

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.

Содержание

Введение	4
1 Постановка задачи	6
1.1 Общее описание	6
1.2 Обязательные поля для хранения	6
1.3 Проверка вводимых данных	6
1.3.1 Фамилия, Имя, Отчество	6
1.3.2 Телефонный номер	7
1.3.3 Дата рождения	7
1.3.4 Email	7
1.4 Функциональность приложения	7
1.5 Требования к интерфейсу	8
1.6 Требования к хранению данных	8
2 Реализация	9
2.1 Архитектура приложения	9
2.2 Описание классов	10
2.2.1 Структура Contact	10
2.2.2 Класс ContactManager	10
2.2.3 Класс ContactTableModel	12
2.2.4 Класс Validator	14
2.2.5 Класс MainWindow	15
2.3 Механизм сигналов и слотов	17
2.4 Класс QDataWidgetMapper	17
2.5 Класс QSortFilterProxyModel	18
2.6 Алгоритм добавления и редактирования контакта	18
2.7 Алгоритм поиска и фильтрации	19
2.8 Сохранение и загрузка данных	19
3 Тестирование приложения	20
3.1 Начальное состояние приложения	20
3.2 Сценарий 1: Добавление нового контакта	21
3.2.1 Шаг 1: Заполнение формы	21
3.2.2 Шаг 2: Нажатие кнопки «Добавить»	21
3.2.3 Шаг 3: Результат добавления	22
3.3 Сценарий 2: Валидация некорректных данных	22
3.3.1 Пример 1: Некорректное имя	22
3.3.2 Пример 2: Некорректный email	23
3.3.3 Пример 3: Множественные ошибки	23
3.4 Сценарий 3: Редактирование существующего контакта	23
3.4.1 Шаг 1: Выбор контакта	23
3.4.2 Шаг 2: Переход в режим редактирования	24

3.4.3	Шаг 3: Изменение данных и сохранение	24
3.4.4	Шаг 4: Отмена редактирования	25
3.5	Сценарий 4: Управление телефонными номерами	26
3.5.1	Добавление нескольких типов номеров	26
3.5.2	Переключение между типами номеров	26
3.6	Сценарий 5: Удаление контакта	26
3.6.1	Шаг 1: Выбор контакта для удаления	26
3.6.2	Шаг 2: Результат удаления	27
3.7	Сценарий 6: Поиск контактов	27
3.7.1	Поиск по фамилии	27
3.7.2	Поиск по другим полям	28
3.7.3	Очистка поиска	28
3.8	Сценарий 7: Сортировка данных	28
3.8.1	Сортировка по фамилии	28
3.8.2	Сортировка по дате рождения	28
3.9	Сценарий 8: Работа с файлами	29
3.9.1	Сохранение в файл	29
3.9.2	Загрузка из файла	30
3.10	Сценарий 9: Использование календаря для выбора даты	30
3.11	Тестирование регулярных выражений	31
3.12	Результаты тестирования	32
4	Заключение	33
4.1	Реализованная функциональность	33
4.2	Использованные технологии и концепции	34
4.3	Использованные инструменты	35
4.4	Выводы	36
5	Список использованных источников	37
	Приложение	38
	Приложение А. Файл сборки CMakeLists.txt	38
	Приложение Б. Заголовочные файлы	39
	Приложение В. Исходные файлы	41
	Приложение Д. Заголовочный файл MainWindow	47
	Приложение Е. Исходные файлы MainWindow	48
	Приложение Ж. Точка входа программы	58
	Приложение З. Пример файла данных	58

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 Общее описание

Необходимо разработать приложение «Телефонный справочник» на базе фреймворка Qt с использованием модуля Qt Widgets. Приложение должно обеспечивать полный цикл работы с контактными данными: создание, чтение, обновление и удаление записей (CRUD-операции).

1.2 Обязательные поля для хранения

Каждая запись в справочнике должна содержать следующие обязательные поля:

- **Имя** — строка, содержащая имя контакта;
- **Фамилия** — строка, содержащая фамилию контакта;
- **Отчество** — строка, содержащая отчество (может быть пустой);
- **Адрес** — строка с адресом проживания;
- **Дата рождения** — дата в формате день.месяц.год;
- **Email** — адрес электронной почты;
- **Телефонные номера** — коллекция номеров различных типов.

1.3 Проверка вводимых данных

Для обеспечения корректности вводимых данных необходимо реализовать валидацию с использованием регулярных выражений:

1.3.1 Фамилия, Имя, Отчество

- Должны содержать только буквы различных алфавитов (кириллица, латиница), цифры, пробелы и дефисы;
- Должны начинаться с заглавной буквы;
- Не могут начинаться или заканчиваться дефисом;
- Все незначимые пробелы (в начале и конце строки) должны автоматически удаляться.

1.3.2 Телефонный номер

- Должен быть записан в международном формате;
- Может содержать символ +, цифры, пробелы, скобки и дефисы;
- Примеры допустимых форматов:

– +7 812 1234567

– 8(800)123-1212

– +38 (032) 123 11 11

1.3.3 Дата рождения

- Должна быть меньше текущей даты;
- Число месяцев: от 1 до 12;
- Число дней: от 1 до 31 с учётом количества дней в месяце и високосных годов;
- Для ввода даты использовать виджет `QDateEdit` с возможностью выбора через календарь (`QCalendarWidget`).

1.3.4 Email

- Должен содержать имя пользователя, состоящее из латинских букв и цифр;
- Должен содержать символ @, разделяющий имя пользователя и домен;
- Домен состоит из латинских букв и цифр;
- Все незначимые пробелы (включая пробелы до и после @) должны быть удалены.

1.4 Функциональность приложения

Приложение должно обеспечивать следующие возможности:

- Добавление/удаление записей
- Редактирование всех полей
- Сортировка отображаемых данных по указанному полю
- Поиск по нескольким полям

1.5 Требования к интерфейсу

- Для представления данных использовать табличный виджет `QTableView` с поддержкой автоматической сортировки;

1.6 Требования к хранению данных

- Данные должны храниться в виде файла;
- Для чтения/записи файла можно использовать класс `QFile`.

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.

