (trimmedName.startsWith(' ') || trimmedName.endsWith(' '))
11        return false;
12
13    QRegularExpression re(
14        "[A-ЯА-҃] [А-҃а-҃А-҃а-҃ -]*[А-҃а-҃А-҃а-҃0-9]$");
15    return re.match(trimmedName).hasMatch();
16 }
17
18 bool Validator::isValidPhoneNumber(const QString& number) {
19     QRegularExpression re("^(\\+)?[\\s\\d()-]+$");
20     return re.match(number).hasMatch();
21 }
22
23 bool Validator::isValidEmail(const QString& email) {
24     QString trimmedEmail = email.trimmed();
25     if (trimmedEmail.isEmpty()) return false;
26
27     QRegularExpression re(
28         "[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[a-zA-Z0-9]+$");
29     return re.match(trimmedEmail).hasMatch();
30 }
```

```

32 bool Validator::isValidBirthDate(const QDate& date) {
33     return date.isValid() && date < QDate::currentDate();
34 }

```

Листинг 20: Validator.cc

Приложение Д. Заголовочный файл MainWindow

```

1 // MainWindow.h
2 #pragma once
3
4 #include < QMainWindow >
5 #include < QSortFilterProxyModel >
6 #include "ContactManager.h"
7 #include "ContactTableModel.h"
8
9 class QPushButton;
10 class QLineEdit;
11 class QDateEdit;
12 class QComboBox;
13 class QVBoxLayout;
14 class QTableView;
15 class QDataWidgetMapper;
16
17 class MainWindow : public QMainWindow {
18     Q_OBJECT
19
20 public:
21     explicit MainWindow(QWidget *parent = nullptr);
22     ~MainWindow() = default;
23
24 private slots:
25     void onAddButtonClicked();
26     void onRemoveButtonClicked();
27     void onEditButtonClicked();
28     void onSaveButtonClicked();
29     void onLoadButtonClicked();
30     void onCancelButtonClicked();
31     void onSearchTextChanged(const QString& text);
32     void onSelectionChanged();
33     void onPhoneTypeChanged(const QString& type);
34
35 private:
36     static constexpr int ContactIdRole = Qt::UserRole + 1;
37
38     void setupUi();
39     void setupMapper();
40     void clearInputFields();
41     Contact getCurrentContactFromForm() const;
42     bool validateContact(const Contact& contact);
43
44     bool isInEditMode_ = false;

```

```

45     void setEditingMode(bool isInEditMode);
46
47     ContactManager contactManager_;
48     ContactTableModel* tableModel_;
49     QSortFilterProxyModel* proxyModel_;
50     QTableView* tableView_;
51
52     QDataWidgetMapper* mapper_;
53
54     QLineEdit* firstNameInput_;
55     QLineEdit* lastNameInput_;
56     QLineEdit* middleNameInput_;
57     QLineEdit* addressInput_;
58     QDateEdit* birthDateInput_;
59     QLineEdit* emailInput_;
60     QLineEdit* mainPhoneInput_;
61     QComboBox* phoneTypeComboBox_;
62
63     QPushButton* addButton_;
64     QPushButton* removeButton_;
65     QPushButton* editButton_;
66     QPushButton* cancelButton_;
67
68     QLineEdit* searchInput_;
69     QVBoxLayout* mainLayout_;
70 };

```

Листинг 21: MainWindow.h

Приложение Е. Исходные файлы MainWindow

```

1 // MainWindow.cc
2 #include "MainWindow.h"
3 #include "ContactTableModel.h"
4 #include "ContactManager.h"
5 #include "Contact.h"
6
7 #include <QTableView>
8 #include <QPushButton>
9 #include <QLineEdit>
10 #include <QItemSelectionModel>
11 #include <QHBoxLayout>
12 #include <QVBoxLayout>
13 #include <QDataWidgetMapper>
14 #include <QMessageBox>
15 #include <QFileDialog>
16 #include <QDateEdit>
17 #include <QComboBox>
18
19 MainWindow::MainWindow(QWidget *parent)
20     : QMainWindow(parent)
21 {

```