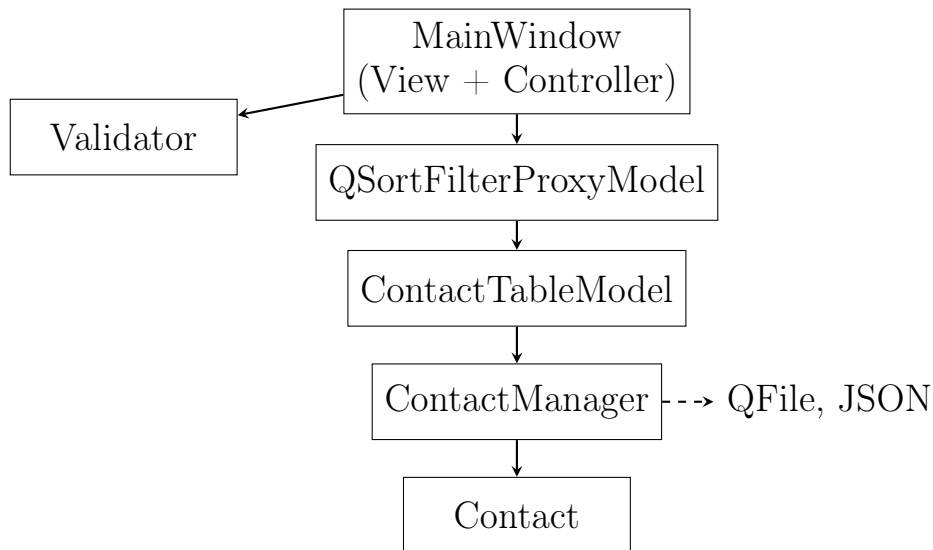


Рис. 1: Архитектура приложения

2.2 Описание классов

2.2.1 Структура Contact

Contact - структура, отвечающая за хранение одной записи контакта.

```
1
2 #pragma once
3
4 #include <QString>
5 #include <QDate>
6 #include <QMap>
7 #include <QUuid>
8
9 /** служит моделью QTable
10 struct Contact {
11     QString firstName_;
12     QString lastName_;
13     QString middleName_;
14     QString adress_;
15     QDate birthDate_;
16     QString email_;
17     QMap<QString, QString> phoneNumbers_;
18 };
```

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.

```

1 #pragma once
2 #include <QList>
3 #include "Contact.h"
4
5 // * класс для управления коллекцией контактов
6 class ContactManager {
7 public:
8     ContactManager() = default;
9     ~ContactManager() = default;
10
11     // * --- CRUD операции ---
12     void addContact(const Contact& contact);
13     void removeContact(int index);
14     void updateContact(int index, const Contact& contact);
15
16     // * --- геттеры, доступ к данным ---
17     const QList<Contact>& getContacts() const;
18     QList<Contact>& getContactsMutable();
19     int contactCount() const;
20     Contact* getContact(int index);
21
22     // * --- работа с данными ---
23     bool loadFromFile(const QString& filePath);
24     bool saveToFile(const QString& filePath) const;
25
26 private:
27     QList<Contact> m_contacts; // основная коллекция для всех кон
        тактов
28 };

```

Листинг 3: ContactManager

Псевдокод метода saveToFile:

Функция saveToFile(filePath):

- создать пустой JSON-массив `contactArray`
- для каждого контакта в коллекции:
 - создать JSON-объект `contactObject`
 - записать все поля контакта в `contactObject`
 - для каждого телефона в `phoneNumbers`:
 - добавить пару (тип → номер) в `phonesObject`
 - добавить `phonesObject` в `contactObject`
 - добавить `contactObject` в `contactArray`
- создать JSON-документ из `contactArray`
- открыть файл по пути `filePath` для записи
- если файл успешно открыт:
 - записать JSON-документ в файл
 - закрыть файл
 - вернуть `true`
- иначе вернуть `false`

Псевдокод метода loadFromFile:

Функция loadFromFile(filePath):

открыть файл по пути `filePath` для чтения
если файл не открыт:
 вернуть `false`
считать весь файл в переменную `data`
закрыть файл
разобрать `data` как JSON-документ
если документ невалидный или не является массивом:
 вернуть `false`
очистить текущую коллекцию контактов
для каждого элемента в JSON-массиве:
 создать новый объект `Contact`
 считать все поля из JSON-объекта
 для каждого телефона в `phonesObject`:
 добавить пару (тип → номер) в `contact.phoneNumbers`
 добавить `contact` в коллекцию
вернуть `true`

2.2.3 Класс ContactTableModel

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

```
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 class ContactTableModel : public QAbstractTableModel {
11     Q_OBJECT
12
13 public:
14     // конструктор принимает указатель на contactManager
15     // (источник данных)
16     explicit ContactTableModel(ContactManager* contactManager,
17                               QObject* parent = nullptr);
18
19     /* --- обязательные методы QAbstractTableModel ---
20
21     // возвращает количество строк (записей Contact)
22     int rowCount(const QModelIndex& parent = QModelIndex())
23         const override;
```

```

21 // возвращает количество столбцов (полей Contact)
22 int columnCount(const QModelIndex& parent = QModelIndex())
    const override;
23 // данные для ячейки (index) в заданной роли (role)
24 QVariant data(const QModelIndex& index, int role =
    Qt::DisplayRole) const override;
25 // возвращает название столбцов
26 QVariant headerData(int section, Qt::Orientation
    orientation, int role = Qt::DisplayRole) const override;
27 // записывает новое value в index и обновляет модель
28 bool setData(const QModelIndex& index, const QVariant&
    value, int role = Qt::EditRole) override;
29 // определяет свойства ячейки
30 Qt::ItemFlags flags(const QModelIndex& index) const override;
31
32 /* --- методы-обертки CRUD операций (синхронизация модели)
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 // * --- вспомогательные методы ---
41 QString normalizeDigits(const QString& s);
42 QString normalizeDigits(const QString& s) const;
43
44 // *статические константы для пользовательских ролей
45 static constexpr int NormalizedPhonesRole = Qt::UserRole + 1;
46 static constexpr int ContactIdRole = Qt::UserRole + 2;
47
48 public slots:
49     // слот для полного сброса и перерисовки всей таблицы
50     void resetTable();
51
52 private:
53     // указатель на источник данных
54     ContactManager* contactManager_;
55
56 };

```

Листинг 4: ContactManager

Механизм синхронизации Model и View:

При изменении данных модель должна явно 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.

Псевдокод метода `data`:

Функция `data(index, role)`:

если индекс невалиден или выходит за границы:

вернуть пустой `QVariant`

получить ссылку на контакт по индексу строки

если роль равна `DisplayRole` или `EditRole`:

в зависимости от номера столбца:

столбец 0: вернуть `firstName`

столбец 1: вернуть `lastName`

столбец 2: вернуть `middleName`

столбец 3: вернуть `address`

столбец 4: вернуть `birthDate`

столбец 5: вернуть `email`

столбец 6: вернуть объединение всех телефонов через запятую

если роль равна `ContactIdRole`:

вернуть `contact.id`

вернуть пустой `QVariant`

2.2.4 Класс `Validator`

`Validator` — утилитарный класс для валидации пользовательского ввода. Все методы статические, так как не зависят от состояния объекта.

```
1 #pragma once
2
3 #include <QString>
4 #include <QDate>
5
6 class Validator {
7 public:
8     static bool isValidName(const QString& name);
9     static bool isValidPhoneNumber(const QString& number);
10    static bool isValidEmail(const QString& email);
11    static bool isValidBirthDate(const QDate& date);
12};
```

Листинг 5: `Validator`

Каждый метод использует класс `QRegularExpression` для проверки соответствия строки заданному шаблону регулярного выражения.