```

22     setupUi();
23
24     tableView_ = new ContactTableModel(&contactManager_, this);
25
26     proxyModel_ = new QSortFilterProxyModel(this);
27     proxyModel_->setSourceModel(tableView_);
28     proxyModel_->setFilterCaseSensitivity(Qt::CaseInsensitive);
29
30     tableView_->setModel(proxyModel_);
31     tableView_->setSortingEnabled(true);
32
33     connect(addButton_, &QPushButton::clicked,
34             this, &MainWindow::onAddButtonClicked);
35     connect(removeButton_, &QPushButton::clicked,
36             this, &MainWindow::onRemoveButtonClicked);
37     connect(editButton_, &QPushButton::clicked,
38             this, &MainWindow::onEditButtonClicked);
39     connect(cancelButton_, &QPushButton::clicked,
40             this, &MainWindow::onCancelButtonClicked);
41     connect(phoneTypeComboBox_, &QComboBox::currentTextChanged,
42             this, &MainWindow::onPhoneTypeChanged);
43
44     connect(searchInput_, &QLineEdit::textChanged,
45             this, &MainWindow::onSearchTextChanged);
46
47     connect(tableView_->selectionModel(),
48             &QItemSelectionModel::currentChanged,
49             this, &MainWindow::onSelectionChanged);
50
51     QPushButton* saveButton = new QPushButton("Сохранить файл");
52     QPushButton* loadButton = new QPushButton("Загрузить файл");
53
54     QHBoxLayout* fileButtonsLayout = new QHBoxLayout();
55     fileButtonsLayout->addWidget(saveButton);
56     fileButtonsLayout->addWidget(loadButton);
57
58     mainLayout_->insertLayout(0, fileButtonsLayout);
59
60     connect(saveButton, &QPushButton::clicked,
61             this, &MainWindow::onSaveButtonClicked);
62     connect(loadButton, &QPushButton::clicked,
63             this, &MainWindow::onLoadButtonClicked);
64
65     setupMapper();
66
67     phoneTypeComboBox_->setCurrentText("Домашний");
68
69     if (contactManager_.loadFromFile("contacts.json"))
70         tableView_->resetTable();
71 }
72
73 void MainWindow::setupMapper() {
74     mapper_ = new QDataWidgetMapper(this);

```

```

75     mapper_->setModel(proxyModel_);
76     mapper_->setSubmitPolicy(QDataWidgetMapper::ManualSubmit);
77
78     mapper_->addMapping(firstNameInput_, 0);
79     mapper_->addMapping(lastNameInput_, 1);
80     mapper_->addMapping(middleNameInput_, 2);
81     mapper_->addMapping(addressInput_, 3);
82     mapper_->addMapping(birthDateInput_, 4, "date");
83     mapper_->addMapping(emailInput_, 5);
84 }
```

Листинг 22: MainWindow.cc — конструктор и инициализация

```

1 // MainWindowUI.cc
2 #include "MainWindow.h"
3
4 #include <QVBoxLayout>
5 #include <QGridLayout>
6 #include <QHBoxLayout>
7 #include <QLabel>
8 #include <QDateEdit>
9 #include <QPushButton>
10 #include <QLineEdit>
11 #include <QTableView>
12 #include <QHeaderView>
13 #include <QComboBox>
14
15 void MainWindow::setupUi()
16 {
17     // устанавливаем центральный виджет
18     QWidget* centralWidget = new QWidget(this);
19     setCentralWidget(centralWidget);
20
21     // главный вертикальный слой
22     mainLayout_ = new QVBoxLayout(centralWidget);
23
24     // * --- 1. секция формы ввода (Grid Layout) ---
25     QGridLayout* inputLayout = new QGridLayout();
26
27     //инициализация виджетов ввода
28     lastNameInput_ = new QLineEdit();
29     firstNameInput_ = new QLineEdit();
30     middleNameInput_ = new QLineEdit();
31     addressInput_ = new QLineEdit();
32     // др
33     birthDateInput_ = new QDateEdit(QDate::currentDate());
34     birthDateInput_->setDisplayFormat("dd.MM.yyyy");
35     birthDateInput_->setCalendarPopup(true);
36
37     emailInput_ = new QLineEdit();
38
39     mainPhoneInput_ = new QLineEdit();
40     phoneTypeComboBox_ = new QComboBox();
```

```

41 phoneTypeComboBox_->addItem("Рабочий");
42 phoneTypeComboBox_->addItem("Домашний");
43 phoneTypeComboBox_->addItem("Служебный");
44
45 // размещение полей в сетке (QLabel, QLineEdit)
46 inputLayout->addWidget(new QLabel("Фамилия:"), 0, 0);
47 inputLayout->addWidget(lastNameInput_, 0, 1);
48 inputLayout->addWidget(new QLabel("Имя:"), 1, 0);
49 inputLayout->addWidget(firstNameInput_, 1, 1);
50 inputLayout->addWidget(new QLabel("Отчество:"), 2, 0);
51 inputLayout->addWidget(middleNameInput_, 2, 1);
52 inputLayout->addWidget(new QLabel("Адрес:"), 3, 0);
53 inputLayout->addWidget(addressInput_, 3, 1);
54 inputLayout->addWidget(new QLabel("Дата рождения:"), 4, 0);
55 inputLayout->addWidget(birthDateInput_, 4, 1);
56 inputLayout->addWidget(new QLabel("Email:"), 5, 0);
57 inputLayout->addWidget(emailInput_, 5, 1);
58
59 // размещение телефона: тип и номер в одной строке
60 QHBoxLayout* phoneInputRow = new QHBoxLayout();
61 phoneInputRow->addWidget(phoneTypeComboBox_);
62 phoneInputRow->addWidget(mainPhoneInput_);
63 inputLayout->addWidget(new QLabel("Телефон:"), 6, 0);
64 inputLayout->addLayout(phoneInputRow, 6, 1);
65
66 mainLayout_->addLayout(inputLayout);
67
68 // * --- 2 секция кнопок (Horizontal Layout)
69 QHBoxLayout* buttonLayout = new QHBoxLayout();
70 addButton_ = new QPushButton("Добавить");
71 removeButton_ = new QPushButton("Удалить");
72 editButton_ = new QPushButton("Редактировать");
73 cancelButton_ = new QPushButton("Отмена");
74
75 // инициализация состояния кнопок
76 removeButton_->setEnabled(false); // нельзя удалить пока ничего не выбрано
77 editButton_->setEnabled(false); // нельзя редактировать пока ничего не выбрано
78 cancelButton_->hide(); // скрыта до начала редактирования/добавления
79
80 buttonLayout->addWidget(addButton_);
81 buttonLayout->addWidget(removeButton_);
82 buttonLayout->addWidget(editButton_);
83 buttonLayout->addWidget(cancelButton_);
84 mainLayout_->addLayout(buttonLayout);
85
86 // * 3 секция поиска (Horizontal Layout)
87 QHBoxLayout* searchLayout = new QHBoxLayout();
88
89 // поиск
90 searchInput_ = new QLineEdit();

```

```

91     searchInput_->setPlaceholderText("Поиск по имени по всем полям");
92     searchLayout->addWidget(new QLabel("Поиск:"));
93     searchLayout->addWidget(searchInput_);
94
95     mainLayout_->addLayout(searchLayout);
96
97     // * --- 4 секция таблицы (View)
98     tableView_ = new QTableView();
99     tableView_->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
100    // расстягнуть столбцы
101    tableView_->setSelectionBehavior(QAbstractItemView::SelectRows);
102    // выделение только целых строк
103    tableView_->setSelectionMode(QAbstractItemView::SingleSelection);
104    // только одна строка может быть выбрана
105    tableView_->setEditTriggers(QAbstractItemView::NoEditTriggers);
106    // запрещаем редактирование в самой таблице
107
108    mainLayout_->addWidget(tableView_);
109 }

```

Листинг 23: MainWindowUI.cc

```

1 // MainWindowLogic.cc
2 #include "MainWindow.h"
3
4 #include "Contact.h"
5 #include "Validator.h"
6 #include "ContactTableModel.h"
7
8 #include <QMessageBox>
9 #include <QFileDialog>
10 #include <QDateEdit>
11 #include <QLineEdit>
12 #include <QTableView>
13 #include <QDataWidgetMapper>
14 #include <QPushButton>
15 #include <QComboBox>
16
17 // слот для кнопки "Добавить" / "Сохранить"
18 void MainWindow::onAddButtonClicked() {
19     Contact contact = getCurrentContactFromForm();
20
21     if (!validateContact(contact)) {
22         return; // валидация не пройдена
23     }
24
25     // кнопка используется для двух целей
26     if (isInEditMode_) {
27         // * режим редактирования:
28         // --- 1 ручное обновление телефона в модели
29         QModelIndex proxyIndex = tableView_->currentIndex();
30         QModelIndex sourceIndex =

```