Псевдокод метода `isValidName`:

Функция `isValidName(name)`:

удалить пробелы в начале и конце строки

если строка пустая:

вернуть `false`
если строка начинается или заканчивается дефисом:
вернуть `false`
создать регулярное выражение:
`^[А-ЯА-Z][А-Яа-яА-Zа-z -]*[А-Яа-яА-Zа-z0-9]$`
вернуть результат проверки строки на соответствие выражению

2.2.5 Класс MainWindow

`MainWindow` — главное окно приложения. Наследуется от `QMainWindow` — базового класса Qt для создания главных окон приложений.

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

```
1 #pragma once
2
3 #include <QMainWindow>
4 #include <QSortFilterProxyModel>
5 #include "ContactManager.h"
6 #include "ContactTableModel.h"
7
8 class QPushButton;
9 class QLineEdit;
10 class QDateEdit;
11 class QComboBox;
12 class QVBoxLayout;
13 class QTableView;
14 class QDataWidgetMapper;
15 class QListWidget;
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     // * --- слоты для кнопок (CRUD) ---
26     void onAddButtonClicked();
27     void onRemoveButtonClicked();
28     void onEditButtonClicked();
29     void onSaveButtonClicked();
30     void onLoadButtonClicked();
31     void onCancelButtonClicked();
32
33     // * --- слоты для взаимодействия с таблицей и поиском ---
34     // обработка ввода в поле поиска
35     void onSearchTextChanged(const QString& text);
36     // обработка выбора строки в QTableView
```

```

37     void onSelectionChanged();
38     // выбор телефона
39     void onPhoneTypeChanged(const QString& type);
40
41 private:
42     // статическая константа для пользовательской роли
43     static constexpr int ContactIdRole = Qt::UserRole + 1;
44
45     // * --- инициализация и служебные методы ---
46     void setupUi();
47     void setupMapper();
48     void clearInputFields();
49     Contact getCurrentContactFromForm() const;
50     bool validateContact(const Contact& contact);
51
52     // * --- управление состоянием ---
53     bool isInEditMode_ = false;
54     void setEditingMode(bool isInEditMode);
55
56     // * --- основные компоненты ядра и модели ---
57     ContactManager contactManager_;
58     ContactTableModel* tableModel_;
59     QSortFilterProxyModel* proxyModel_;
60     QTableView* tableView_;
61
62     // * --- виджеты для ввода данных ---
63     QDataWidgetMapper* mapper_;
64
65     QLineEdit* firstNameInput_;
66     QLineEdit* lastNameInput_;
67     QLineEdit* middleNameInput_;
68     QLineEdit* addressInput_;
69     QDateEdit* birthDateInput_;
70     QLineEdit* emailInput_;
71
72     QLineEdit* mainPhoneInput_;
73     QComboBox* phoneTypeComboBox_;
74
75     // * --- кнопки ---
76     QPushButton* addButton_;
77     QPushButton* removeButton_;
78     QPushButton* editButton_;
79     QPushButton* cancelButton_;
80
81     // * --- поиск и сортировка ---
82     QLineEdit* searchInput_;
83
84     // * --- слои ---
85     QVBoxLayout* mainLayout_;
86 };

```

Листинг 6: MainWindow

2.3 Механизм сигналов и слотов

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

Примеры используемых связей:

```
1 // Кнопка "Добавить" -> слот onAddButtonClicked
2 connect(addButton_, &QPushButton::clicked,
3         this, &MainWindow::onAddButtonClicked);
4
5 // Поле поиска -> слот onSearchTextChanged
6 connect(searchInput_, &QLineEdit::textChanged,
7         this, &MainWindow::onSearchTextChanged);
8
9 // Выбор строки в таблице -> слот onSelectionChanged
10 connect(tableView_ ->selectionModel(),
11         &QItemSelectionModel::currentChanged,
12         this, &MainWindow::onSelectionChanged);
```

Листинг 7: Связывание сигналов и слотов

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.

Псевдокод метода onAddButtonClicked:

Функция onAddButtonClicked():

собрать данные из полей формы в объект `contact`

если валидация контакта не пройдена:

вернуться (показать сообщение об ошибке)

если находимся в режиме редактирования:

получить индекс выбранной строки из прокси-модели

преобразовать индекс в исходную модель

получить ссылку на редактируемый контакт

обновить телефон по выбранному типу

вызвать `mapper_ -> submit()` для применения изменений

если успешно:

показать сообщение "Данные успешно отредактированы"

иначе:
показать сообщение об ошибке
иначе (режим добавления):
вызвать `tableModel_->addContact(contact)`
показать сообщение "Контакт успешно добавлен"
сохранить данные в файл `contacts.json`
вызвать `setEditingMode(false)` для возврата в режим просмотра

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.

Псевдокод метода `onSearchTextChanged`:

Функция `onSearchTextChanged(text)`:

вызвать `proxyModel_->setFilterRegularExpression(text)`
прокси-модель автоматически:
проверяет каждую строку на соответствие выражению
скрывает несоответствующие строки
обновляет представление

2.8 Сохранение и загрузка данных

Данные сохраняются в формате JSON с использованием классов `QJsonDocument`, `QJsonObject` и `QJsonArray`.

Структура JSON-файла:

```
1  {
2      "firstName": "Имя",
3      "lastName": "Фамилия",
4      "middleName": "Отчество",
5      "address": "улица дом кв",
6      "birthDate": "2005-10-14",
7      "email": "yourmail@mail.com",
8      "phoneNumbers": {
9          "Рабочий": "+711 111 11 11"
10     }
11 }
```

Листинг 11: Пример `contacts.json`

При запуске приложения автоматически вызывается:

```
1  if (contactManager_.loadFromFile("contacts.json"))
2      tableModel_->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.

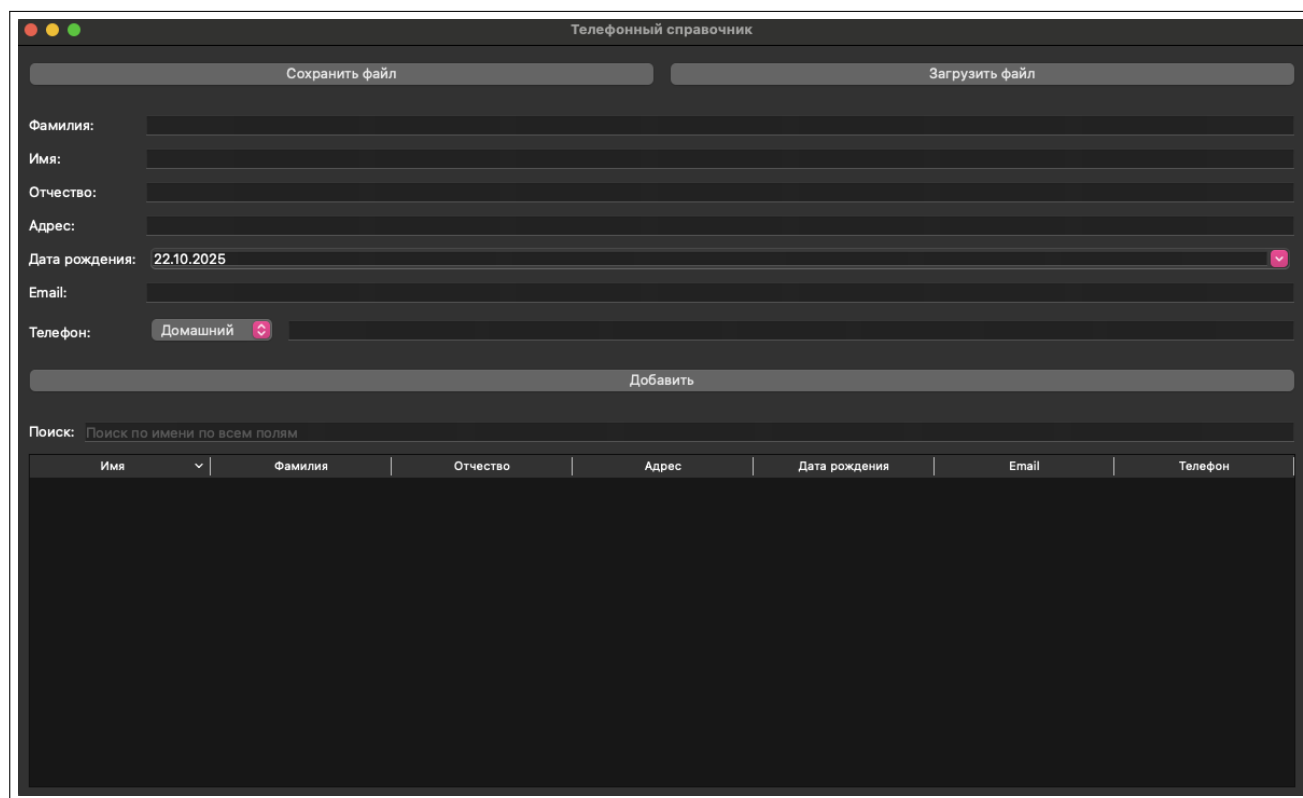


Рис. 2: Начальное состояние приложения

Элементы интерфейса:

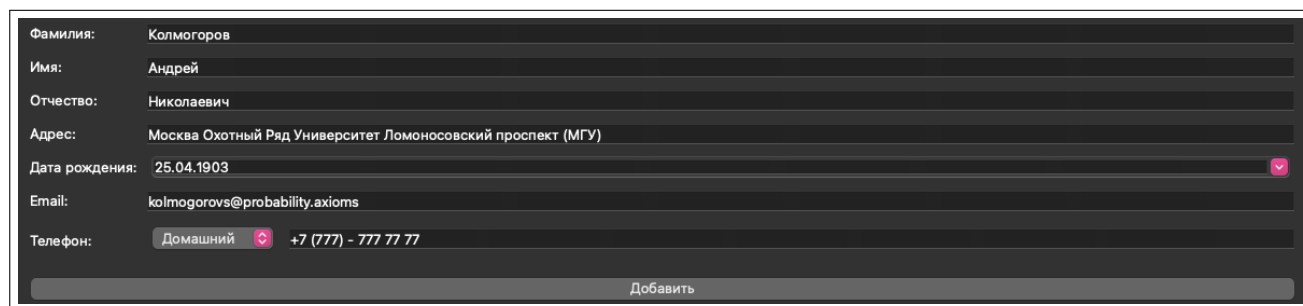
- Форма ввода данных (верхняя часть окна);
- Кнопки управления: "Добавить" "Удалить" "Редактировать";
- Поле поиска;
- Таблица для отображения контактов;
- Кнопки "Сохранить файл" и "Загрузить файл".

3.2 Сценарий 1: Добавление нового контакта

3.2.1 Шаг 1: Заполнение формы

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

- Фамилия: Колмогоров
- Имя: Андрей
- Отчество: Николаевич
- 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 image shows a dark-themed contact form with the following fields and values:

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

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

Рис. 3: Заполнение формы для добавления контакта

3.2.2 Шаг 2: Нажатие кнопки «Добавить»

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

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

3.2.3 Шаг 3: Результат добавления

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

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. 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

Добавить

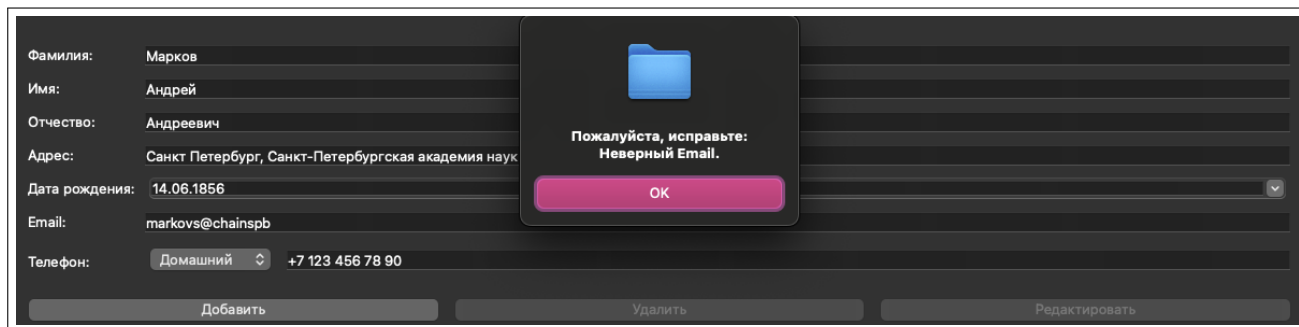
Удалить

Редактировать

Рис. 6: Ошибка валидации имени

3.3.2 Пример 2: Некорректный email

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

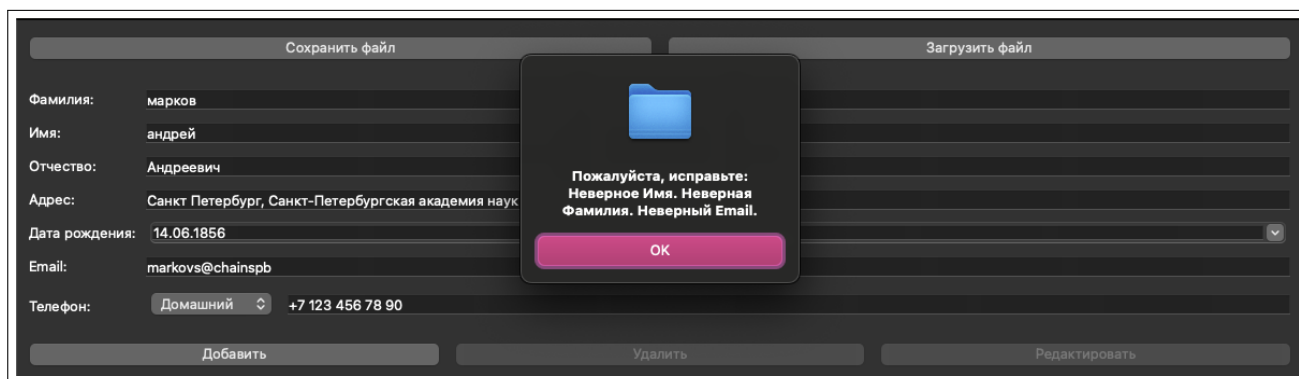


The screenshot shows a contact form with the following fields: Фамилия: Марков, Имя: Андрей, Отчество: Андреевич, Адрес: Санкт Петербург, Санкт-Петербургская академия наук, Дата рождения: 14.06.1856, Email: markovs@chainspb, and Телефон: Домашний +7 123 456 78 90. A modal dialog box is displayed in the center with a blue folder icon and the text: "Пожалуйста, исправьте: Неверный Email." with an "OK" button. At the bottom of the form are three buttons: "Добавить", "Удалить", and "Редактировать".

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).



The screenshot shows the same contact form as in Figure 7. The modal dialog box displays the text: "Пожалуйста, исправьте: Неверное Имя. Неверная Фамилия. Неверный Email." with an "OK" button. The form fields and buttons are the same as in Figure 7.

Рис. 8: Множественные ошибки валидации

3.4 Сценарий 3: Редактирование существующего контакта

3.4.1 Шаг 1: Выбор 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

Добавить Удалить Редактировать

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. 9: Выбор контакта для редактирования

3.4.2 Шаг 2: Переход в режим редактирования

После нажатия кнопки "Редактировать":