```

    proxyModel_->mapToSource(proxyIndex);

31
32     QString selectedType = phoneTypeComboBox_->currentText();
33     QString newNumber = mainPhoneInput_->text().trimmed();
34
35     // получаем изменяемый контакт и обновляем телефон по выб-
36     // ранному типу
37     Contact& contactToEdit =
38         contactManager_.getContactsMutable() [sourceIndex.row()];
39     contactToEdit.phoneNumbers_[selectedType] = newNumber;
40
41     // 2 --- авт обновление остальных полей
42     // используем маппер для записи данных из формы обратно в
43     // Модель
44     if (mapper_->submit()) {
45         QMessageBox::information(this, "Успех", "Данные успеш-
46         но отредактированы.");
47     } else {
48         // ошибка submit обычно означает проблему с типом дан-
49         // ных
50         QMessageBox::warning(this, "Ошибка", "Не удалось сохр-
51         -анить изменения.");
52         return;
53     }
54     } else {
55         // * режим добавления:
56         // маппер не нужен. используем прямую вставку через модел-
57         // ъ
58         tableModel_->addContact(contact);
59         QMessageBox::information(this, "Успех", "Контакт успешно
60         добавлен.");
61     }

62     // общие действия
63     contactManager_.saveToFile("../data/contacts.json");      // со
64     хранием на диск
65     setEditMode(false);                                         // во
66     // возвращаемся в режим просмотра
67 }

68
69 // слот для кнопки "Удалить"
70 void MainWindow::onRemoveButtonClicked() {
71     // получаем индекс выбранной строки
72     QModelIndex proxyIndex = tableView_->currentIndex();
73     if (!proxyIndex.isValid()) {
74         QMessageBox::information(this, "Удаление", "Пожалуйста, в-
75         ыберите запись для удаления.");
76     }
77
78     // преобразуем индекс из Proxy-модели в индекс исходной модели
79     // и
80     int sourceRow = proxyModel_->mapToSource(proxyIndex).row();

```

```

71 // удаляем из исходной модели
72 tableView_->removeContact(sourceRow);
73
74 contactManager_.saveToFile("../data/contacts.json");
75 // setEditingMode(false); // ! теперь явно остается контакт
76 }
77
78 // слот для кнопки "Редактировать"
79 void MainWindow::onEditButtonClicked() {
80     QModelIndex proxyIndex = tableView_->currentIndex();
81     if (!proxyIndex.isValid()) {
82         QMessageBox::information(this, "Редактирование",
83             "Пожалуйста, выберите запись для редактирования.");
84         return;
85     }
86
87     setEditingMode(true);
88     mapper_->setCurrentIndex(proxyIndex.row()); // указываем мапперу строку
89
90     QMessageBox::information(this, "Редактирование", "Теперь вы можете редактировать данные. Нажмите 'Сохранить' для применения.");
91 }
92
93 void MainWindow::onCancelButtonClicked() {
94     mapper_->revert(); // откатываем все изменения в полях, сделанные маппером
95     setEditingMode(false);
96 }
97
98 void MainWindow::onSearchTextChanged(const QString& text) {
99     QString trimmed = text.trimmed();
100    proxyModel_->setFilterKeyColumn(-1);
101    proxyModel_->setFilterCaseSensitivity(Qt::CaseInsensitive);
102
103    if (trimmed.isEmpty()) {
104        proxyModel_->setFilterRegularExpression(QRegularExpression());
105        proxyModel_->setFilterRole(Qt::DisplayRole);
106        return;
107    }
108
109    // извлекаем только цифры из запроса
110    QString digits;
111    for (QChar c : trimmed) if (c.isDigit()) digits += c;
112
113    if (!digits.isEmpty()) {
114        // ищем по нормализованной строке (в модели только цифры)
115        proxyModel_->setFilterRole(ContactTableModel::NormalizedPhonesRole);
116        proxyModel_->setFilterRegularExpression(QRegularExpression(QRegul
117    } else {

```