- Приложение переходит в режим редактирования (`isInEditMode_ = 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: 25.04.1903

Email: kolmogorovs@probability.axioms

Телефон: Домашний +7 (777) - 777 77 77

Сохранить Отмена

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. 10: Режим редактирования контакта

3.4.3 Шаг 3: Изменение данных и сохранение

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).

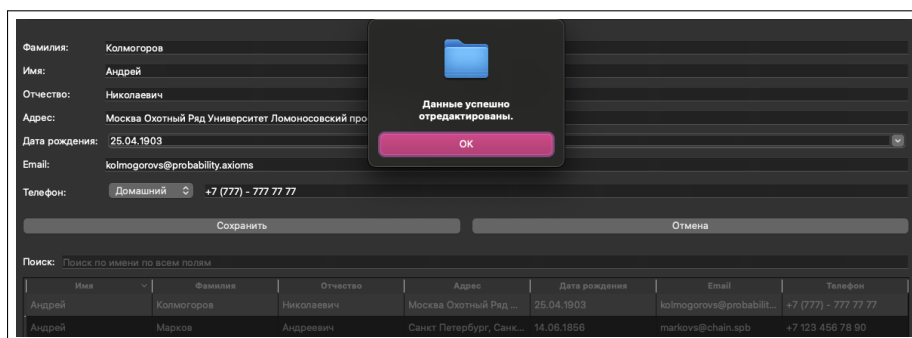


Рис. 11: Успешное редактирование контакта

3.4.4 Шаг 4: Отмена редактирования

При нажатии кнопки "Отмена":

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

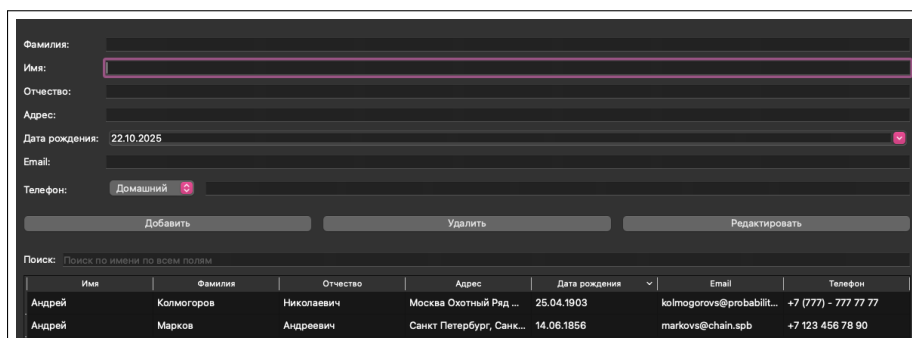


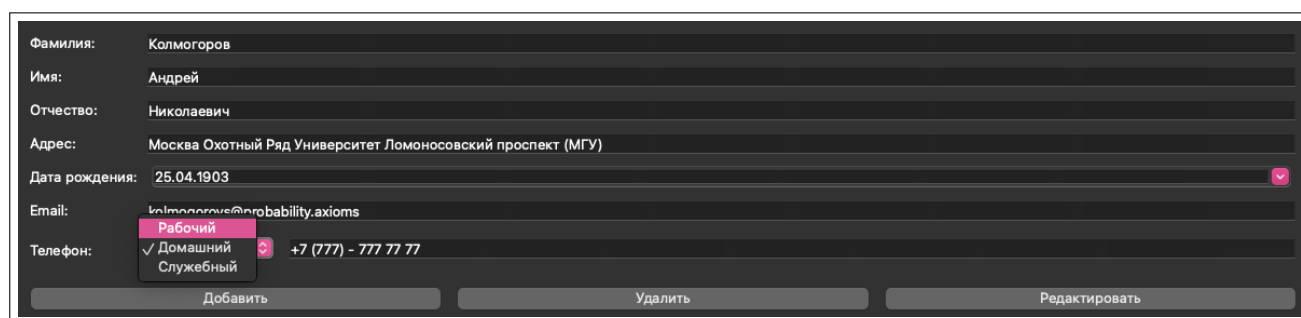
Рис. 12: Отмена редактирования

3.5 Сценарий 4: Управление телефонными номерами

3.5.1 Добавление нескольких типов номеров

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).

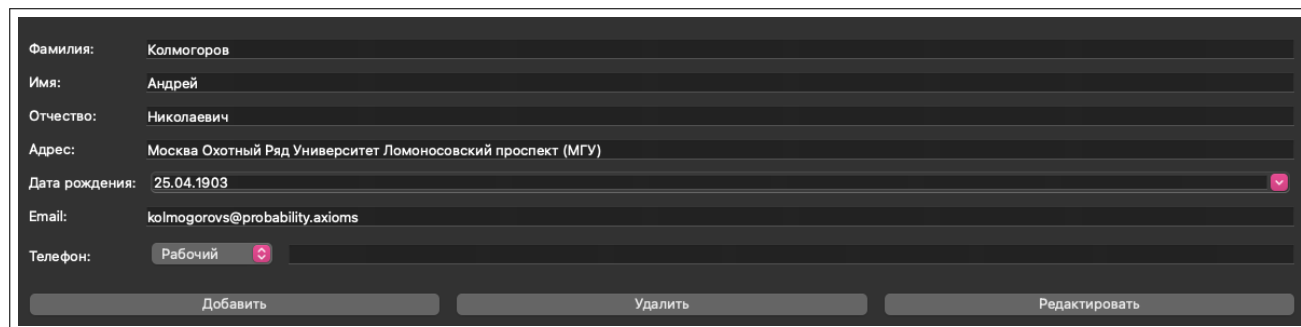


The screenshot shows a contact form with the following fields: Фамилия: Колмогоров, Имя: Андрей, Отчество: Николаевич, Адрес: Москва Охотный Ряд Университет Ломоносовский проспект (МГУ), Дата рождения: 25.04.1903, Email: kolmogorovs@probability.axioms. The 'Телефон:' field has a dropdown menu open, showing three options: 'Рабочий' (highlighted in pink), '✓ Домашний', and 'Служебный'. The phone number '+7 (777) - 777 77 77' is entered next to the dropdown. At the bottom are three buttons: 'Добавить', 'Удалить', and 'Редактировать'.

Рис. 13: Выбор типа телефонного номера

3.5.2 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).



The screenshot shows the same contact form as in Figure 13, but the dropdown menu for 'Телефон:' is now closed and shows 'Рабочий' as the selected option. The phone number field is empty. The buttons 'Добавить', 'Удалить', and 'Редактировать' remain at the bottom.

Рис. 14: Переключение между типами телефонов

3.6 Сценарий 5: Удаление контакта

3.6.1 Шаг 1: Выбор контакта для удаления

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

Добавить

Удалить

Редактировать

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	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

3.6.2 Шаг 2: Результат 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

Добавить

Удалить

Редактировать

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. 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 введенную 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@probabilit...	+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.

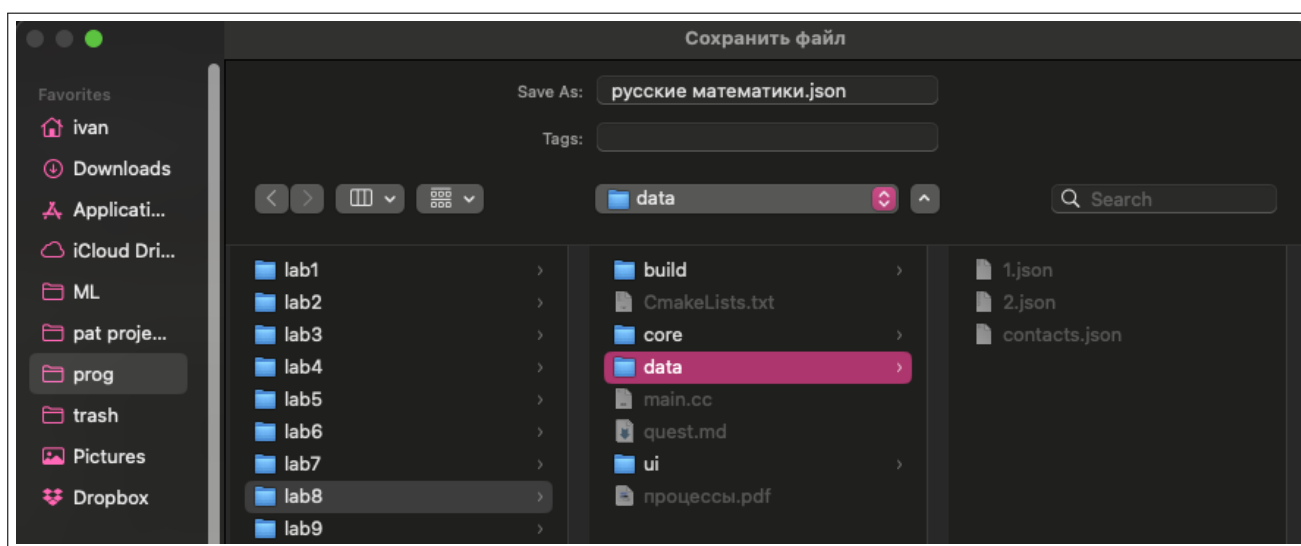


Рис. 22: Диалог сохранения файла

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

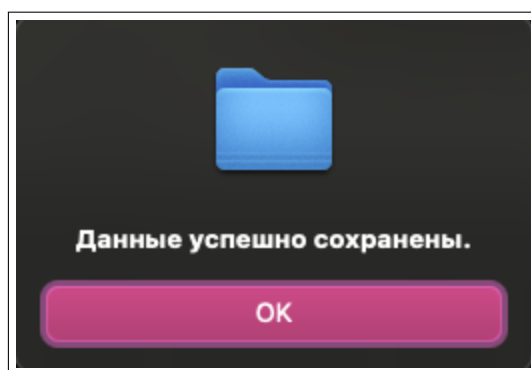


Рис. 23: Успешное сохранение файла

3.9.2 Загрузка из файла

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

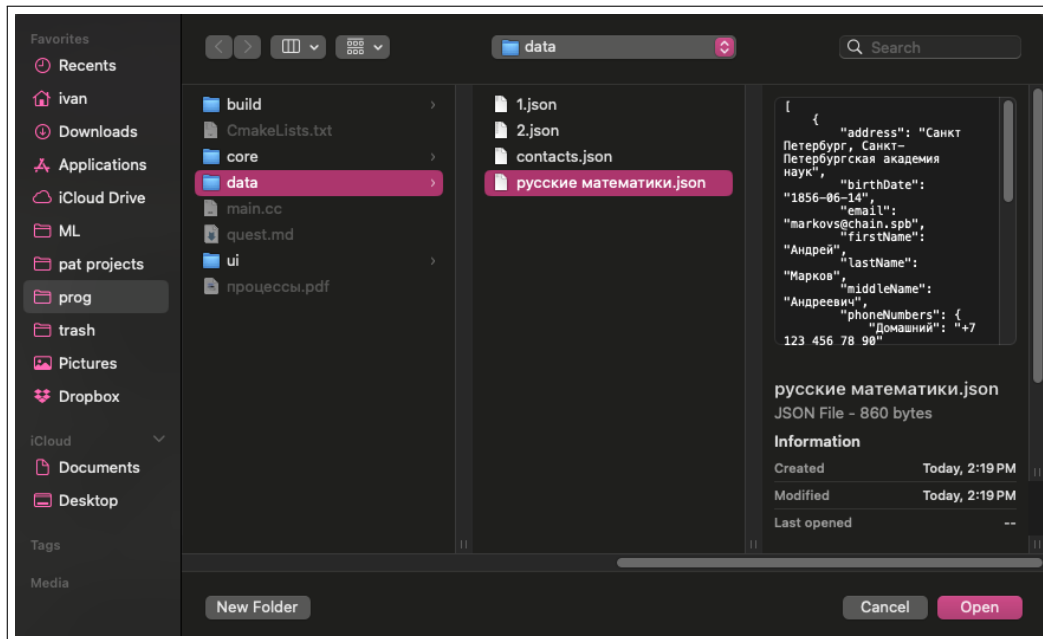


Рис. 24: Диалог загрузки файла

После успешной загрузки:

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

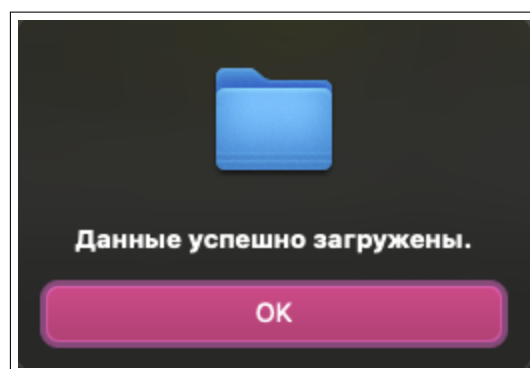


Рис. 25: Успешная загрузка файла

3.10 Сценарий 9: Использование календаря для выбора даты

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

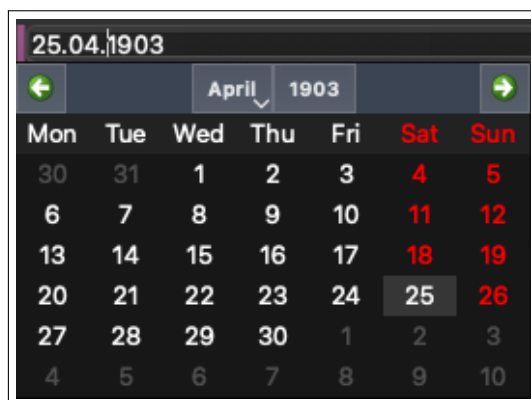


Рис. 26: Виджет календаря для выбора даты

3.11 Тестирование регулярных выражений

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

Таблица 1: 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 компоненты:

1. Полный цикл CRUD-операций:

- Добавление новых контактов с валидацией всех полей;
- Редактирование существующих записей;
- Удаление контактов из справочника;
- Просмотр всех контактов в табличном представлении.

2. Система валидации данных:

- Проверка формата ФИО с помощью регулярных выражений;
- Валидация телефонных номеров;
- Проверка корректности email-адресов;
- Контроль допустимости даты рождения.

3. Поиск и фильтрация:

- Фильтрация записей по введённому запросу;
- Поиск по всем полям таблицы одновременно.

4. Сортировка данных:

- Возможность сортировки по любому столбцу таблицы;
- Поддержка сортировки в обоих направлениях;
- Автоматическая сортировка при клике на заголовок столбца.

5. Работа с файлами:

- Сохранение всех контактов в формате JSON;
- Загрузка контактов из JSON-файла;
- Автоматическая загрузка данных при запуске приложения;

6. Управление телефонными номерами:

- Поддержка нескольких типов номеров;
- Переключение между типами номеров.

7. Пользовательский интерфейс:

- Табличное представление данных с возможностью выбора строки;
- Форма ввода/редактирования с соответствующими виджетами;
- Календарь для выбора даты рождения;
- Контекстно-зависимое состояние кнопок управления.

4.2 Используемые технологии и концепции

В процессе разработки приложения были изучены и применены на практике следующие технологии и концепции:

1. Архитектура Model/View:

- Разделение данных (модель) и представления (вид);
- Использование `QAbstractTableModel` для создания пользовательских моделей;
- Применение `QSortFilterProxyModel` для фильтрации и сортировки без изменения исходных данных;
- Связывание моделей с представлениями через `QTableView`.

2. Механизм сигналов и слотов:

- Связывание сигналов виджетов со слотами-обработчиками;
- Использование макросов `Q_OBJECT` и `signals/slots`.

3. Работа с виджетами Qt:

- Использование `QLineEdit`, `QPushButton`, `QComboBox`;
- Работа с `QTableView` для табличного представления;
- Использование `QFileDialog` для открытия/сохранения файлов.

4. Класс `QDataWidgetMapper`:

- Автоматическая синхронизация данных между моделью и виджетами формы;
- Настройка маппинга столбцов на виджеты;