```

118     // обычный текстовый поиск по DisplayRole
119     proxyModel_->setFilterRole(Qt::DisplayRole);
120     proxyModel_->setFilterRegularExpression(QRegularExpression(QRegula
121                                         QRegula
122 }
123 }
124
125 // слот для обработки выбора строки в таблице
126 void MainWindow::onSelectionChanged() {
127     QModelIndex proxyIndex = tableView_->currentIndex();
128     bool rowIsSelected = proxyIndex.isValid();
129
130     // показываем / скрываем кнопки редактирования и удаления в з
131     // ависимости от выбора
132     // также прячем/показываем только если мы не в режиме редакти
133     // рования
134     if (!isEditMode_) {
135         removeButton_->setVisible(rowIsSelected);
136         editButton_->setVisible(rowIsSelected);
137         removeButton_->setEnabled(rowIsSelected);
138         editButton_->setEnabled(rowIsSelected);
139     } else {
140         // в режиме редактирования кнопки могут быть скрыты по ло
141         // гику setEditingMode
142         removeButton_->setVisible(false);
143         editButton_->setVisible(false);
144     }
145
146     if (rowIsSelected && !isEditMode_) {
147         // синхронизируем маппер с текущей строкой
148         mapper_->setCurrentIndex(proxyIndex.row());
149
150         // ручное управление телефоном
151         QModelIndex sourceIndex =
152             proxyModel_->mapToSource(proxyIndex);
153         const Contact& contact =
154             tableView_->getContact(sourceIndex.row());
155
156         // берем ключ из ComboBox
157         QString selectedType = phoneTypeComboBox_->currentText();
158
159         // отображаем номер, соответствующий выбранному типу
160         QString phoneNumber =
161             contact.phoneNumbers_.value(selectedType);
162         mainPhoneInput_->setText(phoneNumber);
163     } else if (!rowIsSelected && !isEditMode_) {
164         clearInputFields();
165     }
166 }
167
168 void MainWindow::onPhoneTypeChanged(const QString& type) {
169     QModelIndex proxyIndex = tableView_->currentIndex();
170     if (!proxyIndex.isValid() || isEditMode_) {

```

```

165     // если ничего не выбрано или мы в режиме редактирования
166     // добавления,
167     // просто очищаем поле, чтобы не было путаницы.
168     mainPhoneInput_->clear();
169     return;
170 }
171
172 // читаем данные из модели для выбранной строки и нового типа
173 // телефона
174 QModelIndex sourceIndex =
175     proxyModel_->mapToSource(proxyIndex);
176 const Contact& contact =
177     tableView_->getContact(sourceIndex.row());
178
179 QString phoneNumber = contact.phoneNumbers_.value(type);
180 mainPhoneInput_->setText(phoneNumber);
181 }
182
183 void MainWindow::clearInputFields() {
184     firstNameInput_->clear();
185     lastNameInput_->clear();
186     middleNameInput_->clear();
187     addressInput_->clear();
188     birthDateInput_->setDate(QDate::currentDate());
189     emailInput_->clear();
190     mainPhoneInput_->clear();
191 }
192
193 Contact MainWindow::getCurrentContactFromForm() const {
194     Contact contact;
195     QString phoneType = phoneTypeComboBox_->currentText();
196
197     contact.firstName_ = firstNameInput_->text().trimmed();
198     contact.lastName_ = lastNameInput_->text().trimmed();
199     contact.middleName_ = middleNameInput_->text().trimmed();
200     contact.adress_ = addressInput_->text().trimmed();
201     contact.birthDate_ = birthDateInput_->date();
202     contact.email_ = emailInput_->text().trimmed();
203     contact.phoneNumbers_[phoneType] =
204         mainPhoneInput_->text().trimmed();
205     return contact;
206 }
207
208 void MainWindow::setEditMode(bool isEditing) {
209     isInEditMode_ = isEditing;
210     if (isInEditMode_) {
211         addButton_->setText("Сохранить");
212         removeButton_->hide();
213         editButton_->hide();
214         cancelButton_->show();
215         tableView_->setEnabled(false);
216     } else {
217         addButton_->setText("Добавить");

```