- Управление режимами отправки данных (ручной/автоматический).

5. Регулярные выражения:

- Использование класса `QRegularExpression`;
- Разработка шаблонов для валидации различных типов данных.

6. Работа с JSON:

- Использование классов `QJsonDocument`, `QJsonObject`, `QJsonArray`;
- Сериализация структур данных в JSON;
- Десериализация JSON-данных в объекты C++.

7. Объектно-ориентированное проектирование:

- Применение принципа единственной ответственности;
- Разделение логики на отдельные классы (`Contact`, `ContactManager`, `Validator`);
- Использование наследования (`QAbstractTableModel`, `QMainWindow`);
- Инкапсуляция данных и методов.

8. Система сборки CMake:

- Конфигурирование проекта Qt с помощью CMake;
- Настройка автоматических инструментов Qt (MOC, UIC, RCC).

4.3 Используемые инструменты

Для разработки приложения использовались следующие инструменты и версии:

- **Фреймворк:** Qt 6.x (модуль Qt Widgets);
- **Язык программирования:** C++17;
- **Система сборки:** CMake 3.16+;
- **Среда разработки:** Visual Studio Code / CLion;
- **Операционная система:** macOS ;
- **Система контроля версий:** Git.

4.4 Выводы

Выполнение данной лабораторной работы позволило получить практический опыт разработки полноценного desktop-приложения с использованием современного фреймворка Qt. Были изучены ключевые концепции разработки GUI-приложений, такие как архитектура Model/View, механизм сигналов и слотов, работа с виджетами и валидация данных.

Все требования задания выполнены в полном объёме.

Полученные знания и навыки могут быть применены в будущем при разработке других приложений.

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 assistant*. URL: gemini.google.com (дата обращения: 21.10.2025).

Приложение

Приложение А. Файл сборки CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.16)
2
3 project(lab8 VERSION 1.0 LANGUAGES CXX)
4
5 # включаем автоматикку qt
6 set(CMAKE_AUTOMOC ON)
7 set(CMAKE_AUTOUIC ON)
8 set(CMAKE_AUTORCC ON)
9
10 find_package(Qt6 REQUIRED COMPONENTS Widgets)
11
12 set(SOURCES
13     main.cc
14     ui/src/MainWindow.cc
15     ui/src/MainWindowUI.cc
16     ui/src/MainWindowLogic.cc
17     core/src/ContactManager.cc
18     core/src/Validator.cc
19     core/src/ContactTableModel.cc
20 )
21
22 set(HEADERS
23     ui/include/MainWindow.h
24     core/include/Contact.h
25     core/include/ContactManager.h
26     core/include/Validator.h
27     core/include/ContactTableModel.h
28 )
29
30 add_executable(${PROJECT_NAME}
31     ${SOURCES}
32     ${HEADERS}
33 )
34
35 target_include_directories(${PROJECT_NAME} PRIVATE
36     ${PROJECT_SOURCE_DIR}/ui/include
37     ${PROJECT_SOURCE_DIR}/core/include
38 )
39
40 target_link_libraries(${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 /** служим 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 contact;

```

```

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
93     QJsonObject phonesObject =
94         contactObject["phoneNumbers"].toObject();
95     for (const QString& key : phonesObject.keys()) {
96         contact.phoneNumbers_[key] =
97             phonesObject[key].toString();
98     }
99     m_contacts.append(contact);
100 }
101 return true;
102 }
103
104 QList<Contact>& ContactManager::getContactsMutable() {
105     return m_contacts;
106 }
107
108 Contact* ContactManager::getContact(int index) {
109     if (index >= 0 && index < m_contacts.size()) {
110         return &m_contacts[index];
111     }
112     return nullptr;
113 }
114
115 int ContactManager::contactCount() const {
116     return m_contacts.size();
117 }

```

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)
    const {
15     Q_UNUSED(parent);
16     return 7;
17 }
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         if (role == ContactIdRole) {
43             return contact.id_;
44         }
45
46         return QVariant();
47     }
48
49     QVariant ContactTableModel::headerData(int section,
50                                             Qt::Orientation
51                                             orientation,
52                                             int role) const {
53         if (role != Qt::DisplayRole || orientation !=
54             Qt::Horizontal) {
55             return QVariant();
56         }
57
58         switch (section) {
59             case 0: return "Имя";
60             case 1: return "Фамилия";
61             case 2: return "Отчество";
62             case 3: return "Адрес";
63             case 4: return "Дата рождения";
64             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 }
115 }

```

```

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

```

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-Яa-z][A-Яa-zA-Za-z -]*[A-Яa-zA-Za-z0-9]$");
15     return re.match(trimmedName).hasMatch();
16 }
17
18 bool Validator::isValidPhoneNumber(const QString& number) {
19     QRegularExpression re("^((\\+)?[\\s\\d()-]+)$");
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 }
31

```

```

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     tableModel_ = new ContactTableModel(&contactManager_, this);
25
26     proxyModel_ = new QSortFilterProxyModel(this);
27     proxyModel_ -> setSourceModel(tableModel_);
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         tableModel_ -> 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 ниче
77 го не выбрано
78 editButton_ ->setEnabled(false); // нельзя редактировать пок
79 а ничего не выбрано
80 cancelButton_ ->hide(); // скрыта до начала редакти
81 рования/добавления
82
83 buttonLayout->addWidget(addButton_);
84 buttonLayout->addWidget(removeButton_);
85 buttonLayout->addWidget(editButton_);
86 buttonLayout->addWidget(cancelButton_);
87 mainLayout_ ->addLayout(buttonLayout);
88
89 // * 3 секция поиска (Horizontal Layout)
90 QHBoxLayout* searchLayout = new QHBoxLayout();
91
92 // поиск
93 searchInput_ = new QLineEdit();

```

```

91     searchInput_ ->setPlaceholderText("Поиск по имени по всем поля
        м");
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::Str
        // растянуть столбцы
100    tableView_ ->setSelectionBehavior(QAbstractItemView::SelectRows);
        // выделение только целых строк
101    tableView_ ->setSelectionMode(QAbstractItemView::SingleSelection);
        // только одна строка может быть выбрана
102    tableView_ ->setEditTriggers(QAbstractItemView::NoEditTriggers);
        // запрещаем редактирование в самой таблице
103
104    mainLayout_ ->addWidget(tableView_);
105 }

```

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
        ранному типу
36     Contact& contactToEdit =
        contactManager_.getContactsMutable()[sourceIndex.row()];
37     contactToEdit.phoneNumbers_[selectedType] = newNumber;
38
39     // 2 --- авт обновление остальных полей
40     // используем маппер для записи данных из формы обратно в
        Модель
41     if (mapper_ ->submit()) {
42         QMessageBox::information(this, "Успех", "Данные успеш
            но отредактированы.");
43     } else {
44         // ошибка submit обычно означает проблему с типом дан
            ных
45         QMessageBox::warning(this, "Ошибка", "Не удалось сохр
            анить изменения.");
46         return;
47     }
48 } else {
49     // * режим добавления:
50     // маппер не нужен. используем прямую вставку через модел
        ь
51     tableModel_ ->addContact(contact);
52     QMessageBox::information(this, "Успех", "Контакт успешно
        добавлен.");
53 }
54
55 // общие действия
56 contactManager_.saveToFile("../data/contacts.json"); // со
        храняем на диск
57 setEditingMode(false); // во
        звращаемся в режим просмотра
58 }
59
60 // слот для кнопки "Удалить"
61 void MainWindow::onRemoveButtonClicked() {
62     // получаем индекс выбранной строки
63     QModelIndex proxyIndex = tableView_ ->currentIndex();
64     if (!proxyIndex.isValid()) {
65         QMessageBox::information(this, "Удаление", "Пожалуйста, в
            ыберите запись для удаления.");
66         return;
67     }
68
69     // преобразуем индекс из Proxy-модели в индекс исходной модел
        и
70     int sourceRow = proxyModel_ ->mapToSource(proxyIndex).row();

```

```

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

```

```

118 // обычный текстовый поиск по DisplayRole
119 proxyModel_ ->setFilterRole(Qt::DisplayRole);
120 proxyModel_ ->setFilterRegularExpression(QRegularExpression(QRegul
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 (!isInEditMode_) {
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         // гике setEditingMode
142         removeButton_ ->setVisible(false);
143         editButton_ ->setVisible(false);
144     }
145
146     if (rowIsSelected && !isInEditMode_) {
147         // синхронизируем мAPPER 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             tableModel_ ->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 && !isInEditMode_) {
164         clearInputFields();
165     }
166 }
167
168 void MainWindow::onPhoneTypeChanged(const QString& type) {
169     QModelIndex proxyIndex = tableView_ ->currentIndex();
170     if (!proxyIndex.isValid() || isInEditMode_) {

```



```

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

```

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",
9     }
10 ]

```

```

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