```

213     removeButton_ ->show();
214     editButton_ ->show();
215     cancelButton_ ->hide();
216     tableView_ ->setEnabled(true);
217     clearInputFields();
218     tableView_ ->clearSelection();
219 }
220 }
221
222 void MainWindow::onSaveButtonClicked() {
223     QString filePath = QFileDialog::getSaveFileName(this,
224         "Сохранить файл", "", "JSON Files (*.json)");
225     if (!filePath.isEmpty()) {
226         if (contactManager_.saveToFile(filePath)) {
227             QMessageBox::information(this, "Сохранение", "Данные
228                 успешно сохранены.");
229         } else {
230             QMessageBox::warning(this, "Ошибка", "Не удалось сохр
231                 анить файл.");
232         }
233     }
234 }
235
236 void MainWindow::onLoadButtonClicked() {
237     QString filePath = QFileDialog::getOpenFileName(this,
238         "Загрузить файл", "", "JSON Files (*.json)");
239     if (!filePath.isEmpty()) {
240         if (contactManager_.loadFromFile(filePath)) {
241             tableModel_ ->resetTable();
242             QMessageBox::information(this, "Загрузка", "Данные ус
243                 пешно загружены.");
244         } else {
245             QMessageBox::warning(this, "Ошибка", "Не удалось загр
246                 узить файл.");
247         }
248     }
249 }
250
251 bool MainWindow::validateContact(const Contact& contact) {
252     bool valid = true;
253     QString errors;
254
255     if (!Validator::isValidName(contact.firstName_)) { errors +=  

256         "Неверное Имя. "; valid = false; }  

257     if (!Validator::isValidName(contact.lastName_)) { errors +=  

258         "Неверная Фамилия. "; valid = false; }  

259     if (!contact.middleName_.isEmpty() &&  

260         !Validator::isValidName(contact.middleName_)) {  

261         errors += "Неверное Отчество. "; valid = false;
262     }  

263     if (!Validator::isValidBirthDate(contact.birthDate_)) {  

264         errors += "Неверная Дата рождения. "; valid = false; }  

265     if (!Validator::isValidEmail(contact.email_)) { errors +=

```

```

256         "Неверный Email. "; valid = false; }
257
258     for (const QString& number : contact.phoneNumbers_.values())
259     {
260         if (!Validator::isValidPhoneNumber(number) &&
261             !number.isEmpty()) {
262             errors += "Неверный Телефон. ";
263             valid = false;
264             break;
265         }
266     }
267
268     if (!valid) {
269         QMessageBox::warning(this, "Ошибка валидации",
270                             "Пожалуйста, исправьте:\n" + errors);
271     }
272     return valid;
273 }
```

Листинг 24: MainWindowLogic.cc

Приложение Ж. Точка входа программы

```

1 // main.cc
2 #include <QApplication>
3 #include "MainWindow.h"
4
5 int main(int argc, char *argv[]) {
6     QApplication app(argc, argv);
7
8     MainWindow window;
9     window.setWindowTitle("Телефонный справочник");
10    window.resize(1000, 600);
11    window.show();
12
13    return app.exec();
14 }
```

Листинг 25: main.cc

Приложение З. Пример файла данных

```

1 [
2 {
3     "address": "Санкт Петербург, Санкт-Петербургская академия
4                 наук",
5     "birthDate": "1856-06-14",
6     "email": "markovs@chain.spb",
7     "firstName": "Андрей",
8     "lastName": "Марков",
```

```
8     "middleName": "Андреевич",
9     "phoneNumbers": {
10        "Домашний": "+7 123 456 78 90"
11    }
12 },
13 {
14   "address": "Москва Охотный Ряд Университет Ломоносовский
15   проспект (МГУ)",
16   "birthDate": "1903-04-25",
17   "email": "kolmogorovs@probability.axioms",
18   "firstName": "Андрей",
19   "lastName": "Колмогоров",
20   "middleName": "Николаевич",
21   "phoneNumbers": {
22     "Домашний": "+7 (777) - 777 77 77"
23   }
24 }
```

Листинг 26: contacts.json